개념

변수 유도 기법 (Variab결e-directed techniques)은 p 개의 원 변수 들의 상관 관계(공분산 행렬이나 상관계수 행렬)를 이용하여 주성분 변수(원 변수의 선형결합)를 도출하여 80% 규칙에 의해 p차원을 2~4개의 주성분 차원으로 축약하는 주성분 분석과 원 변수를 요인(factor)이라는 잠재 변인을 이용하여 상호 배타적으로 그룹화하는 방법인 요인분석으로 나눌 수 있다.

판별 분석(Discriminant Analysis)은 군집 분석(Clustering Analysis)과 함께 개체들에 대해 측정된 특성(변수) 값을 이용하여 개체를 판별하는 식을 유도하여 새로운 개체의 집단을 판별하거나 개체의 유사성을 계산하여 유사한 개체끼리 군집화 하는 개체 유도기법(individual directed techniques)이다. 집단의 범주를 판별하는 것은 체에 대한 집단을 예측하는 것과 동일하므로 ML에서는 판별규칙=예측모형으로 예측으로 간주된다.

측정 변수

개체 $\begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1p} \\ x_{21} & x_{22} & \cdots & x_{2p} \\ & \cdots\cdots & & \\ x_{n1} & x_{n2} & \cdots & x_{np} \end{bmatrix}$

개체 분류

- 군집 분석: 원 측정변수들을 이용하여 개체간 거리(유사성)를 구하여 개체를 굽집으로 분류한다.
- 판별 분석: 원 측정변수들의 선형 결합 형태의 판별식을 구하고 이를 이용하여 새로운 개체의 집단을 판별한다. 집단을 판별은 집단 예측으로 간주하여 ML에서는 예측모형의 한 방법이다.

변수 축약

- 주성분 분석: 측정변수들의 공분산(상관) 구조관계를 이용하여 원 변수의 선형결합으로 주성분변수를 유도하고, 이를 이용하여 개체들을 저차원 공간에 시각화 하고 정보를 얻는다. 변수의 개수 p를 원 변수 변동 설명 기여율 80% 기준 저차원으로 축약한다.
- SVD : 고유값 분해방법을 이용하여 $n \times p$ 차원의 행과 열의 차원을 동시에 축약한다.

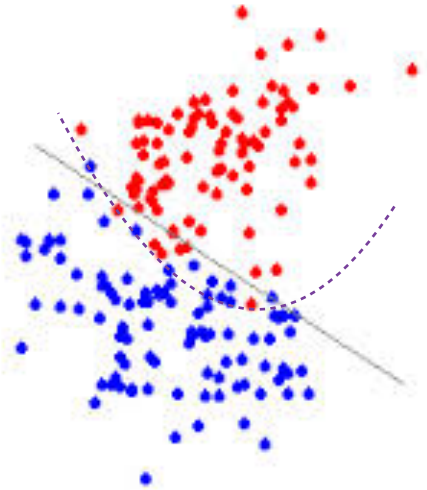
군집분석과 판별분석

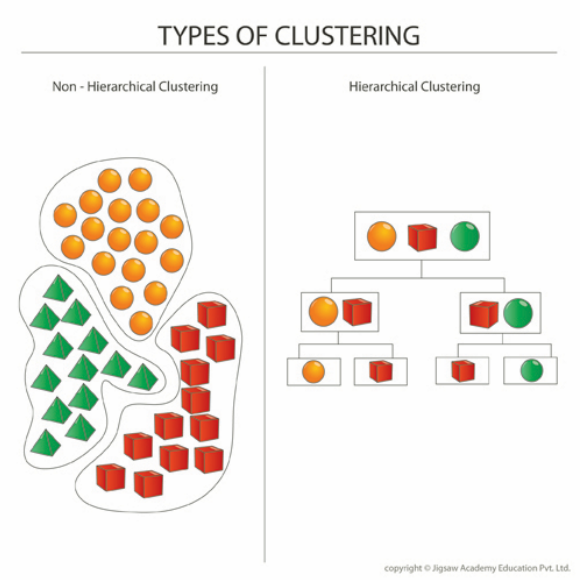
H대학교 졸업생 40명에 대해 평점, 비만도(=키/몸무게: 외모), 영어 성적, 자격증 개수, 원서 지원 회수, 가족 총 연소득 (재정 능력), 친구 수(사교력 측정) 조사하였다고 하자. 측정된 7개의 변수들을 활용하여 졸업생 간 유사성을 측정하여(일반적으로 유크리디안 거리) 유사성이 높은 개체끼리 상호 배타적으로 분류하는 것을 군집분석이라 한다. 군집분석의 핵심적 단계는 유사성 계산, 유사성에 의한 집단으로 분류해 가는 기준, 군집의 개수 결정, 개체 군집에 대한 적절한 이름 부여이다.

위의 예제에서 졸업생 40명에 대하여 졸업 후 6개월 이전에 취업여부를 조사하였다고 하자. 측정된 7개 변수를 이용하여 취업여부(2개의 집단)를 판별하는 판별규칙(판별함수)을 도출하여 새로운 개체(졸업생)에 대하여 졸업 후 6개월 이전에 취업을 할 수 있는지를 판별하는 기법을 판별분석이라 한다.

판별분석은 개체의 그룹이 분류 전에 조사되어 있고 그룹 변수와 측정 변수들을 이용하여 개체의 그룹을 판별하는데 적절한 판별식을 구하고 이를 이용하여 새로운 개체를 분류한다.

좌측 판별분석은 2개의 집단을 판별하는 직선 판별식, 2차 판별식을 그린 예제이며, 우측 군집분석은 개체를 3개의 군집으로 나눈 것으로 비계층적 방법과 계층적 방법을 적용한 예이다.

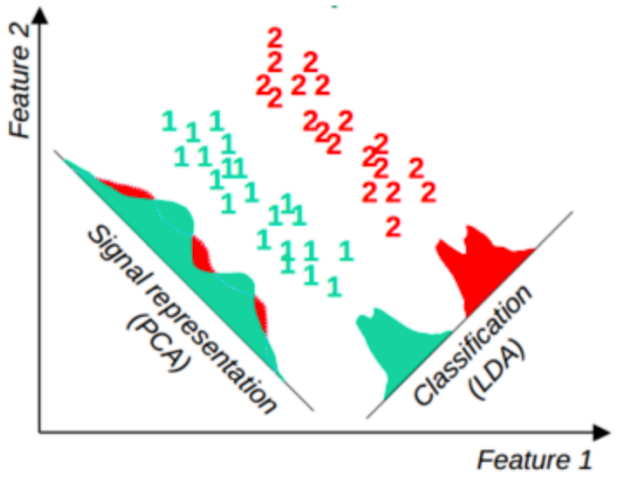




판별분석 군집분석 (그림) 인터넷 구글링 이미지

군집분석에서는 개체의 그룹에 대한 정보 없이 (사전에는 🔴, △, ❎구분이 없이 동일하나 분석 후 나누어짐) 유사성이 가까운 개체들끼리 계층적으로 묶어 가거나 군집의 개수를 정하여 군집의 중심점을 이용하여 개체를 군집화 하는 방법이다. 판별분석은 자료 수집 시 이미 그룹이 나누어져 있어(🔴, 🔵) 이를 가장 잘 판별하는 판별규칙을 도출하여 새로운 개체의 군집을 판별하는 방법이다. 곡선의 판별규칙을 도출하는 것은 쉽지 않다.

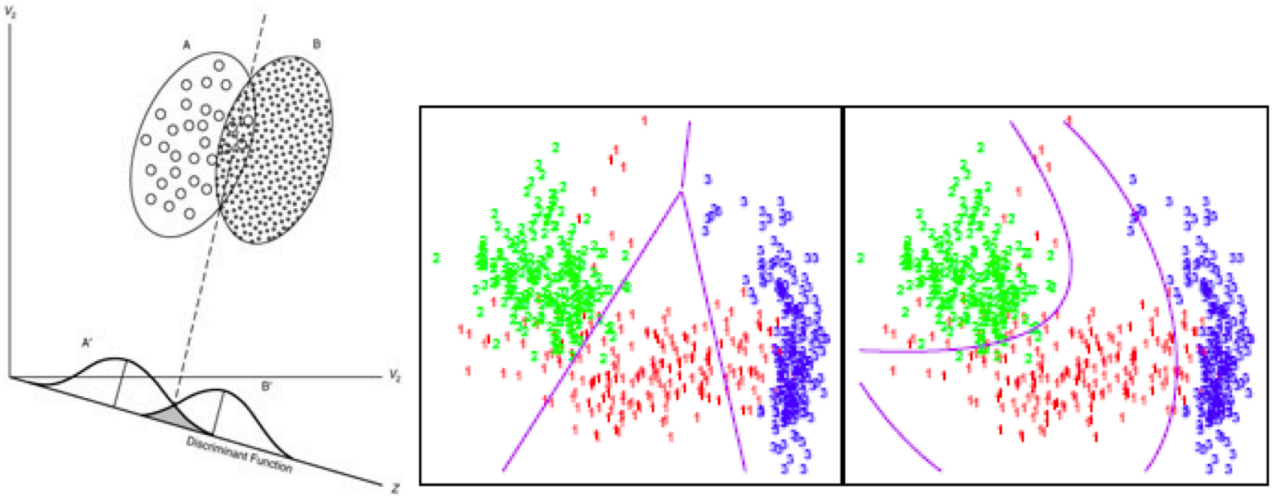
주성분분석과 판별분석비교



- 주성분 분석은 축의 회전을 통하여 변수들이 독립 (상관계수 0)이 되도록 하여 저차원 공간에 표현하는 방법이다. => 공분산을 최대화 하는 변환
- 판별분석은 개체를 분류하는 판별규칙에 의해 서로 개체의 집단을 판별 - 집단간 거리 차이를 최대화 하는 변환

판별 규칙 = 예측모형

판별규칙은 모집단 그룹을 정확하게(오분류 비율 최소화) 하는 '판별' 변수의 함수식으로 판별규칙을 설정 한다. 아래 그림은 인터넷 구글링 결과 얻은 그림이다.



[왼쪽] 2개 모집단, 판별변수 (y, z)의 함수식으로 점선 직선=판별규칙을 얻었음. 아래 히스토그램의 빗금 친부분 중(B모집단의판별규칙왼쪽꼬리빗금친확률)=B모집단에속한개체를판별규칙에의해A집 단으로 분류한 오분류 확률, (A 모집단의 판별규칙 오른쪽 꼬리 빗금친 확률)=A 모집단에 속한 개체를 판 별규칙에 의해 B집단으로 분류한 오분류 확률

[오른쪽] 왼쪽은 직선 판별규칙이고 오른쪽은 다항식 판별규칙임. 현행 산점도를 보면 다항식 판별규칙이 오분류가 적어 보임. 그러나 실제 이차식까지만 판별규칙이 가능, 실제 판별규칙의 선택은 오분류 표를 활용하여 최적 판별 규칙을 얻는다.

판별예측모형 평가 방법

오분류 비율 misclassification ratio

마취과 의사는 심장 수술에 마취가 안전한지 알아보기 위하여 나이, 혈압, 몸무게 등을 조사하고 마취 후 안전 여부(그룹)를 조사하였다고 하자. (나이, 혈압, 몸무게)는 판별(예측)변수, (마취가능=음성, 마취위험=양성)으로 하여 판별한 결과 표이다. **양성은 관심 집단(성공)이며 통계적 가설검정에서는 대립가설이다.**

판별 (예측) ➠ 실제(모집단) ⬇	마취 안전 (negative) **귀무가설 채택**	마취 위험 (positive) **귀무가설 기각**
마취 안전 (negative) **귀무가설 참**	정분류 true negative **TN** **specifivity 특이도**	오분류① **FP** false positive 거짓 양성 **1종 오류 α**
마취 위험 (positive) **대립가설 참**	오분류② **FN** false negative 거짓 음성 **2종 오류 $1-\beta$**	정분류 true positive **TP** **sensitivity(민감도)** **검정력 test power**

positive 양성 - 문제 있음, 감염, 마취 위험

개체의 집단이 2개인 경우 오분류는 2가지 경우가 생긴다. 위의 예를 살펴보면 (1)마취를 해도 괜찮은 환자를 마취하면 안 되는 환자로 분류하거나(오분류①) (2)마취를 해서 안 되는 환자를 마취해도 괜찮은 환자로 분류하는 (오분류②) 잘못을 저지르게 된다. 이 예제에서 오분류①이 발생하면 환자에게 고통을 주거나 병원 수입이 줄어들게 되고 오분류②는 의료 사고로 이어질 수 있으므로 오분류②에 의해 발생하는 비용이 훨씬 크다. 일반적으로 오분류 비용 계산이 어려우므로 오분류 비용은 동일하다고 가정한다. 그러므로 판별 분석은 오분류를 최소화 할 수 있는 판별식을 구하는 것이 주목적이다. **오분류 비율 상황은 가설검정과 동일하다.**

민감도 Sensitivity

양성 개체를 양성으로 예측하는 정분류 비율 : $sensitivity = \dfrac{TP}{TP+FN}$

특이도 Specificity

음성 개체를 음성으로 예측하는 정분류 비율 : $specificity = \dfrac{TN}{TP+FP}$

양성 예측도 Positive Prediction Value

양성으로 예측된 개체가 참 양성일 비율 : $PPV = \dfrac{TP}{TP+FP}$

음성 예측도 Negative Prediction Value

음성으로 예측된 개체가 참 음성일 비율 : $NPV = \dfrac{TN}{TN+FN}$

예측 정확도 Prediction Accuracy

양성 개체를 양성, 음성 개체를 음성으로 판별하는 비율 : $Accuracy = \frac{TP + TN}{TP + FP + TN + FN}$

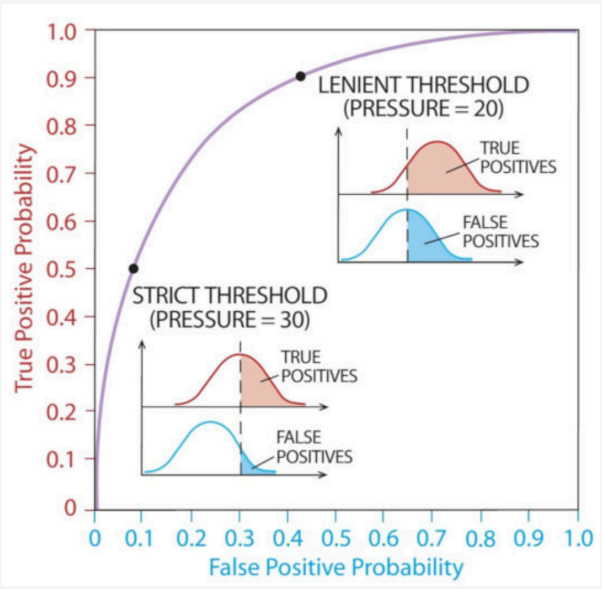
ROC and AUC

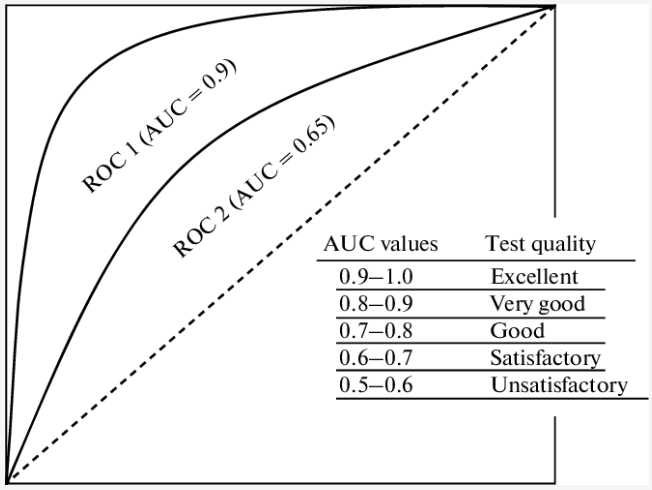
Receiver Operating Characteristic curve : 판별규칙, 예측모형의 정확도를 시각화

Area Under the Curve : ROC 커브의 하단 면적으로 예측모형의 신뢰도의 척도

- 면적이 70% 이상이어야 예측 정확도(good)가 좋다고 할 수 있다.

- X 축 : 음성을 양성으로 오분류 비율, **1-특이도,** 음성개체를 양성으로 예측하는 오분류
- Y 축 : 양성을 양성으로 참분류 비율, **민감도,** 양성개체를 양성으로 예측하는 정분류
- 분류기준=양성으로 판단하는 예측확률(양성으로 판단될 확률) 값을 0에서 1까지 변동시키면서 특이도와 민감도를 계산하여 그래프에 그린다.
- 양성을 판단하는 예측확률이 0인 경우 모든 개체는 양성으로 예측되어 민감도 1이고 1-특이도=0이다. 원점
- 양성을 판단하는 예측확률이 0인 경우 모든 개체는 양성으로 예측되어 민감도 1이고 1-특이도=0이다. 원점





ROC 가설검정과 동일하다.

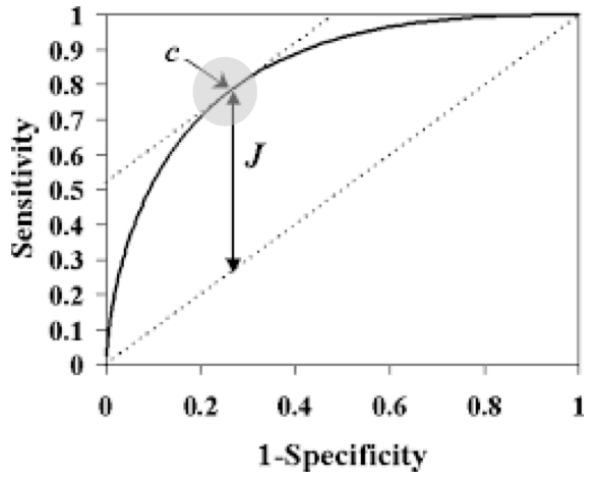
귀무가설=음성, 대립가설=양성 positive, 1종 오류 : 거짓 양성, 2종 오류 : 거짓 음성

검정력 : 양성 개체를 양성으로 예측

양성을 판별하는 기준 확률을 움직임에 민감도, 1-특이도 변동 <=> 대립가설 모수 크기 변동에 따라 1종 오류, 2종 오류 풍선효과처럼 변동

최적 기준값 설정방법 Youden's J statistic

AUC는 판별규칙의 정확도를 측정하는 방법이고 판별규칙에 의해 성공으로 판별(예측)하는 최적 (사후)확률 값은 Youden 방법에 의해 결정된다.



Youden J 통계량이 가장 큰 곳의 값을 기준 값으로 함 - 다음 그림의 원의 c값을 만들어내 기준값을 선택하면 된다.

Youden, W.J. (1950). "Index for rating diagnostic tests". Cancer. **3**: 32–35

오분류 계산방법

Re-substitution 규칙

수집된 데이터로부터 얻은 판별식을 원 데이터에 적용하여 개체를 분류하여 오분류 비율을 구하는 것으로 정분류 비율이 높게 추정될(overestimate) 가능성이 있어 거의 사용하지 않는다.

테스트 데이터 이용

데이터를 양분하여 한 개체 그룹으로부터 판별식을 유도하고, 이 판별식을 사용하여 다른 그룹의 개체를 분류하여 오분류 비율을 추정한다. 표본 자료의 1/2만 사용하여 판별식을 구하므로 모집단 분류에 적합한 판별식을 얻을 가능성이 낮고 데이터를 많이 수집해야 한다는 단점으로 인하여 이 방법 역시 사용 빈도가 낮다.

Cross-validation 추정법

Lachenbrush(1968)가 제안한 방법으로 가장 널리 사용된다. 첫 번째 개체 하나를 제외하고 판별식을 구하여 그 개체를 분류하고, 첫 번째 개체를 다시 넣고 두 번째 개체를 제외하고 판별식을 구한 후 두 번째 개체를 분류하고…… 이렇게 하여 오분류 비율을 추정한다. 이 방법을 Jackknife 방법이라고도 한다.

두 개 모집단 개요

상황

다변량 정규분포를 따르는 2개의 모집단이 있다고 가정하자. 예측(판별에 활용되는 변수의 개수는 p개이고 모두 측정형(정량형)이라 하자. 두 집단의 평균과 분산은 서로 상이하다.

판별 측정변수 벡터 : $\underline{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_p \end{pmatrix} \sim (\underline{\mu}, \Sigma)$

모집단 1 : $\pi_1 \sim f_1(\underline{x}) \sim MN(\underline{\mu}_1, \Sigma_1)$, 모집단 2 : $\pi_2 \sim f_2(\underline{x}) \sim MN(\underline{\mu}_2, \Sigma_2)$

각 모집단으로부터 (n_1, n_2) 크기의 표본 개체를 추출하여 판별 측정변수의 관측값을 조사했다고 하자. 각 표본의 평균벡터를 $(\underline{x}_1, \underline{x}_2)$, 공분산행렬을 (S_1, S_2) 이라 하자.

예제 데이터

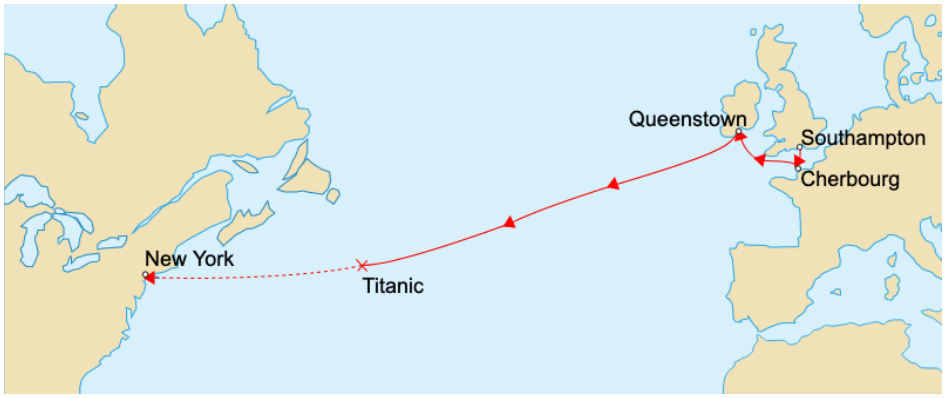
타이타닉 탐승객 정보와 생존여부(1=survived, 0=not)에 대한 데이터이다. **(N=1309)**
[측정형 예측변수] 나이, 탑승요금, (sibsp+parch) 동반승객 수
[범주형 예측변수] 탑승권 등급, 출항 항구, 성별

```
import pandas as pd
df=pd.read_excel('http://biostat.mc.vanderbilt.edu/wiki/pub/Main/DataSets/titanic3.xls')
df.info()
```

```
RangeIndex: 1309 entries, 0 to 1308
Data columns (total 14 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   pclass     1309 non-null   int64
 1   survived   1309 non-null   int64
 2   name       1309 non-null   object
 3   sex        1309 non-null   object
 4   age        1046 non-null   float64
 5   sibsp      1309 non-null   int64
 6   parch      1309 non-null   int64
 7   ticket     1309 non-null   object
 8   fare       1308 non-null   float64
 9   cabin      295 non-null    object
 10  embarked   1307 non-null   object
 11  boat       486 non-null    object
 12  body       121 non-null    float64
 13  home.dest  745 non-null    object
```

- Pclass : 탑승권 등급, 1(upper), 2(middle), 3(lower)
- Sibsp : 동반 탑승한 형제자매, 배우자 수
- Parch : 탑승한 부모자녀 수
- Fare : 탑승권 지불요금
- Cabin : 선실 명
- Embarked : 출항 항구 : C = Cherbourg, Q = Queenstown, S = Southampton(출발지)
- Boat : 탑승한 구명보트 번호
- Body : 사망 인식번호, 번호 없으면 실종

```
df=pd.read_csv('http://wolfpack.hnu.ac.kr/Stat_Notes/adv_stat/DATA/titanic_original.csv') []
```



데이터 변환

예측모형의 집단변수는 (0, 1=성공, 생존) 이진형 숫자 변수로 활용하는 것이 적절함
'http://wolfpack.hnu.ac.kr/Stat_Notes/adv_stat/DATA/titanic_original.csv'#df.dropna(subset=['survived'],inplace=True) 에서 데이터를 불러오면 survuved 데이터는 float으로 되어 있음

```
#df['survived']=df['survived'].astype(int) #integer로 변환
```

```
df.dropna(subset=['survived'],inplace=True) #결측치 제외
df['survived'].mean()
```

```
0.3819709702062643
```

1인 성공, 0이 실패이므로 생존율=평균이므로 38.2% 생존율

범주형 예측변수별 생존율 계산

생존율 : 여성 탑승객 72.7%, 1등급 승객 61.9%, ,Chersburg 항구 출발지 승객 55.6%

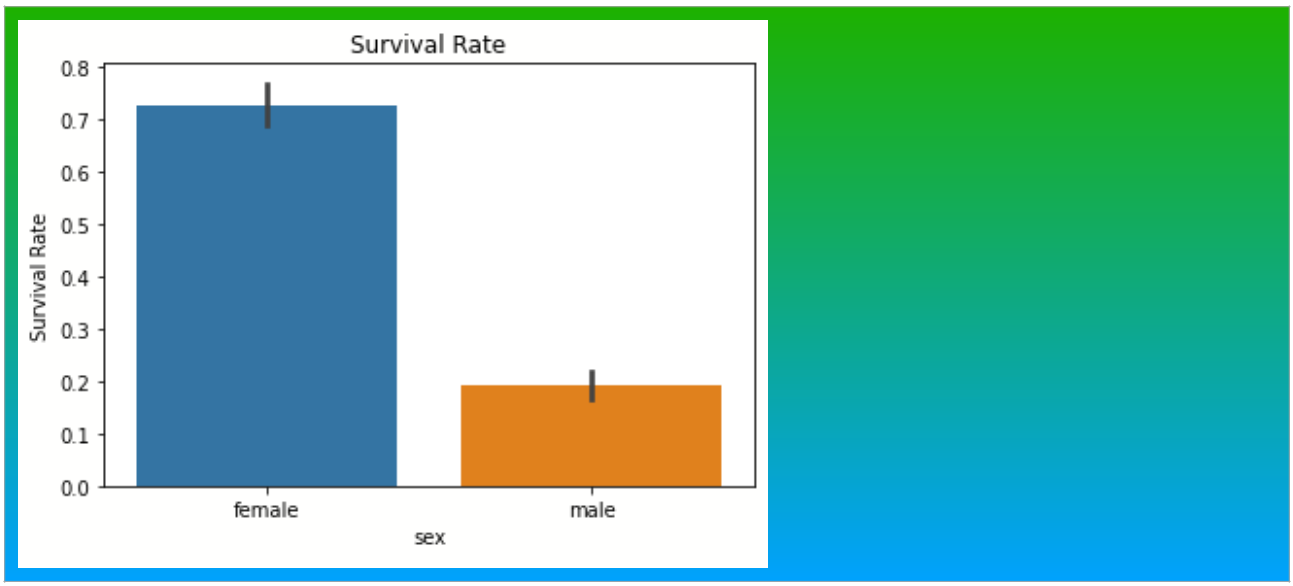
```
titanic=df
titanic.groupby('sex').mean()['survived']
```

```
sex
female    0.727468
male      0.190985
```

```
pclass
1.0    0.619195
2.0    0.429603
3.0    0.255289
```

```
embarked
C    0.555556
Q    0.357724
S    0.332604
```

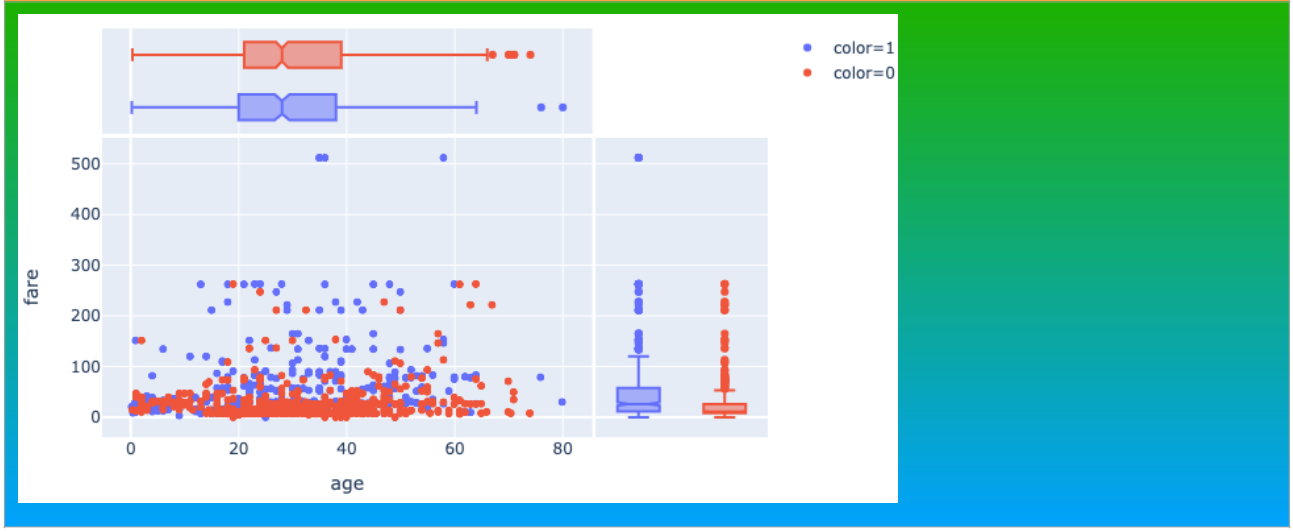
```
import seaborn as sns
sns.barplot(x="sex",y="survived",data=titanic)
plt.title("Survival Rate")
plt.ylabel("Survival Rate")
plt.show()
```



판별변수 2개 : 객실 요금, 승객 나이

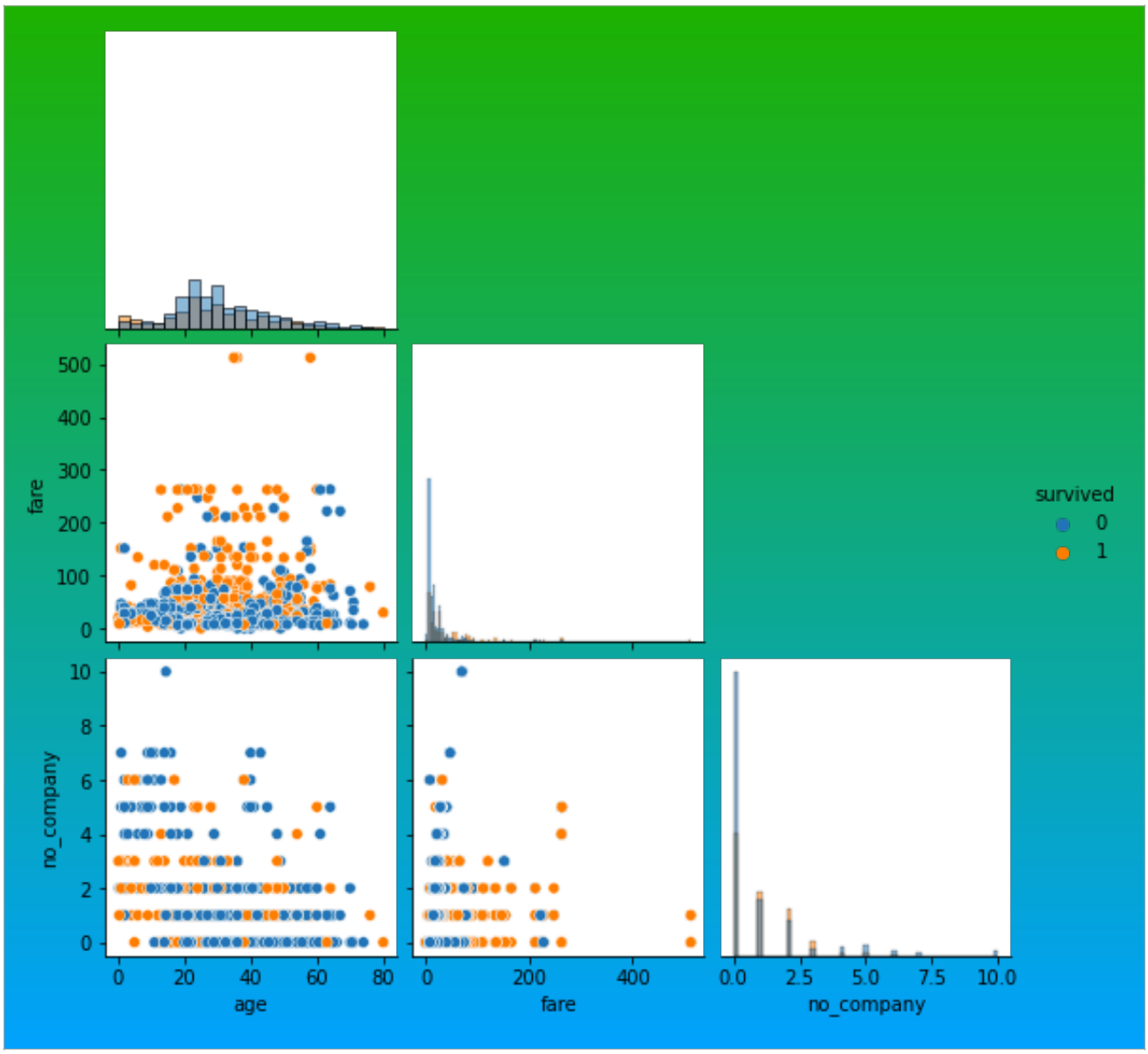
`titanic_da.dropna() : Nan 결측치 데이터 제외`

```
titanic_da=titanic[['age','fare','survived']]
titanic_da=titanic_da.dropna()
titanic_da
import plotly.express as px
fig = px.scatter(titanic_da,x="age",y="fare",
color=titanic_da["survived"].astype(str),marginal_y="box",margina
l_x="box")
fig.show()
```



판별변수 3개 : 객실 요금, 승객 나이, 동반승객 수

```
import seaborn as sns
df=titanic[['age','fare','no_company','survived']]
ax=sns.pairplot(df,hue='survived',corner=True,diag_kind="hist")
```


fare
no_company
age
fare
no_company
survived
0
1
0
20
40
60
80
0
200
400
0.0
2.5
5.0
7.5
10.0
0
100
200
300
400
500
0
2
4
6
8
10

가정

- 집단은 2개이다.
- 예측(판별)변수는 측정형이며 정규분포를 따른다. 적어도 좌우 대칭의 분포를 따른다.

(간단한 정규변환 normal transformation 방식

우로 치우침 : $\sqrt{X} \rightarrow ln(X) \rightarrow \frac{1}{X}$ -> 치우침의 정도가 극심

좌로 치우침 : $\sqrt{max(X+1)-X} \rightarrow ln(max(X+1)-X) \rightarrow \frac{1}{max(X+1)-X}$

- 예측변수에 결측치가 있는 경우 그 행(개체)는 모든 분석에서 제외된다.
- 단위가 다른 경우에는 적어도 중심화 하거나 표준화 한다.
- 개체의 수는 이론적으로 $p = n - 2$이면 가능하나 변수 수의 5배(변수의 개수가 4개 미만) 빅데이터에서는 변수 개수의 20배 이상 데이터가 있어야 과적합 overfitting 문제가 발생하지 않는다.

기호

- $P(\pi_i)$ = 모집단 i의 사전확률 ($\pi_1 + \pi_2 = 1$)
- $P(X \in R_i \mid \pi_j)$ = 모집단 j에 속한 개체가 판별식에 의해 모집단 i로 판별할 확률 , $i = j$이면 정분류, $i \neq j$이면 오분류
- 비용 $c(j \mid i)$ = 모집단 i에 속한 개체를 모집단 j로 판별하여 발생하는 비용. $i = j$ 이면 비용 0임
- $P(\pi_1)P(X \in R_1 \mid \pi_1)$ = 모집단 1에서 추출된 개체를 모집단 1로 정분류
- $P(\pi_2)P(X \in R_2 \mid \pi_2)$ = 모집단 2에서 추출된 개체를 모집단 2로 정분류
- $P(\pi_1)P(X \in R_2 \mid \pi_1)$ = 모집단 1에서 추출된 개체를 모집단 2로 오분류
- $P(\pi_2)P(X \in R_1 \mid \pi_2)$ = 모집단 2에서 추출된 개체를 모집단 1로 오분류

판별규칙

오분류 기대비용 ECM 최소화

오분류 가대비용 = (오분류확률*비용함수)

$$ECM = c(2|1)P(2|1)p_1 + c(1|2)P(1|2)p_2$$

판별 $R_1 : \frac{f_1(\underline{x})}{f_2(\underline{x})} \geq (\frac{p_2}{p_1})(\frac{c(1|2)}{c(2|1)})$ $R_2 : \frac{f_2(\underline{x})}{f_1(\underline{x})} \geq (\frac{p_1}{p_2})(\frac{c(2|1)}{c(1|2)})$

(만약 비용이 동일하다면 $R_1 : \frac{f_1(\underline{x})}{f_2(\underline{x})} \geq (\frac{p_2}{p_1})$) (모집단 비율도 동일하다면 $R_1 : \frac{f_1(\underline{x})}{f_2(\underline{x})} \geq 1$

우도함수 likelihood ratio (다변량 정규분포)

모집단이 다변량정규분포를 따른다면, 새로운 개체 $\underline{x}_0$에 대하여 $L(\underline{x}_0; \underline{x}_1, \hat{\Sigma}_1) \geq L(\underline{x}_0; \underline{x}_2, \hat{\Sigma}_2)$이 성립하면 모집단 1로 그렇지 않으면 모집단 2로 판별한다.

Fisher 선형 linear 판별

다변량 정규분포의 두 모집단이 동일한 공분산(Σ)을 갖는다면 다음과 같이 통합분산에 의해 추정된다.

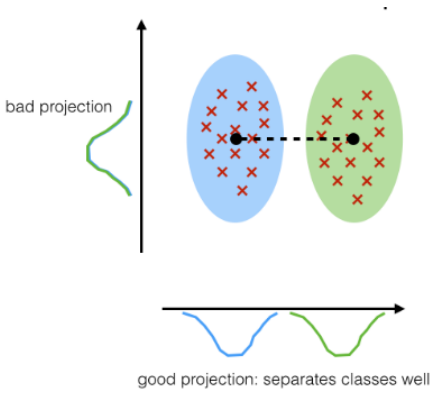
통합 공분산 : $\hat{\Sigma} = \frac{(n_1 - 1)S_1 + (n_2 - 1)S_2}{(n_1 + n_2 - 2)}$

p차 확률변수 벡터 $\underline{x}_p$가 모집단 k에 속할 확률, $P(\underline{X}_p = \underline{x} \,|\, k - th \;\; Pop) = f_k(\underline{x}_p)$ 확률분포함수는

$f_k(\underline{x}_p) = \frac{1}{(2\pi)^{p/2} |\Sigma|^{1/2}} exp(-\frac{1}{2}(\underline{x} - \mu_k)'(\underline{x} - \mu_k))$이다.

Fisher 판별 규칙 : $\delta_k(x) = \log \pi_k - \frac{1}{2}\mu_k^T \Sigma^{-1} \mu_k + x^T \Sigma^{-1} \mu_k$

- 개체 $\underline{x}$의 $\delta_k(\underline{x})$값이 가장 큰 모집단 k로 판별한다.
- 개체는 각 모집단의 중심점(평균)과의 거리가 가까운 집단으로 판별된다.
- x-축 판별변수는 집단 판별 능력이 높으나 y-축 판별변수는 판별능력이 매우 낮다.

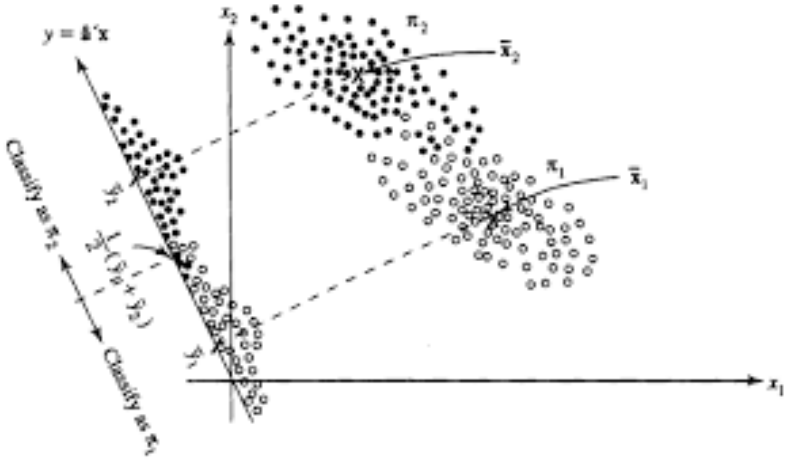


사후 posterior 확률 이용

관측치 $x_0 = (x_{10}, x_{20}, \ldots, x_{p0})'$ (판별변수 p개인 경우) 모집단(π_k, $k = 1,2$)에 속할 사후 확률 계산 :

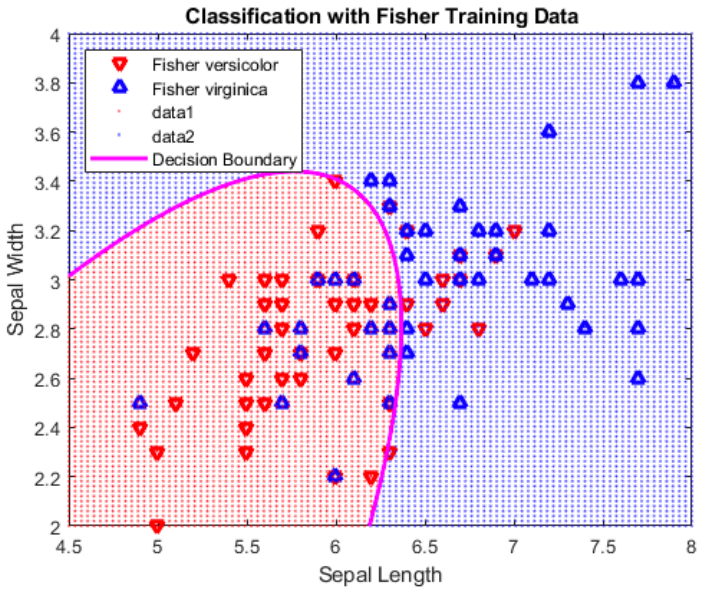
$$\delta_k(x_0) = P(x_0 \hat{\subset} \pi_k) = \underline{x}\hat{\Sigma}^{-1} - \frac{1}{2}\hat{\mu}_k\hat{\Sigma}^{-1} - \hat{\mu}_k + ln(\hat{\pi}_k)$$

$P(\pi_1 \supseteq \underline{x}_0 \,|\, data) \geq P(\pi_2 \supseteq \underline{x}_0 \,|\, data)$ 이면 새로운 개체 $\underline{x}_0$는 모집단 1로 판별한다.



두 모집단의 공분산이 동일하지 않은 경우 Fisher는 Quadratic 판별규칙 제안

사후확률 계산식 $$\delta_k(x) = -\frac{1}{2}\log|\Sigma_k| - \frac{1}{2}(x-\mu_k)^T\Sigma_k^{-1}(x-\mu_k) + \log\pi_k$$



로지스틱 판별회귀 모형

모집단1={성공, π_1}, 모집단2={실패, π_2}, 성공확률 $p = P(y_i \subseteq \pi_1)$

판별변수 = "설명변수" : 모형 $P(Y_i = \left\{ sucess \right\}) = \frac{\exp(X\underline{b})}{1+\exp(X\underline{b})}$

$y_i = 0, 1$, 모집단1={성공}={1}, 모집단2-{실패}={0}, $\pi_i = P(Y_i = ' Success')$)

설명변수(예측변수) 개수 p개인 회귀모형 : $E(Y|\underline{x}) = X\underline{b} + \underline{e}, (\underline{e} \sim N(\underline{0}, \sigma^2 I))$

종속변수 y_i가 측정형이 아니라 (성공, 실패)의 베르누이 분포를 따른다면, $E(Y|\underline{x}) = p$이다.

만약 Y_i= 성공확률이 (π_i-추정)인 i-번째 그룹(총 관측 개수 n_i)에서 관심사건 발생건수라 정의하자.
$Y_i \sim B(n_i, \pi_i)$을 따른다. 그러므로 $E(Y_i) = n_i\pi_i$이다.

회귀계수 추정

연결함수 $g(\mu) = X\underline{b}$ where $\mu =$ 종속변수 평균 (여기서는 $\mu = p$)

- $g(\mu)$는 종속변수가 $(-\infty, \infty)$의 값을 갖도록 변환하는 함수

오즈(odds) : 성공확률을 실패확률로 나눈 값 $odds_i = \frac{\pi_i}{1-\pi_i}$

- 오즈는 $(0, \infty)$를 가지므로 로짓변환하여 (-∞,∞) 값을 가질 수 있으므로 일반 회귀모형 적합 가능

로짓 logit 혹은 log-odds : $logit(\pi_i) = ln(\frac{\pi_i}{1-\pi_i}) = X\underline{b} + \underline{e}, \underline{e} \sim MN(\underline{0}, \Sigma)$

회귀계수 $\hat{b}_k$의 의미

- 설명변수 X_k는 개체 성공 확률에 양/음(회귀계수의 부호와 일치)의 영향을 미친다.
- 설명변수 X_k가 한 단위 증가할 때 오즈비는 $e^{\hat{b}_k}$ 배만큼 **곱**으로 증가한다.

판별규칙

사후확률 $\hat{p} = P(y_i \subseteq \pi_1 \,|\, data)$이 0.5보다 크면 {성공}, 작으면 {실패} 집단으로 분류한다. (일반적 규칙)

최적 판별 기준 사후확률 값은 Youden's J statistic이 이용된다.

Fisher 방법에 비해 장점

- 회귀계수의 부호에 의해 해당 판별변수가 성공 집단에 속할 확률에 미치는 영향 정도를 평가할 수 있음
- 성공 집단에 속할 사후확률(0~1사이 실수)을 추정하여 이를 활용하여 개체의 순위를 결정할 수 있음
- ROC 분석을 통하여 오분류 최소화 하는 "성공/실패" 집단 판별하는 기준값을 설정할 수 있음

단계적 판별분석 방법

판별 분석에서 많은 판별변수(항목)들이 측정되었을 경우 아마 여러분은 의문이 생길 것이다. (1)모든 변수가 판별에 필요한가? (2)어떤 변수가 가장 판별을 잘하는 변수인가? 판별에 적절한 변수만 사용하는 것이 좋으며 판별 능력은 분산 분석의 개념을 이용하여 판단한다.

[선택이유 : Parsimony Rule] 오분류 비율이 동일하거나 차이가 없다면 판별 변수의 수가 적은 판별 방법이 더 효율적이다. 이유는 측정 오류 발생 가능성이 적으며 새로운 개체 판별을 위해 측정해야 하는 변수 수가 적으므로 경제적이다.

Forward 방법

다음 절차에 의해 개체를 판별하는데 가장 유의한 변수 순으로 유의한 변수가 존재하지 않을 때까지 하나씩 넣어 가는 방법이다.

(1) 개체 집단을 설명 변수(요인)로 하고 각 측정 변수를 종속 변수(반응 변수)로 하여 분산 분석(ANOVA)을 실시한다. F-값이 가장 큰 변수를 제일 먼저 선택한다. 이유는 집단의 평균 차이가 가장 크다는 것은 그 변수에 의해 집단 분류가 가장 잘된다는 것이다.

(2) 두 번째 변수 선택은? 첫 번째 선택된 변수를 공변량(covariate)으로 하여 공분산 분석 (ANCOVA) 시행하여 그룹의 F-값이 가장 큰 변수를 선택한다. 공변량은 종속 변수에서 그 변수의 효과를 제외할 때 사용되므로 첫 번째 선택된 변수의 판별 효과를 제외하는 것을 의미한다.

공분산분석: 수학 강의에 대한 새로운 교육방법이 제안되었다. 기존의 교육 방법보다 나은지 알아보기 위하여 각 20명씩 2개의 그룹을 만들어 하나의 그룹에는 새로운 교육 방법, 다른 그룹에는 기존의 교육 방법을 적용해 보자. 그룹 학생들간에는 차이가 있을 것을 예상하여 교육 전 수학 시험을 보았다. 일전 기간 교육 후 수학 능력 시험을 봐 그 성적의 차이가 있는지 분석하였다. 교육 후 점수(Y)가 그룹(새 교육/기존 교육)간 차이가 있는지 알아보려면 분산분석(ANOVA). 그러나 교육 전 이들의 수학 능력이 고려되지 않았다. 모두 수학에 대한 능력이 같지는 않을 것이다. 이 효과를 제외해 주자. 이 역할을 하는 것이 교육 전 수학 점수이고 이를 공변량이라 한다. 이에 적합한 분석이 공변량 분석이다. 여전히 주요 관심은 교육 효과이고 공변량에는 관심이 없다.

(3) 세 번째 변수 선택은? 첫 번째, 두 번째 선택된 변수들을 공변량(covariate)으로 하여 공분산 분석 (ANCOVA) 시행하여 그룹의 F-값이 가장 큰 변수를 선택한다. 이렇게 변수 선택을 반복한다. 만약 F-값이 가장 큰 것이 유의하지 않으면 (p-값이 유의수준보다 크면) 변수 선택을 멈춘다.

Backward 방법

다음 절차에 의해 개체를 판별하는데 유의한 변수를 선택하는 방법으로 일단 모든 변수를 다 고려한 후 유의하지 않은 순서대로 변수를 제거해 나가는 방법

(1) 하나의 변수를 반응 변수, 다른 변수들은 공변량, 그리고 그룹을 요인(설명 변수)으로 하여 공분산 분석을 실시하여 집단의 (Partial SS) F-값이 가장 낮은 변수를 제거한다.

(2) 같은 방법으로 변수를 하나씩 제거해 간다. 집단의 F-값이 모두 유의하면 (p-값이 유의수준보다 작으면) 제거를 멈춘다.

Stepwise 방법

Forward 방법과 매우 유사하다. 일단 선택된 변수들도 다른 변수가 들어간 상태에서 유의성 검정을 하여 새로운 변수보다 덜 유의하면 제거된다. 즉 처음에는 가장 유의하였지만 여러 변수들이 선택된 상황에서는 유의 정도가 떨어질 수 있다. 변수 선택을 위하여 변수를 넣었다 뺐다 하는 반복이 많아 계산이 복잡하고 번거롭지만 컴퓨터 하드웨어, 소프트웨어 발달로 인하여 현재는 가장 많이 사용되는 방법이다.

변수 선택 시 주의점

변수의 수가 15개 이상인 경우 Backward 방법, 15개 미만이 경우는 Stepwise 방법을 사용하는 것을 권한다. 판별분석에서 변수 선택은 다음과 같은 문제점을 지니고 있다. 판별에 유의한 변수를 찾는 경우 F-값만 가지고 선택하므로 그 변수가 얼마나 잘 판별하는지를 고려되지 않는다. 두 변수 (X1, X2) 대해 집단간 산점도가 그림(1)과 같다면 변수 X1이 집단을 가장 잘 분류하므로 먼저 선택되고 X2가 그 다음으로 선택된다. 모두 유의하다. 별 문제가 없어 보인다.

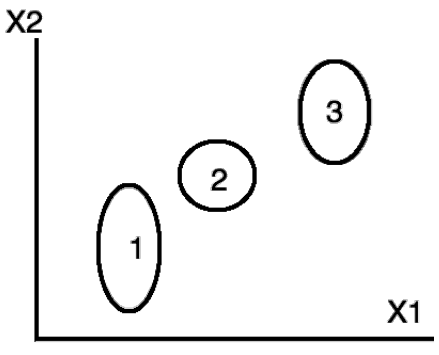


그림 (1)

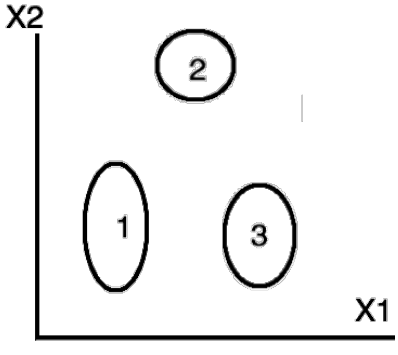


그림 (2)

만약 산점도가 그림(2)와 같다면 실제로는 X1(집단 1,2,3을 모두 분류)가 더 나은 판별을 하지만 변수 선택 방법으로 선택하면 X1(1,3과 2에 집단간 차이가 커서)이 선택될 것이다. 그러므로 변수 선택 방법에 의해 선택된 변수가 판별을 잘한다는 보장은 없다. 그럴지라도 변수 선택 방법은 하나의 좋은 기준을 제시한다.

등분산 검정

Box's (1949) M-test for homogeneity of covariance matrices obtained from multivariate normal data according to one or more classification factors.

모집단 등분산 가정이 만족하면 QDA 방법을 사용한다.

파이썬 함수가 없어 R을 사용하였음

두 집단 등분산 Box M-검정 결과 귀무가설(등분산) 기각되어 QDA 판별분석이 가능함

=> 판별규칙 선택은 Fusion 행렬(정분류 표)를 이용하고 판별 정확도에 의해 최종 결정되므로 굳이 등분산 검정을 이용하여 판별규칙을 선정할 필요는 없다.

```
titanic<-read.csv('http://wolfpack.hnu.ac.kr/Stat_Notes/adv_stat/DATA/titanic_original.csv')
install.packages('heplots')
library(heplots)
boxM(titanic[,c(5,9)],titanic[,2])
```

```
> boxM(titanic[,c(5,9)],titanic[,2])

        Box's M-test for Homogeneity of Covariance Matrices

data:  titanic[, c(5, 9)]
Chi-Sq (approx.) = 255.42, df = 3, p-value < 2.2e-16
```

전처리

Fisher 방법은 개체간의 유클리디안 거리(예측변수 활용)를 이용하여 중심점에 가까운 모집단으로 판별하므로 예측변수 단위를 맞추는 것(표준화)이 적절하다.
X1=탑승객 나이, X2=탑승권 요금

```
import pandas as pd
titanic=pd.read_excel('http://biostat.mc.vanderbilt.edu/wiki/pub/Main/DataSets/titanic3.xls')
titanic.dropna(subset=['survived'],inplace=True)
#titinic['survived']=titinic['survived'].astype(int)
df=titanic[['age','fare','survived']]
df=df.dropna()
df.info()
```

```
from sklearn.preprocessing import scale
X=scale(df[['age','fare']])
y=np.array(df['survived'])
```

Fisher LDA Linear Discriminant Analysis

판별규칙유도

예측변수 벡터 X, 집단변수를 설정하여 LDA 함수를 실행하면 적합 결과가 model_lda에 저장된다. 타이타닉과 같이 모집단 데이터 전체를 사용하거나 모집단 집단 비율과 동일하게 집단으로 층화하여 표본데이터를 사용하는 경우에는 사전 확률을 지정할 필요는 없다. 만약 생존비율이 40%라고 알려져 있다면 다음과 같이 사전 설정하여 최적 판별규칙을 도출할 수 있다.

```
LinearDiscriminantAnalysis(priors=[0.4,0.6])
```

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
import numpy as np
lda=LinearDiscriminantAnalysis()
model_lda=lda.fit(X,y)
```

사전확률 및 판별계수 출력

```
print(model_lda.priors_,model_lda.coef_)
```

모집단 데이터(n=1045, 전처리 후) 생존 비율 40.8%, 사망 59.1%이었으므로 이를 모집단 사전확률로 사용하였다. 앞에서 언급하였듯이 사전 정보가 있어 이를 바꾸고 싶다면

```
LinearDiscriminantAnalysis(priors=[0.4,0.6])
```

도출된 판별규칙 : $-0.222 * (z_1) + 0.585 * (z_2)$, $z_1 = x_1$의 표준화 값, $z_2 = x_2$의 표준화 값,

```
[0.59138756 0.40861244] [[-0.22263017  0.58499963]]
```

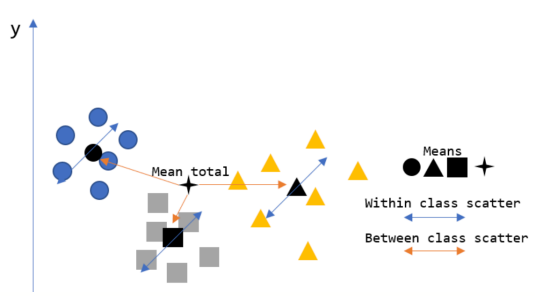
사후확률 출력

```
probs_lda=model_lda.predict_proba(X)
probs_lda
```

개별 개체가 각 모집단(**Y=0: 사망, Y=1: 생존**)에 속할 확률이 저장 : $[P(Y=0), P(Y=1)]$

```
array([[0.19026851, 0.80973149],
       [0.22181564, 0.77818436],
       [0.22472367, 0.77527633],
```

	age	fare	survived
0	29.0000	211.3375	1
1	0.9167	151.5500	1
2	2.0000	151.5500	0



- 첫 승객 나이=29세, 요금 211파운드, 생존 - 판별규칙 적용 결과 사망 사후확률 19%, 생존 사후확률 81%로 생존으로 정분류 판별
- 3번째 승객 2세, 요금 151파운드, 사망, 그러나 판별규칙적용 결과 생존 확률이 78%로 높아 생존으로 오분류 되었음

판별집단

```
pred_lda=model_lda.predict(X)
pred_lda
#print(np.unique(pred_lda, return_counts=True))
```

도출된 판별규칙 : $-0.222 * (z_1) + 0.585 * (z_2)$ 이용하여 각 개체의 집단이 판별 (918명 사망, 127명 생존)-> 앞의 3개 개체(승객)는 1, 1, 1 모두 생존으로 마지막 3명의 승객은 사망으로 (사후) 판별되었다.

```
array([1, 1, 1, ..., 0, 0, 0])
```

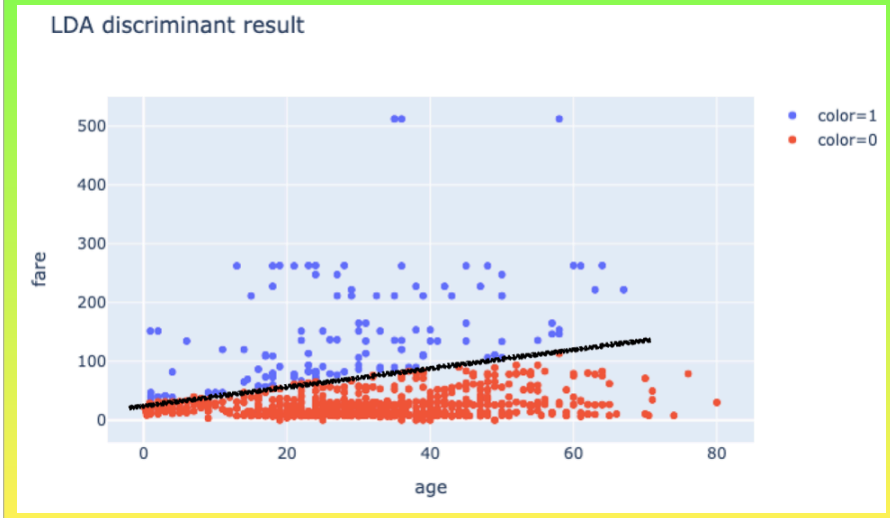
```
(array([0, 1]), array([918, 127]))
```

판별결과 시각화

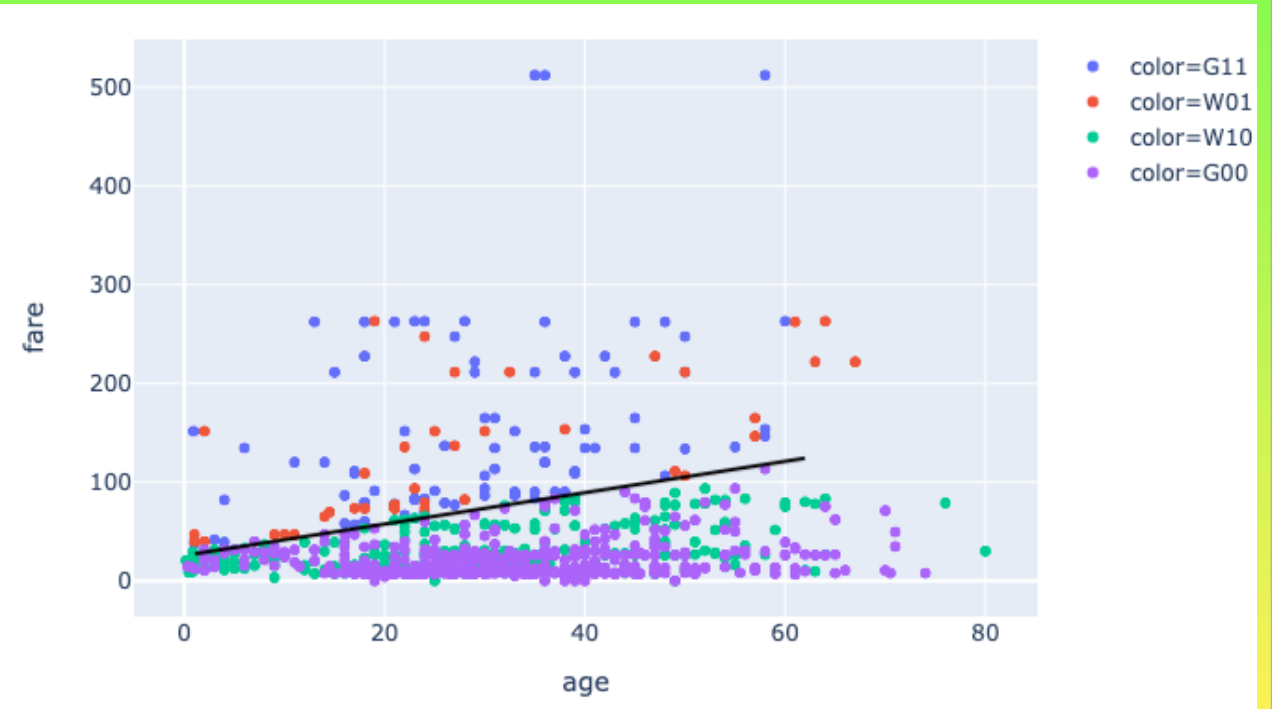
```
lda_y=pd.DataFrame(pred_lda)
lda_y.columns=['y_lda']
lda_y.set_index(df.index,inplace=True)
df_lda=pd.concat([df,lda_y],axis=1)
df_lda.loc[(df_lda['survived']==1) &
(df_lda['y_lda']==1),'result']='G11'
df_lda.loc[(df_lda['survived']==1) &
(df_lda['y_lda']==0) ,'result']='W10'
df_lda.loc[(df_lda['survived']==0) &
(df_lda['y_lda']==1) ,'result']='W01'
df_lda.loc[(df_lda['survived']==0) &
(df_lda['y_lda']==0) ,'result']='G00'
df_lda.head(3)
```

	age	fare	survived	y_lda	result
0	29.0000	211.3375	1	1	G11
1	0.9167	151.5500	1	1	G11
2	2.0000	151.5500	0	1	W01

```
import plotly.express as px
fig = px.scatter(df_lda,x="age",y="fare",
color=df_lda["y_lda"].astype(str),title="LDA discriminant rule")
fig.show()
```



```
import plotly.express as px
fig = px.scatter(df_lda,x="age",y="fare",
color=df_lda["result"].astype(str),title="LDA discriminant result")
fig.show()
```



판별 교차표 및 정분류 비율

판별 정확율 63.8%, 사망 판별 정확도 63.1%(사망자로 판별된 928명(579+339) 중 사망으로 정분류된 579명 비율), 생존 판별 정확도 69.3% (생존 판별 127명 중 생존으로 판별된 88명 비율)

$$precision = \frac{tp}{tp+fp}, recall = \frac{tp}{tp+fn}, f1score = 2\frac{precison \times recall}{precison + recall}$$

```
from sklearn.metrics import confusion_matrix,
classification_report, precision_score
print(confusion_matrix(pred_lda,y))
print(classification_report(y, pred_lda, digits=3))
```

```
pd.crosstab(df_lda['survived'],df_lda['y_lda']) #동일결과
```

```
[[579 339]
 [ 39  88]]
              precision    recall  f1-score   support

           0      0.631     0.937     0.754       618
           1      0.693     0.206     0.318       427

    accuracy                         0.638      1045
   macro avg      0.662     0.571     0.536      1045
weighted avg      0.656     0.638     0.576      1045
```

y_lda	0	1
survived		
0	579	39
1	339	88

ROC & AUC

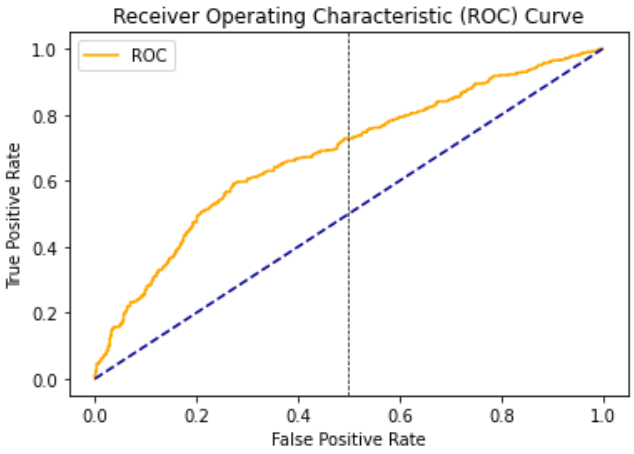
```
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
auc = roc_auc_score(y,probs_lda[:,1])
fpr,tpr,thresholds=roc_curve(y,probs_lda[:,1])
print('AUC for LDA: %.2f' % auc)
```

```
AUC for LDA: 0.68
```

0.68 Satisfactory 만족하는 수준

```
def plot_roc_curve(fpr, tpr,p):
  import matplotlib.pyplot as plt
  plt.plot(fpr, tpr, color='orange', label='ROC')
  plt.plot([0, 1], [0, 1], color='darkblue', linestyle='--')
  plt.xlabel('False Positive Rate')
  plt.ylabel('True Positive Rate')
  plt.title('Receiver Operating Characteristic (ROC) Curve')
  plt.axvline(x=p,linestyle='--',linewidth=0.7,color='k')
  plt.legend()
  plt.show()
```

```
plot_roc_curve(fpr, tpr,0.5)
```



Optimal threshold : 사후확률=0.369 이상 생존(Y=1)으로 판단하면 정확률을 높일 수 있음, **활용도 낮음**

```
def Find_Optimal_Cutoff(target, predicted):
  fpr, tpr, threshold = roc_curve(target, predicted)
  i = np.arange(len(tpr))
  roc = pd.DataFrame({'tf' : pd.Series(tpr-(1-fpr), index=i),
'threshold' : pd.Series(threshold, index=i)})
  roc_t = roc.iloc[(roc.tf-0).abs().argsort()[:1]]
  return list(roc_t['threshold'])
```

```
Find_Optimal_Cutoff(y,probs_lda[:,1])
```

```
[0.3691232279260048]
```

Fisher QDA Quadratic Discriminant Analysis

판별규칙유도

예측변수 벡터 X, 집단변수를 설정하여 LDA 함수를 실행하면 적합 결과가 model_qda에 저장된다. 타이타닉과 같이 모집단 데이터 전체를 사용하거나 모집단 집단 비율과 동일하게 집단으로 층화하여 표본데이터를 사용하는 경우에는 사전 확률을 지정할 필요는 없다. 만약 생존비율이 40%라고 알려져 있다면 다음과 같이 사전 설정하여 최적 판별규칙을 도출할 수 있다.

```
QuadraticDiscriminantAnalysis(priors=[0.4,0.6])
```

```
from sklearn.discriminant_analysis import
QuadraticDiscriminantAnalysis
qda=QuadraticDiscriminantAnalysis()
model_qda=qda.fit(X,y)
```

사전확률 및 판별계수 출력

```
print(model_qda.priors_)
```

LDA 판별분석과는 달리 선형계수-판별식은 제공되지 않는다.

```
[0.59138756 0.40861244]
```

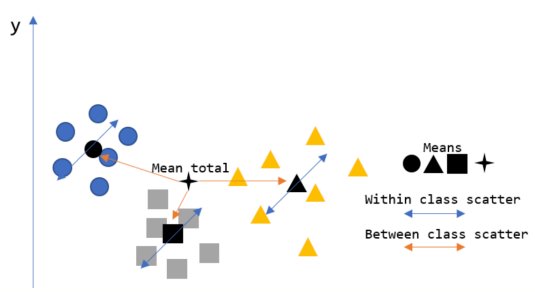
사후확률 출력

```
probs_qda=model_qda.predict_proba(X)
probs_qda[0:3]
```

개별 개체가 각 모집단(**Y=0: 사망, Y=1: 생존**)에 속할 확률이 저장 : $[P(Y=0), P(Y=1)]$

```
array([[2.92682333e-05, 9.99970732e-01],
       [2.89782837e-03, 9.97102172e-01],
       [3.09723129e-03, 9.96902769e-01]])
```

	age	fare	survived
0	29.0000	211.3375	1
1	0.9167	151.5500	1
2	2.0000	151.5500	0



- 첫 승객 나이=29세, 요금 211파운드, 생존 - 판별규칙 적용 결과 사망 사후확률 1%, 생존 사후확률 99%로 생존으로 정분류 판별
- 3번째 승객 2세, 요금 151파운드, 사망, 그러나 판별규칙적용 결과 생존 확률이 99%로 높아 생존으로 오분류 되었음

도출된 이차 판별규칙에 의해 각 개체의 집단이 판별 -> 933 승객은 사망, 112명은 생존으로 판별
앞의 3개 개체(승객)는 1, 1, 1 모두 생존으로 마지막 3명의 승객은 사망으로 (사후) 판별되었다.

```
array([1, 1, 1, ..., 0, 0, 0])
```

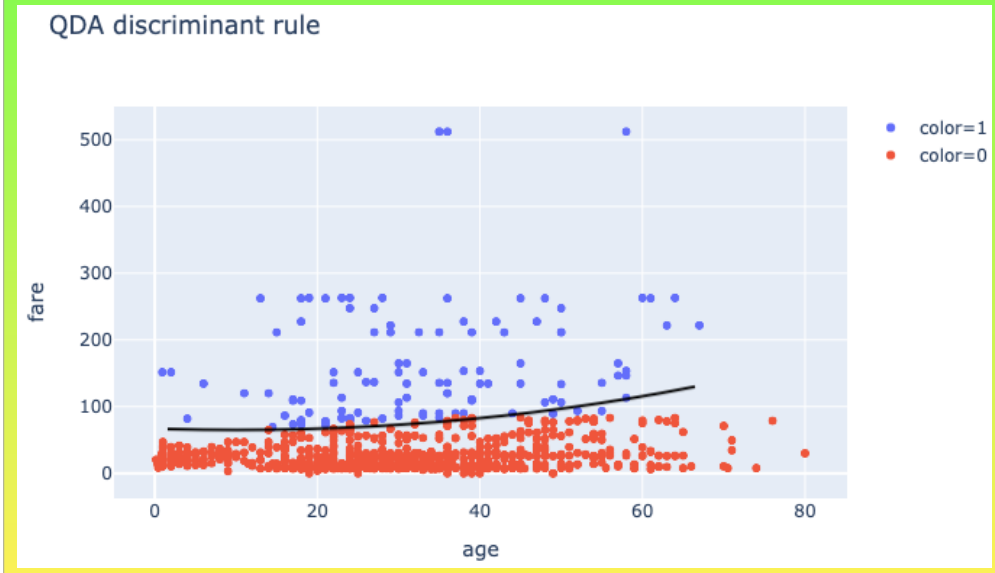
```
(array([0, 1]), array([933, 112]))
```

판별결과 시각화

```
qda_y=pd.DataFrame(pred_qda)
qda_y.columns=['y_qda']
qda_y.set_index(df.index,inplace=True)
df_qda=pd.concat([df,qda_y],axis=1)
df_qda.loc[(df_qda['survived']==1) &
(df_qda['y_qda']==1),'result']='G11'
df_qda.loc[(df_qda['survived']==1) &
(df_qda['y_qda']==0) ,'result']='W10'
df_qda.loc[(df_qda['survived']==0) &
(df_qda['y_qda']==1) ,'result']='W01'
df_qda.loc[(df_qda['survived']==0) &
(df_qda['y_qda']==0) ,'result']='G00'
df_qda.head(3)
```

	age	fare	survived	y_qda	result
0	29.0000	211.3375	1	1	G11
1	0.9167	151.5500	1	1	G11
2	2.0000	151.5500	0	1	W01

```
import plotly.express as px
fig = px.scatter(df_qda,x="age",y="fare",
color=df_lda["y_qda"].astype(str),title="QDA discriminant rule")
fig.show()
```



판별식 -> 이차식

```
import plotly.express as px
fig = px.scatter(df_qda,x="age",y="fare",
color=df_qda["result"].astype(str),title="QDA discriminant result")
fig.show()
```

판별 교차표 및 정분류 비율

판별 정확율 63.3%(LDA 63.8%), 사망 판별 정확도 62.6%(LDA-63.1%, 사망자로 판별된 933명(584+349) 중 사망으로 정분류된 584명 비율), 생존 판별 정확도 69.6% (LDA-69.3%, 생존 판별 112명 중 생존으로 판별된 78명 비율)

$$precision = \frac{tp}{tp+fp}, recall = \frac{tp}{tp+fn}, f1score = 2\frac{precison \times recall}{precison + recall}$$

```
from sklearn.metrics import confusion_matrix,
classification_report, precision_score
print(confusion_matrix(pred_qda,y))
print(classification_report(y, pred_qda, digits=3))
```

```
pd.crosstab(df_qda['survived'],df_qda['q_lda']) #동일결과
```

```
[[584 349]
 [ 34  78]]
              precision    recall  f1-score   support

           0      0.626     0.945     0.753       618
           1      0.696     0.183     0.289       427

    accuracy                          0.633      1045
   macro avg      0.661     0.564     0.521      1045
weighted avg      0.655     0.633     0.564      1045
```

ROC & AUC

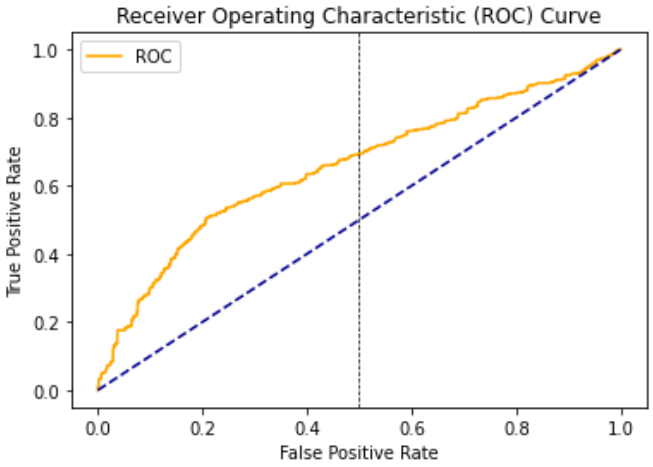
```
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
auc = roc_auc_score(y,probs_qda[:,1])
fpr,tpr,thresholds=roc_curve(y,probs_qda[:,1])
print('AUC for LDA: %.2f' % auc)
```

```
AUC for LDA: 0.66
```

0.66 Satisfactory 만족하는 수준 (선형판별식 68%보다 낮음)

```
def plot_roc_curve(fpr, tpr,p):
  import matplotlib.pyplot as plt
  plt.plot(fpr, tpr, color='orange', label='ROC')
  plt.plot([0, 1], [0, 1], color='darkblue', linestyle='--')
  plt.xlabel('False Positive Rate')
  plt.ylabel('True Positive Rate')
  plt.title('Receiver Operating Characteristic (ROC) Curve')
  plt.axvline(x=p,linestyle='--',linewidth=0.7,color='k')
  plt.legend()
  plt.show()
```

```
plot_roc_curve(fpr, tpr,0.5)
```



Optimal threshold : 사후확률=0.2429 이상 생존(Y=1)으로 판단하면 정확률을 높일 수 있음, **활용도 낮음**

```
def Find_Optimal_Cutoff(target, predicted):
  fpr, tpr, threshold = roc_curve(target, predicted)
  i = np.arange(len(tpr))
  roc = pd.DataFrame({'tf' : pd.Series(tpr-(1-fpr), index=i),
'threshold' : pd.Series(threshold, index=i)})
  roc_t = roc.iloc[(roc.tf-0).abs().argsort()[:1]]
  return list(roc_t['threshold'])
Find_Optimal_Cutoff(y,probs_qda[:,1])
```

```
[0.24292416296069144]
```

Logistic Regression Discriminant Analysis

모형 적합

```
import pandas as pd
titanic=pd.read_excel('http://biostat.mc.vanderbilt.edu/wiki/pub/Main/DataSets/titanic3.xls')
titanic.dropna(subset=['survived'],inplace=True)
#titinic['survived']=titinic['survived'].astype(int)
df=titanic[['age','fare','survived']]
df=df.dropna()
df.info()
```

```
from sklearn.linear_model import LogisticRegression
X=df[['age','fare']]
y=df['survived']
logreg = LogisticRegression()
logreg.fit(X,y)
```

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

사후확률, 판별집단

```
logit_pred=np.array(logreg.predict(X))
logit_y_1=np.array(logreg.predict_proba(X)[:,1])
result=np.stack((logit_pred,logit_y_1),axis=1)
logit_pred=pd.DataFrame(result,index=df.index,columns=["logit_y","logit_y1"])
logit_pred.head(3)
```

```
df_logit=pd.concat([df,logit_pred],axis=1)
df_logit.head(3)
```

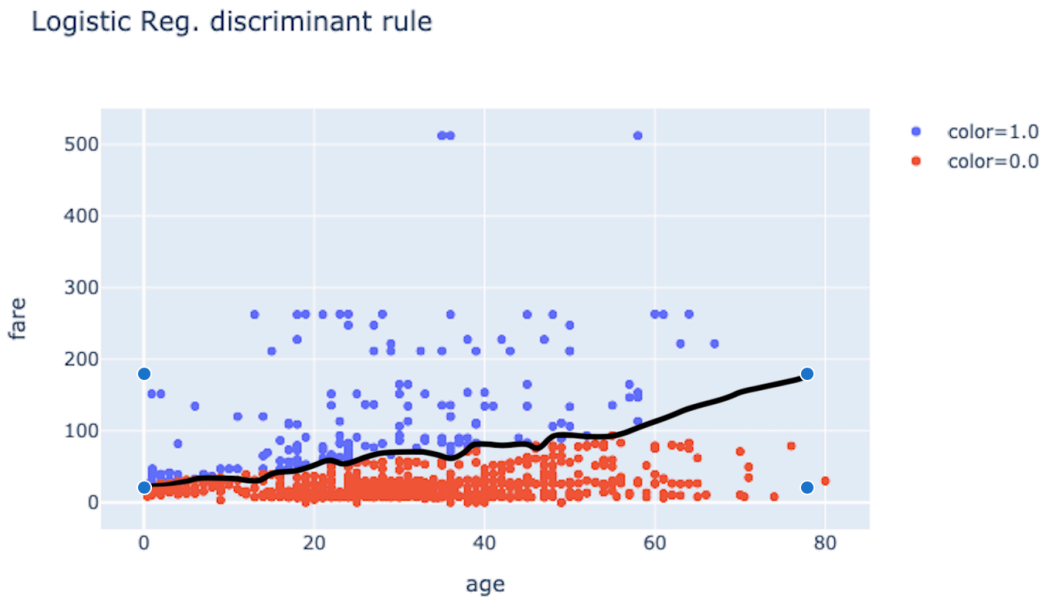
	logit_y	logit_y1
0	1.0	0.882296
1	1.0	0.843046
2	1.0	0.840630

	age	fare	survived	logit_y	logit_y1
0	29.0000	211.3375	1	1.0	0.882296
1	0.9167	151.5500	1	1.0	0.843046
2	2.0000	151.5500	0	1.0	0.840630

판별결과 시각화

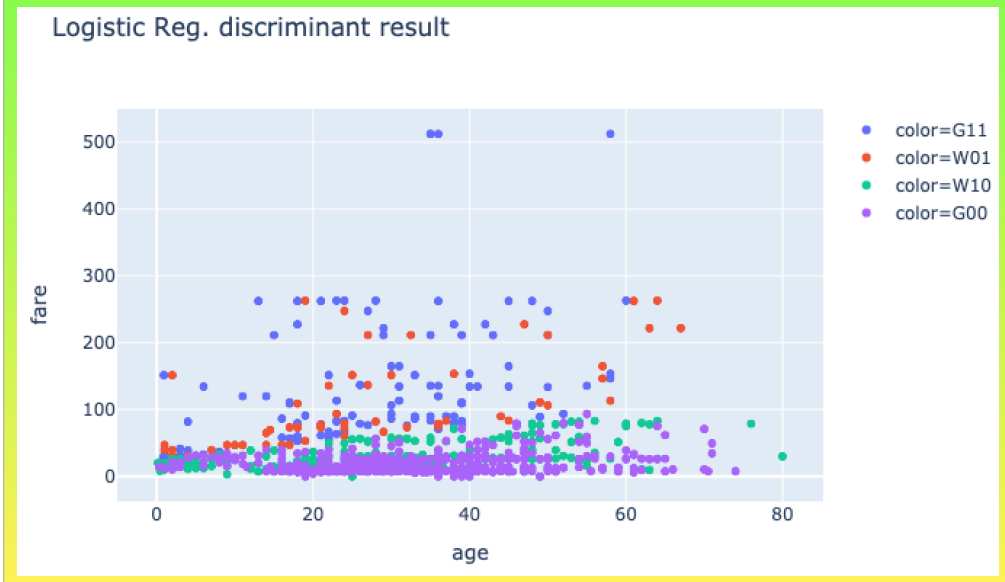
```
df_logit.loc[(df_logit['survived']==1) & (df_logit['logit_y']==1),'result']='G11'
df_logit.loc[(df_logit['survived']==1) & (df_logit['logit_y']==0),'result']='W10'
df_logit.loc[(df_logit['survived']==0) & (df_logit['logit_y']==1),'result']='W01'
df_logit.loc[(df_logit['survived']==0) & (df_logit['logit_y']==0),'result']='G00'
```

```
import plotly.express as px
fig = px.scatter(df_logit,x="age",y="fare",
color=df_logit["logit_y"].astype(str),title="Logistic Reg. discriminant rule")
fig.show()
```



[오분류, 정분류 표현]

```
import plotly.express as px
fig = px.scatter(df_logit,x="age",y="fare",color=
df_logit["result"].astype(str),title="Logistic Reg. discriminant result")
fig.show()
```



Confusion matrix 오분류

판별 정확율 **64%**(LDA 63.8% QDA-63.3%), 사망 판별 정확도 **63.7%** (LDA-63.1%, QDA-62.6% 사망자로 판별된 880명(561+319) 중 사망으로 정분류된 584명 비율), 생존 판별 정확도 **65.5%** (LDA-69.3%, QDA-69.6% 생존 판별 165명 중 생존으로 판별된 108명 비율)

$$precision = \frac{tp}{tp+fp}, recall = \frac{tp}{tp+fn}, f1score = 2\frac{precison \times recall}{precison + recall}$$

```
pd.crosstab(df_logit['survived'],df_logit['logit_y'])
```

```
from sklearn import metrics
cnf_matrix = metrics.confusion_matrix(y, y_pred)
cnf_matrix
```

logit_y	0.0	1.0
survived		
0	561	57
1	319	108

```
array([[561,  57],
       [319, 108]])
```

```
print("Accuracy:",metrics.accuracy_score(y,logit_pred))
print("Precision:",metrics.precision_score(y,logit_pred))
print("Recall:",metrics.recall_score(y,logit_pred))
```

```
Accuracy: 0.6401913875598086
Precision: 0.6545454545454545
Recall: 0.2529274004683841
```

```
from sklearn.metrics import confusion_matrix,
classification_report, precision_score
print(confusion_matrix(logit_pred,y))
print(classification_report(y,logit_pred, digits=3))
```

```
[[561 319]
 [ 57 108]]
              precision    recall  f1-score   support

           0      0.637     0.908     0.749       618
           1      0.655     0.253     0.365       427

    accuracy                          0.640      1045
   macro avg      0.646     0.580     0.557      1045
weighted avg      0.644     0.640     0.592      1045
```

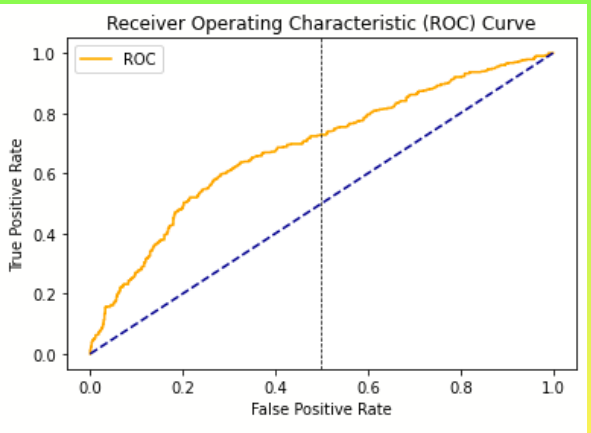
ROC & AUC

```
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
auc = roc_auc_score(y,logreg.predict_proba(X)[:,1])
fpr,tpr,thresholds=roc_curve(y,logreg.predict_proba(X)[:,1])
print('AUC for Logit: %.2f' % auc)
```

```
AUC for Logit: 0.68
```

0.68 Satisfactory 만족하는 수준

```
def plot_roc_curve(fpr, tpr,p):
  import matplotlib.pyplot as plt
  plt.plot(fpr, tpr, color='orange', label='ROC')
  plt.plot([0, 1], [0, 1], color='darkblue', linestyle='--')
  plt.xlabel('False Positive Rate')
  plt.ylabel('True Positive Rate')
  plt.title('Receiver Operating Characteristic (ROC) Curve')
  plt.axvline(x=p,linestyle='--',linewidth=0.7,color='k')
  plt.legend()
  plt.show()
plot_roc_curve(fpr, tpr,0.5)
```



Optimal threshold : 사후확률=0.3671 이상 생존(Y=1)으로 판단하면 정확률을 높일 수 있음, **활용도 낮음**

```
def Find_Optimal_Cutoff(target, predicted):
  fpr, tpr, threshold = roc_curve(target, predicted)
  i = np.arange(len(tpr))
  roc = pd.DataFrame({'tf' : pd.Series(tpr-(1-fpr), index=i),
'threshold' : pd.Series(threshold, index=i)})
  roc_t = roc.iloc[(roc.tf-0).abs().argsort()[:1]]
  return list(roc_t['threshold'])
Find_Optimal_Cutoff(y,probs_qda[:,1])
```

```
[0.3671353826305273]
```

statsmodels 모듈 활용하여 추정방정식 출력

추정 모형 : $logit(\pi_i) = ln(\frac{\pi_i}{1-\pi_i}) = -0.341 - 0.0168Age + 0.0134Fare$

- 나이 변수, 요금 변수 모두 유의확률이 <0.001 로 매우 낮아 유의함
- 나이 회귀계수 부호가 음이므로 나이가 많아질수록 생존 확률 낮아짐 => $E^{-0.0168} = 0.9833$, $1/0.9833 = 1.017$ 나이가 1살 낮아지면 생존 오즈(사망대비 생존확률)가 1.017배 곱으로 상승
- 요금 회귀계수 양이므로 요금 많이 지불할수록 생존확률 높아짐 => $E^{0.0134} = 1.0135$, 요금을 1파운드 더 지불하였으면 생존 오즈 1.0135배 곱 상승

```
import pandas as pd
titanic=pd.read_excel('http://biostat.mc.vanderbilt.edu/wiki/pub/Main/DataSets/titanic3.xls')
titanic.dropna(subset=['survived'],inplace=True)
#titinic['survived']=titinic['survived'].astype(int)
df=titanic[['age','fare','survived']]
df=df.dropna()

import statsmodels.api as sm
X=df[['age','fare']]
y=df['survived']
X=sm.add_constant(X)
model_log=sm.Logit(y,X)
result=model_log.fit(method='newton')
result.summary()
```

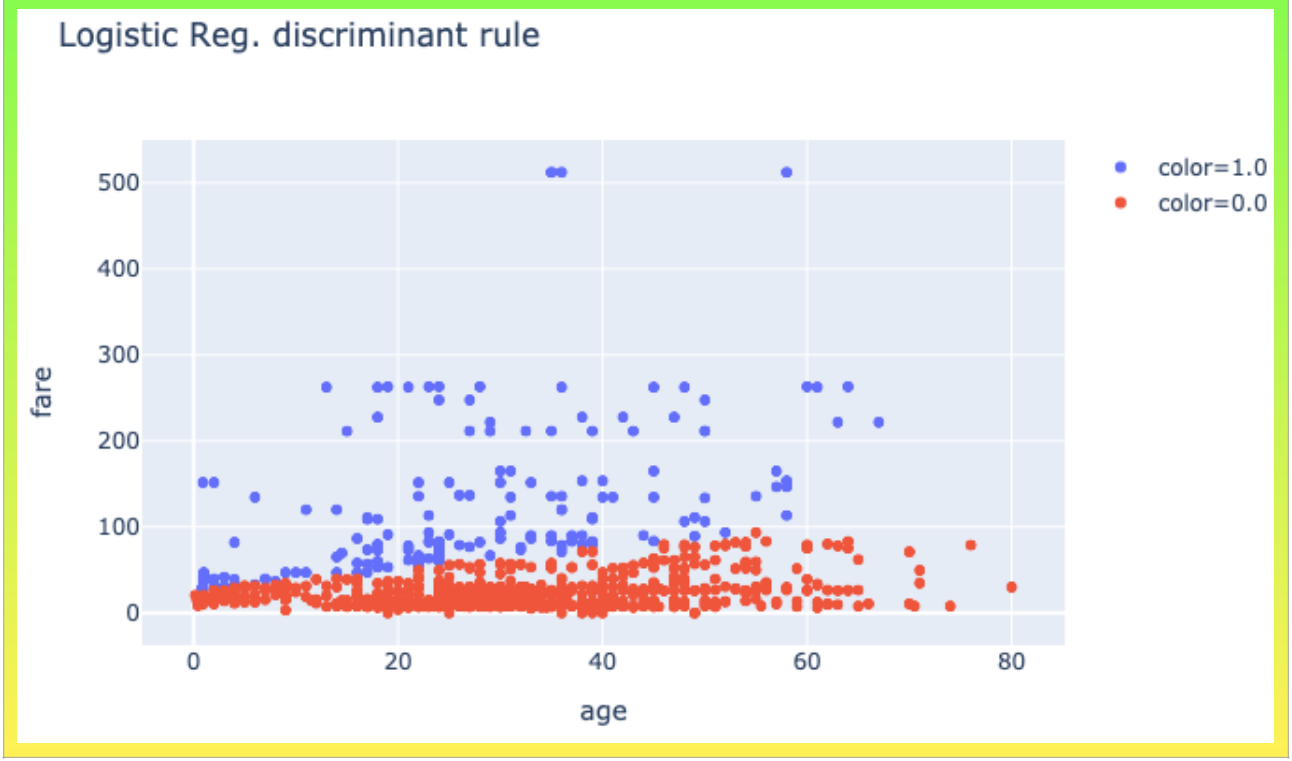
```
Optimization terminated successfully.
         Current function value: 0.634973
         Iterations 6
```

Logit Regression Results

Dep. Variable:	survived	**No. Observations:**	1045
Model:	Logit	**Df Residuals:**	1042
Method:	MLE	**Df Model:**	2
Date:	Sun, 11 Oct 2020	**Pseudo R-squ.:**	0.06118
Time:	01:22:02	**Log-Likelihood:**	-663.55
converged:	True	**LL-Null:**	-706.79
Covariance Type:	nonrobust	**LLR p-value:**	1.667e-19

	coef	std err	z	P>|z|	[0.025	0.975]
const	-0.3410	0.152	-2.250	0.024	-0.638	-0.044
age	-0.0168	0.005	-3.537	0.000	-0.026	-0.007
fare	0.0134	0.002	7.487	0.000	0.010	0.017

```
result.predict(X)
df['logit_sm_y1']=pd.DataFrame(np.array(result.predict(X)),index=
df.index,columns=["sm_logit_y1"])
df.loc[df['logit_sm_y1']>0.5,'sm_result']=1
df.loc[df['logit_sm_y1']<=0.5,'sm_result']=0

import plotly.express as px
fig = px.scatter(df,x="age",y="fare",color=
df["sm_result"].astype(str),title="Logistic Reg. discriminant rule")
fig.show()
```



추정 결과는 당연히 sklearn.linear_model import LogisticRegression와 동일하다.

```
from sklearn.metrics import confusion_matrix, classification_report,
precision_score
print(confusion_matrix(np.array(df['sm_result']),y))
print(classification_report(y,np.array(df['sm_result']), digits=3))
```

```
[[561 319]
 [ 57 108]]
              precision    recall  f1-score   support

           0      0.637     0.908     0.749       618
           1      0.655     0.253     0.365       427

    accuracy                          0.640      1045
   macro avg      0.646     0.580     0.557      1045
weighted avg      0.644     0.640     0.592      1045
```

측정형, 범주형 변수 모두 활용하기

예제 데이터 불러오기

```
import pandas as pd
titanic=pd.read_excel('http://biostat.mc.vanderbilt.edu/wiki/pub/Main/DataSets/titanic3.xls')
titanic.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1309 entries, 0 to 1308
Data columns (total 14 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   pclass      1309 non-null   int64
 1   survived    1309 non-null   int64
 2   name        1309 non-null   object
 3   sex         1309 non-null   object
 4   age         1046 non-null   float64
 5   sibsp       1309 non-null   int64
 6   parch       1309 non-null   int64
 7   ticket      1309 non-null   object
 8   fare        1308 non-null   float64
 9   cabin       295 non-null    object
 10  embarked    1307 non-null   object
 11  boat        486 non-null    object
 12  body        121 non-null    float64
 13  home.dest   745 non-null    object
```

결측값 여부 확인 및 분석 대상 데이터 : 결측값은 모형에 활용 불가능하다.

```
titanic.isnull().sum()
```

```
pclass          0
survived        0
name            0
sex             0
age           263
sibsp           0
parch           0
ticket          0
fare            1
cabin        1014
embarked        2
boat          823
body         1188
home.dest     564
```

```
df=titanic.iloc[:,[0,1,3,4,5,6,8,10]]
df=df.dropna()
df.info()
```

```
Int64Index: 1043 entries, 0 to 1308
Data columns (total 8 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   pclass    1043 non-null   int64
 1   survived  1043 non-null   int64
 2   sex       1043 non-null   object
 3   age       1043 non-null   float64
 4   sibsp     1043 non-null   int64
 5   parch     1043 non-null   int64
 6   fare      1043 non-null   float64
 7   embarked  1043 non-null   object
```

범주형 변수 더미변수 만들기

범주형 범주가 k개이면 $(k-1)$개 더미변수가 필요하다.

더미변수가 가지는 값은 0, 1이다. 회귀모형에서 더미변수는 지시변수라고 한다.
성별-male, famale 2개만 있는 경우 한 개의 더미변수만으로 구별 가능하다. 하여 sex_male은 필요없다.
범주가 3개인 출항정보는 2개의 더미변수만 필요하므로 마지막 embarked_S는 제외
성별 회귀계수의 의미는 여자인 경우이며, embarked는 S기준 다른 항구 차이가 된다.

```
indicate_lst=df[['sex','embarked']]
indicate=pd.get_dummies(indicate_var)
indicate.head(3)
```

	sex_female	sex_male	embarked_C	embarked_Q	embarked_S
0	1	0	0	0	1
1	0	1	0	0	1

로지스틱 회귀모형 추정

```
df=pd.concat([df,indicate],axis
=1)
import statsmodels.api as sm
X=df.iloc[:,
[0,3,4,5,6,8,10,11]]
y=df['survived']
X=sm.add_constant(X)
model_log=sm.Logit(y,X)
result=model_log.fit()
result.summary()
```

Logit Regression Results

Dep. Variable:	survived	No. Observations:	1043
Model:	Logit	Df Residuals:	1034
Method:	MLE	Df Model:	8
Date:	Sun, 11 Oct 2020	Pseudo R-squ.:	0.322
Time:	23:57:24	Log-Likelihood:	-477.
converged:	True	LL-Null:	-704.
Covariance Type:	nonrobust	LLR p-value:	2.794

	coef	std err	z	P>|z|	[0.025	0.975]
const	1.9259	0.446	4.321	0.000	1.052	2.799
pclass	-1.0093	0.133	-7.563	0.000	-1.271	-0.748
age	-0.0377	0.007	-5.681	0.000	-0.051	-0.025
sibsp	-0.3480	0.108	-3.209	0.001	-0.561	-0.135
parch	0.0498	0.104	0.478	0.632	-0.154	0.254
fare	0.0005	0.002	0.239	0.811	-0.003	0.004
sex_female	2.6089	0.179	14.550	0.000	2.258	2.960
embarked_C	0.6791	0.212	3.210	0.001	0.264	1.094
embarked_Q	-0.7674	0.406	-1.889	0.059	-1.564	0.029

- 유의한 설명변수 : 객실등급, 나이, 동반 가족수, 성별, 출항지
- 생존 가능성 높은 변수 : 나이 적을수록(회귀계수 부호 음), 객실등급 좋을수록(pclass 낮은 값이 좋은 객실)
- 여자(sex_famale, 회귀계수 양)이므로 남자에 비래 생존 가능성 높음
- 출항지 C 포트는 양의 회귀부호를 가지므로 기준이 되는 S 항구보다 생존 가능성 높으나 Q의 경우는 S 항구에 비해 생존 가능성 낮음
- 나이와 요금 두 변수만 활용할 때는 요금이 유의하였으나 pclass(객실등급)가 요금을 대체할 수 있고 객실등급 변수가 요금보다 더 생존 여부 판별 능력이 높기 때문이다.

사후확률 및 판별집단 만들기 및 Confusion

```
y_pred=1*(result.predict(X)>0.5)
y_pred
```

```
from sklearn.metrics import confusion_matrix,
classification_report, precision_score
print(confusion_matrix(y_pred,y))
print(classification_report(y,y_pred))
```

판별 정확율 **79%**, 사망 판별 정확도 **81%** (사망자로 판별된 654명(527+127) 중 사망으로 정분류된 527명 비율), 생존 판별 정확도 **77%** (생존 판별 489명 중 생존으로 판별된 298명 비율)

$$precision = \frac{tp}{tp+fp}, recall = \frac{tp}{tp+fn}, f1score = 2\frac{precison \times recall}{precison + recall}$$

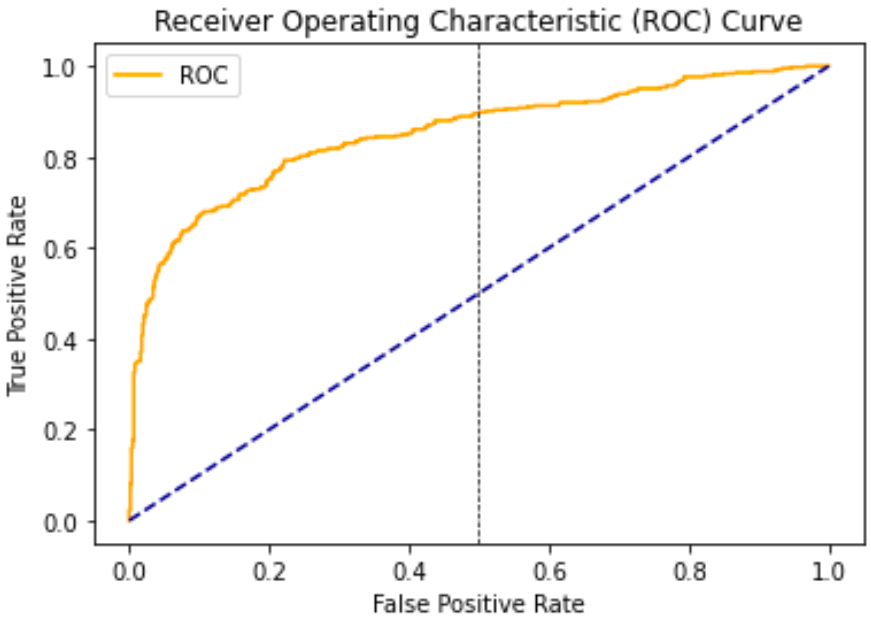
```
[[527 127]
 [ 91 298]]
              precision    recall  f1-score   support

           0       0.81      0.85      0.83       618
           1       0.77      0.70      0.73       425

    accuracy                           0.79      1043
   macro avg       0.79      0.78      0.78      1043
weighted avg       0.79      0.79      0.79      1043
```

ROC & AUC

```
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
auc = roc_auc_score(y,result.predict(X))
fpr,tpr,thresholds=roc_curve(y,result.predict(X))
print('AUC for LDA: %.2f' % auc)
```

```
def plot_roc_curve(fpr, tpr,p):
  import matplotlib.pyplot as plt
  plt.plot(fpr, tpr, color='orange', label='ROC')
  plt.plot([0, 1], [0, 1], color='darkblue', linestyle='--')
  plt.xlabel('False Positive Rate')
  plt.ylabel('True Positive Rate')
  plt.title('Receiver Operating Characteristic (ROC) Curve')
  plt.axvline(x=p,linestyle='--',linewidth=0.7,color='k')
  plt.legend()
  plt.show()
plot_roc_curve(fpr, tpr,0.5)
```

```
AUC for LDA: 0.85
```
매우 좋음 very good 정확도 매우 높음



Optimal 기준 사후확률 결정

```
def Find_Optimal_Cutoff(target, predicted):
  fpr, tpr, threshold = roc_curve(target, predicted)
  i = np.arange(len(tpr))
  roc = pd.DataFrame({'tf' : pd.Series(tpr-(1-fpr), index=i),
'threshold' : pd.Series(threshold, index=i)})
  roc_t = roc.iloc[(roc.tf-0).abs().argsort()[:1]]
  return list(roc_t['threshold'])
Find_Optimal_Cutoff(y,result.predict(X))
```

```
[0.37389987544724196]
```

유의한 변수만 활용하여 판별하기

```
df=pd.concat([df,indicate],axis=1)
import statsmodels.api as sm
X=df.iloc[:,[0,3,4,8,10,11]]
y=df['survived']
X=sm.add_constant(X)
model_log=sm.Logit(y,X)
result=model_log.fit()
result.summary()
```

Logit Regression Results

Dep. Variable:	survived	**No. Observations:**	1043
Model:	Logit	**Df Residuals:**	1036
Method:	MLE	**Df Model:**	6
Date:	Mon, 12 Oct 2020	**Pseudo R-squ.:**	0.3226
Time:	00:13:10	**Log-Likelihood:**	-477.56
converged:	True	**LL-Null:**	-704.99
Covariance Type:	nonrobust	**LLR p-value:**	4.389e-95

	coef	std err	z	P>|z|	[0.025	0.975]
const	1.9781	0.408	4.852	0.000	1.179	2.777
pclass	-1.0232	0.117	-8.746	0.000	-1.252	-0.794
age	-0.0378	0.007	-5.708	0.000	-0.051	-0.025
sibsp	-0.3270	0.102	-3.192	0.001	-0.528	-0.126
sex_female	2.6295	0.176	14.922	0.000	2.284	2.975
embarked_C	0.6919	0.207	3.344	0.001	0.286	1.097
embarked_Q	-0.7802	0.405	-1.924	0.054	-1.575	0.014

판별 정확율 **79%(모든 변수 사용과 동일)**, 사망 판별 정확도 **85%** (81%에서 높아짐), 생존 판별 정확도 **70%** (77%에서 낮아짐) - 생존 판별은 낮으나 사망 판별은 높음 : 비용측면에서 최종 모형 결정하면 된다. 단 통계학의 Occam's razor (오컴 면도날, 적은 가정으로 동일한 결론에 도달하는 설명이 더 좋음, 통계학의 유의성 검정의 근간이 된다) 판단에서는 유의한 변수로 하는 것이 적절함. 향후 판별에 사용할 때 fare(지불요금), parch 동반 친지수) 변수에 대한 정보는 더 이상 필요 없기 때문에 더 유용하다.

```
[[527 128]
 [ 91 297]]
              precision    recall  f1-score   support

           0       0.80      0.85      0.83       618
           1       0.77      0.70      0.73       425

    accuracy                           0.79      1043
   macro avg       0.79      0.78      0.78      1043
weighted avg       0.79      0.79      0.79      1043
```

머신러닝 기법 활용

데이터

```
import pandas as pd
titanic=pd.read_excel('http://biostat.mc.vanderbilt.edu/wiki/pub/Main/DataSets/titanic3.xls')
df=titanic.iloc[:,[0,1,3,4,5,6,8,10]]
df=df.dropna()
df.info()
```

```
indicate=pd.get_dummies(df[['sex','embarked']])
df0=pd.concat([df,indicate],axis=1)
df0.info()
```

```
Int64Index: 1043 entries, 0 to 1308
Data columns (total 10 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   pclass      1043 non-null   int64
 1   survived    1043 non-null   int64
 2   sex         1043 non-null   object
 3   age         1043 non-null   float64
 4   sibsp       1043 non-null   int64
 5   parch       1043 non-null   int64
 6   fare        1043 non-null   float64
 7   embarked    1043 non-null   object
 8   sex_female  1043 non-null   uint8
 9   sex_male    1043 non-null   uint8
 10  embarked_C  1043 non-null   uint8
 11  embarked_Q  1043 non-null   uint8
 12  embarked_S  1043 non-null   uint8
```

```
X=df0.iloc[:,[0,3,4,5,8,10,11]]
y=df0['survived']
X.head(3)
```

	pclass	age	sibsp	parch	sex_female	embarked_C	embarked_Q
0	1	29.0000	0	0	1	0	0
1	1	0.9167	1	2	0	0	0
2	1	2.0000	1	2	1	0	0

Logistic Regression

```
from sklearn.linear_model import LogisticRegression
logreg=LogisticRegression()
logreg.fit(X,y)
y_pred=logreg.predict(X) #판별집단
y_prob=logreg.predict_proba(X) # 사후확률
round(logreg.score(X,y) * 100, 2) #confusion matrix
```

```
79.29
```

로지스틱 회귀 판별 정확도 79.29%

[사후 판별집단 및 사후확률 출력]

```
print(y_pred,'\n',y_prob)
```

```
[1 1 1 ... 0 0 0]
 [[0.08509762 0.91490238]
 [0.33314044 0.66685956]
 [0.04025974 0.95974026]
```

[새로운 개체 판별]

	pclass	age	sibsp	parch	sex_female	embarked_C	embarked_Q
0	1	29.0000	0	0	1	0	0

```
logreg.predict_proba([[1,30,0,1,0,0,1]])
#1등급, 30살, 부모자녀=0, 동반승객=1, 남자, 출항항구 Q 승객 판별
```

```
array([[0.67906074, 0.32093926]])
```

사후확률 사망 67.9%, 생존 32.1%로 예측 => 다음과 같이 판별집단 사망으로 판별

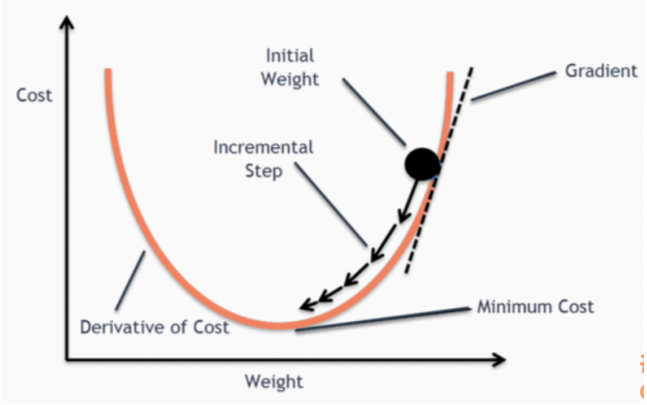
```
logreg.predict([[1,30,0,1,0,0,1]])
```

```
array([0])
```

Stochastic Gradient Descent (SGD) : 실행마다 판별 정확도가 바뀌며 변동이 크다. 42%~78%

목적함수(추정치와 적합치 차이 제곱합)의 최소화 하는 방법 -> 회귀분석의 OLS 추정치와 동일함

Gradient Descent 방법은 기울기(접선, 미분) 활용 최소값을 찾는 과정으로 붙여진 이름



$$mse = \frac{1}{n}\sum_{i=1}^{n}(y_i - (mx_i + b))^2$$

$$\partial/\partial m = \frac{2}{n}\sum_{i=1}^{n} -x_i\,(y_i - (mx_i + b))$$

$$\partial/\partial b = \frac{2}{n}\sum_{i=1}^{n} -(y_i - (mx_i + b))$$

```
from sklearn import linear_model
from sklearn.linear_model import SGDClassifier
sgd=linear_model.SGDClassifier(max_iter=10,loss='log')
sgd.fit(X,y)
y_pred=sgd.predict(X)
y_prob=sgd.predict_proba(X)
sgd.score(X,y) #confusion matrix
```

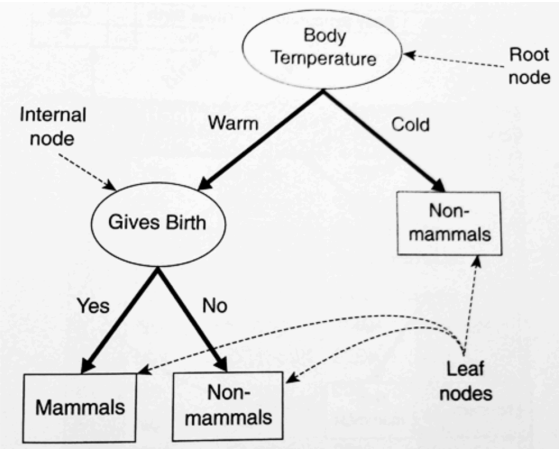
```
  ConvergenceWarning)      ConvergenceWarning)
0.42090124640460214      0.7497603068072867
```

```
print(y_pred,'\n',y_prob)
```

```
[1 1 1 ... 0 0 0]
 [[0.00000000e+000 1.00000000e+000]
 [0.00000000e+000 1.00000000e+000]
 [0.00000000e+000 1.00000000e+000]
```

사후확률이 극단적으로 0, 1의 값을 갖는다.

Decision Tree 의사결정나무 : 93%



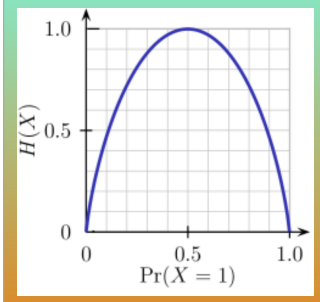
- 결정노드 : 예측(판별)변수 표현
- 분기 branch : 의사결정 규칙 표현
- 리프 노드 : 판별 결과 나타냄, 제일 마지막에 있는 노드를 터미널 노드라고 한다.
- 최상위 노드 = root 노드

[예제] 포유류, 비포유류 구별하는 판별의 의사결정나무
1)판별 정확도 가장 높은 예측변수 [체온] - 차거우면(cold) 비포유류로 판별, Warm(따뜻하면) 보류
2) 보류 집단 소속 개체는 '번식'능력의 여부로 포유류(번식능력),. 그렇지 않으면 비포유류로 판별한다.

의사결정나무는 판별(group prediction, disciminant), 분류(classification, 군빕분석) 모두 적용가능하다.

판별변수가 측정형이면 이진형 변수로 나누는 기준값을 먼저 찾아 이진 범주형 예측변수로 만든다.

판별변수 선택 지표 [Entropy]



- 엔트로피는 주어진 정보(information)의 랜덤성(불확실성 uncertainty, 불순도 impurity)을 의미한다. 엔트로피가 높을수록 주어진 정보로부터 결론을 도출하기 어려워진다.
- 만약 동전을 던져 앞면(X=1)이 나오는 것에 관심을 갖는 경우, 동전이 앞면 나올 확률이 1/2(완전 랜덤)인 경우 엔트로피는 1로 가장 높다. 즉 앞면 나올 확률 예측은 랜덤이다.
- 만약 동전 앞면 나올 확률이 1이라는 정보 혹은 반대로 0이라는 정보가 있다면 동전 던진 결과를 완전하게 예측(판별)할 수 있으므로 엔트로피는 0(homoginiety)이 된다.

E

Play Golf	
Yes	No
9	5

https://www.kdnuggets.com/2020/01/decision-tree-algorithm-explained.html

$$E(S) = \sum_{i}^{c} -p_i log_2 p_i ,$$

$$S : current\ state, c : no\ of\ cells, p_i : prob(i-th\ cell)$$

Entropy(PlayGolf) = Entropy (5,9)
= Entropy (0.36, 0.64)
= - (0.36 $\log_2$ 0.36) - (0.64 $\log_2$ 0.64)
= 0.94

		Play Golf		
		Yes	No	
Outlook	Sunny	3	2	5
	Overcast	4	0	4
	Rainy	2	3	5
				14

현재 날씨 상태로 (outlook) 골프 플레이 예측한 교차표

$T : current\ state, X : pred.\ attribute$

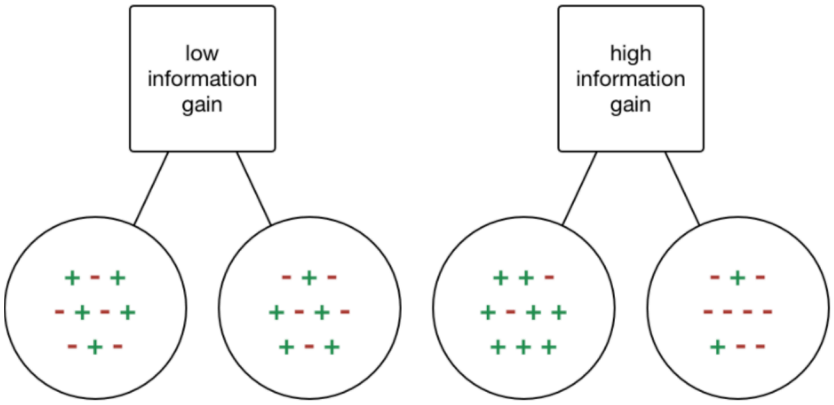
E(PlayGolf, Outlook) = **P**(Sunny)***E**(3,2) + **P**(Overcast)***E**(4,0) + **P**(Rainy)***E**(2,3)

= (5/14)*0.971 + (4/14)*0.0 + (5/14)*0.971

= 0.693

판별변수 선택 지표 **[Information Gain]** $IG(T,X) = Entropy(T) - Entropy(T,X)$

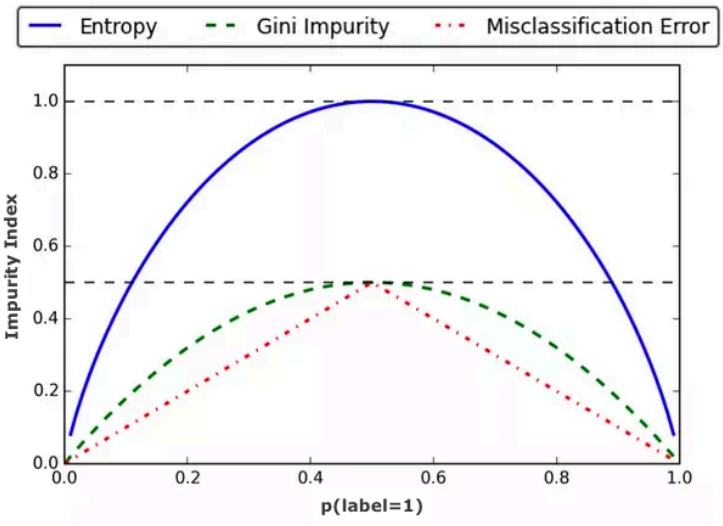
예측변수의 범주로 분할되기 전 엔트로피와 분할 후 엔트로피의 차이를 표현한 것으로 IG 값이 클수록 판별능력이 높은 변수이다.



불순도(오분류)가 낮다 <=> 판별

정보 습득 높음

판별변수 선택 지표 **[Gini Index]** $Gini = 1 - \sum^{c} (p_i)^2$ **: 이진형 예측변수에만 적용**



[의사결정나무 실행]

```
from sklearn.tree import DecisionTreeClassifier
decision_tree=DecisionTreeClassifier()
decision_tree.fit(X,y)
y_pred=decision_tree.predict(X)
y_prob=decision_tree.predict_proba(X)
round(decision_tree.score(X,y) * 100, 2) #confusion matrix
```

```
93.0
```

사후확률 및 판별집단 출력

```
print(y_pred,'\n',y_prob)
```

```
[1 1 0 ... 0 0 0]
 [[0.          1.        ]
 [0.          1.        ]
 [1.          0.        ]
```

[새로운 개체 판별]

	pclass	age	sibsp	parch	sex_female	embarked_C	embarked_Q
0	1	29.0000	0	0	1	0	0

```
decision_tree.predict_proba([[1,30,0,1,0,0,1]])
#1등급, 30살, 부모자녀=0, 동반승객=1, 남자, 출항항구 Q 승객 판별
```

```
array([[1., 0.]])
```

사후확률 사망 100%, 생존 0% => 다음과 같이 판별집단 사망으로 판별

```
decision_tree.predict([[1,30,0,1,0,0,1]])
```

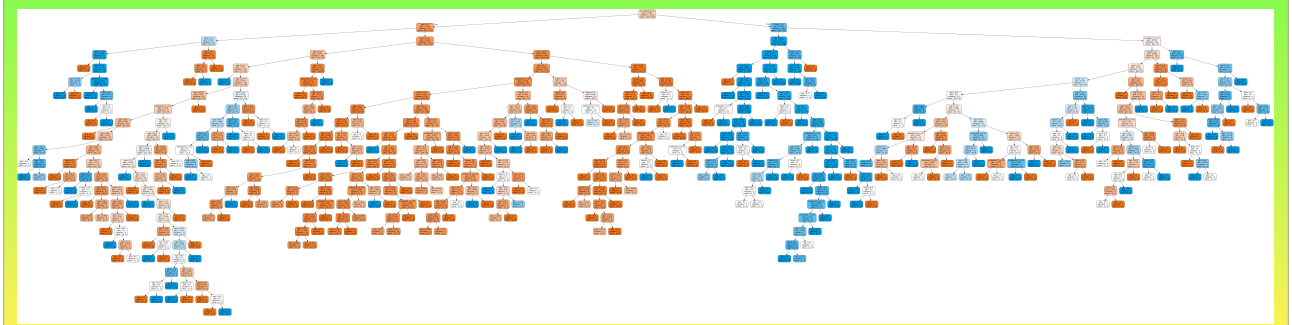
```
array([0])
```

의사결정 나무 판별과정 시각화

```
from sklearn import tree
tree.plot_tree(decision_tree.fit(X,y),feature_names=X.columns)
```

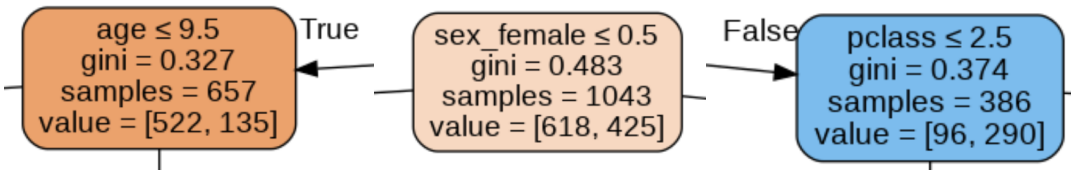
```
[Text(164.99976284983853, 212.71304347826086, 'sex_female <= 0.5\ngini = 0.483\nsamples = 1043\nvalue = [618, 425]'),
 Text(64.21655005382132, 203.2591304347826, 'age <= 9.5\ngini = 0.327\nsamples = 657\nvalue = [522, 135]'),
 Text(9.370075349838537, 193.80521739130435, 'sibsp <= 2.5\ngini = 0.487\nsamples = 43\nvalue = [18, 25]'),
 Text(4.32465016146394, 184.3513043478261, 'age <= 0.375\ngini = 0.198\nsamples = 27\nvalue = [3, 24]'),
 Text(2.8831001076426266, 174.89739130434782, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(5.766200215285253, 174.89739130434782, 'age <= 0.792\ngini = 0.142\nsamples = 26\nvalue = [2, 24]'),
 Text(2.8831001076426266, 165.44347826086957, 'age <= 0.708\ngini = 0.444\nsamples = 3\nvalue = [1, 2]'),
```

```
from sklearn.tree import export_graphviz
from sklearn.externals.six import StringIO
from IPython.display import Image
import pydotplus
dot_data = StringIO()
export_graphviz(decision_tree.fit(X,y),out_file=dot_data,filled=True,rounded=True,
  special_characters=True,feature_names=X.columns)
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
#graph.write_png('decision_tree.png')
Image(graph.create_png())
```

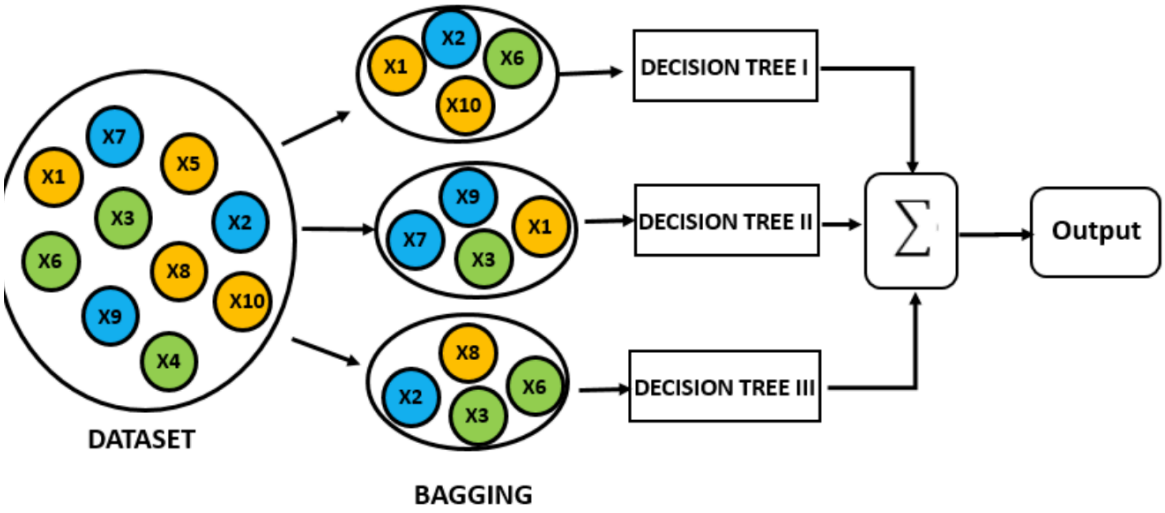


판별조건을 만족하면 왼쪽으로 만족하지 않으면 우측으로 판별된다. 루트 노드(최상위에는 사망 618명, 생존 425명) 처 판별변수 조건이 0.5 이하(sex_female=1 여성, 0은 남성)가 사실인 승객(남성, 657명)은 왼쪽(true)으로 분류되고 여성 승객(386명)은 우측(false) 노드로 가게 된다. X['sex_female'].value_counts()

False 노드의 여성 승객 386명 중 96명 사망, 290명 생존이었다. 이 집단을 탑승권 등급에 의해 2.5등급 이상 이하(1,2등급-3등급) 집단으로 분류하여 판별 도출을 진행하였음을 보여준다.



Random Forest : 93% 매번 차이는 있으나 정확율은 거의 동일



랜덤 포레스트는 의사결정 나무의 판별결과의 정확도를 높이기 위하여 분석 대상 데이터로부터 확률표본(bagging)을 추출하여 각 판별결과 종합하여 최종 판별을 하는 방법이다. 그러므로 확률표본 추출에 따라 결과(정확도 등) 변동될 수 있다.

Bagging (**B**ootstrap **Agg**regat**ing**) 표본추출은 행의 개체를 표본추출하는 것이 아니라 판별변수를 일정 개수(총 판별변수 개수가 p인 경우 일반적으로 분류에는 예측변수의 제곱근, 회귀판별에는 $p/3$, rule of thumbs - 판별변수가 25인 경우 각 bagging에는 5개 예측변수만 있게 된다. 위의 예제는 총 예측변수 $p = 10$개, Bagging에는 4개 판별변수가 있음)

```
from sklearn.ensemble import RandomForestClassifier
random_forest = RandomForestClassifier(n_estimators=50)
random_forest.fit(X,y)
y_pred=random_forest.predict(X)
y_prob=random_forest.predict_proba(X)
random_forest.score(X,y)
```

사후확률 및 판별집단 출력

```
print(y_pred,'\n',y_prob)
```

```
[1 1 0 ... 0 0 0]
 [[0.          1.        ]
 [0.03        0.97      ]
 [0.72        0.28      ]
```

[새로운 개체 판별]

```
random_forest.predict_proba([[1,30,0,1,0,0,1]])
#1등급, 30살, 부모자녀=0, 동반승객=1, 남자, 출항항구 Q 승객 판별
```

```
array([[0.86, 0.14]])
```
사후확률 사망 86%, 생존 14% => 다음과 같이 사망으로 판별

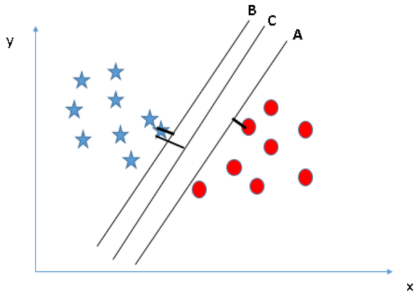
```
random_forest.predict([[1,30,0,1,0,0,1]])
```
```
array([0])
```

Support Vector Machine

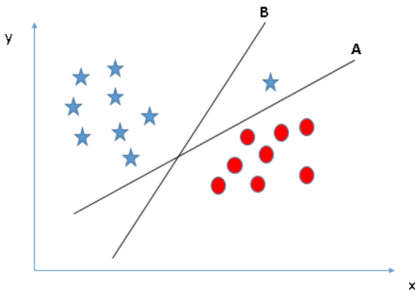
분류 또는 회귀 문제 모두에 사용할 수 있는 머신러닝기법이다. (주로 분류 문제에 사용) p 차원 공간의 개체들을 가장 잘 분류(판별)하는 하이퍼 플레인(hyper-plane)을 찾아 분류를 수행한다.

https://www.analyticsvidhya.com/blog/2017/09/understaing-support-vector-machine-example-code/

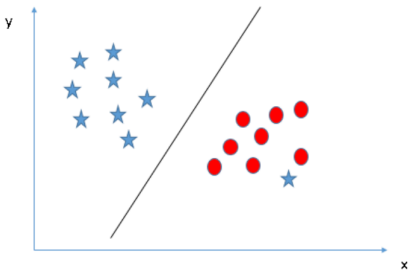
하이퍼플레인(여기 예제에서는 선형 하이퍼플레인)과 개별 개체간의 거리를 margin (그림 1의 직선 거리)가 최대화 되는 하이퍼 플레인이 가장 좋은 판별식이다. 그림1의 경우 하이퍼플레인 C가 가장 좋은 판별규칙이다. 그림2의 경우는 거리(margin) 측면에서는 A가 더 우수한 판별식이나 판별 오류가 존재하므로 거리 측면에서는 판별 능력이 낮으나 오분류가 없는 A를 최적 판별 규칙으로 선택한다. 그러므로 그림 3의 경우처럼 오분류가 발생할 수 밖에 없는 상황에서는 거리 최대화(최대 마진) 판별 규칙을 선택하게 된다.



(그림 1)

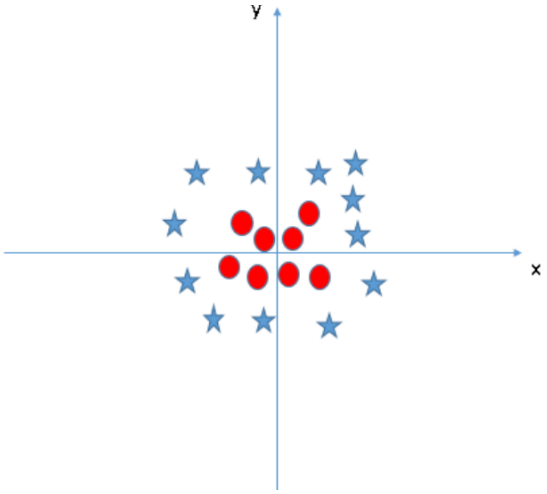


(그림2)

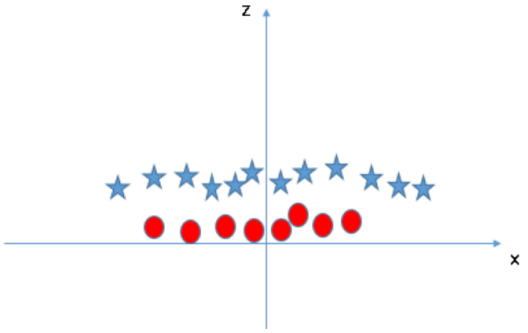


(그림3)

그림 4의 경우 개체를 판별하는 직선 형태의 하이퍼플레인으로는 개체 판별이 불가능하다. 대신 $z^2 = x^2 + y^2$ 변환하여 개체를 표현하면 적절한 직선 하이퍼플레인이 됩니다.



(그림 4)



(그림5)

SVM (선형)

```
from sklearn import svm
SVM_linear=svm.LinearSVC()
SVM_linear.fit(X, y)
round(SVM_linear.score(X,y) * 100, 2)
```

```
/usr/local/lib/python3
  "the number of iterat
79.39
```

판별규칙 convergence 도달하지 못하여 매번 판별 결과가 달라져 confusion matrix이 달라져 정확비율 달라짐.

판별집단만 가능

```
SVM_linear.predict(X)
```

```
array([1, 1, 1, ..., 0, 0, 0])
```

SVM (비선형) 판별 정확도 62.4%

```
from sklearn import svm
SVM=svm.SVC(probability=True)
SVM.fit(X, y)
round(SVM.score(X,y) * 100, 2)
```

```
62.42
```

=>판별규칙이 convergence 도달해 경고 warning 없어 항상 정확율 동일함

판별집단, 사후확률 출력

```
SVM.predict(X),SVM.predict_proba(X)
```

```
(array([0, 1, 1, ..., 0, 0, 0]), array([[0.31615741, 0.68384259],
        [0.01200003, 0.98799997],
        [0.01052188, 0.98947812],
```

[새로운 개체 판별]

```
SVM.predict_proba([[1,30,0,1,0,0,1]])
#1등급, 30살, 부모자녀=0, 동반승객=1, 남자, 출항항구 Q 승객 판별
```

```
array([[0.51084126, 0.48915874]])
```

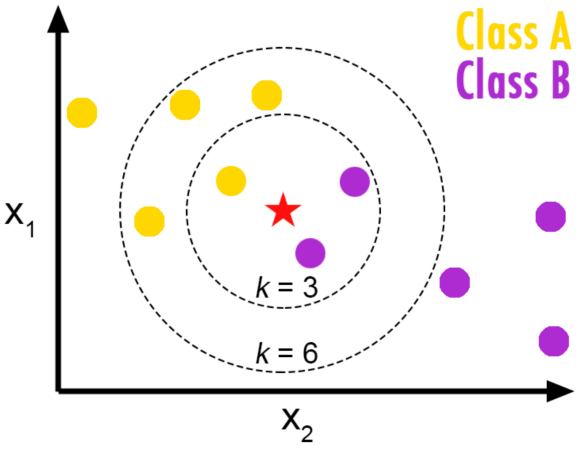
사후확률 사망 51%, 생존 49% => 다음과 같이 사망으로 판별

```
SVM.predict([[1,30,0,1,0,0,1]])
```

```
array([0])
```

K Nearest Neighbor 83.41%

판별하려는 개체(★)에 가까운 그룹 개체가 많은 그룹으로 판별하는 비모수 판별방법이다. K=3 (집단 판별에 사용되는 이웃 개체 수)인 경우는 개체 2개가 속한 집단 B에 분류되고, K=6을 사용하면 집단 A에 분류된다.



K=5 5개 이웃개체 활용

```
from sklearn.neighbors import KNeighborsClassifier
knn=KNeighborsClassifier(n_neighbors=5)
knn.fit(X,y)
Y_pred = knn.predict(X)
round(knn.score(X,y) * 100, 2)
```

```
83.41
```

판별집단, 사후확률 출력

```
knn.predict(X),knn.predict_proba(X)
```

5개 이웃으로 개체 판별하므로 5개 개체 중 많은 집단에 판별한 집단으로 판별하여 사후확률은 (집단에 속한 개체 개수/5)

```
(array([1, 1, 1, ..., 0, 0, 0]), array([[0.2, 0.8],
        [0. , 1. ],
        [0.2, 0.8],
```

[새로운 개체 판별]

```
knn.predict_proba([[1,30,0,1,0,0,1]])
#1등급, 30살, 부모자녀=0, 동반승객=1, 남자, 출항항구 Q 승객 판별
```

```
array([[0.6, 0.4]])
```
사후확률 사망 60%, 생존 40% => 다음과 같이 사망으로 판별

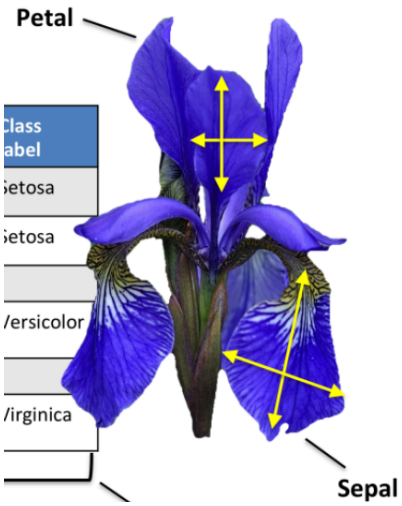
```
knn.predict([[1,30,0,1,0,0,1]])
```
```
array([0])
```

Fisher 방법 (LDA)

iris 데이터 (n=150, 집단변수=3개 종, 예측변수=Petal 꽃잎 넓이, 길이 Sepal 꽃받침조각 길이, 넓이)

```
import pandas as pd
iris=pd.read_csv('https://vincentarelbundock.github.io/Rdatasets/csv/datasets/iris.csv')
iris.info()
```

```
RangeIndex: 150 entries, 0 to 149
Data columns (total 6 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Unnamed: 0     150 non-null    int64  
 1   Sepal.Length   150 non-null    float64
 2   Sepal.Width    150 non-null    float64
 3   Petal.Length   150 non-null    float64
 4   Petal.Width    150 non-null    float64
 5   Species        150 non-null    object 
```



Iris Versicolor Iris Setosa Iris Virginica

전처리 - 예측변수 표준화

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X=sc.fit_transform(iris.iloc[:,1:5])
```

```
1  iris.iloc[:,1:5].columns
```

```
Index(['Sepal.Length', 'Sepal.Width', 'Petal.Length', 'Petal.Width'], dtype='object')
```

주성분변수 저장 및 부하 출력

```
from sklearn.decomposition import PCA
pca=PCA(n_components=2) #주성분 2개 계산
df_pca=pca.fit(X).transform(X) #주성분 변수
```

```
print(iris.iloc[:,1:5].columns,"\n",pca.components_.T)  #loadings
```

```
Index(['Sepal.Length', 'Sepal.Width', 'Petal.Length', 'Petal.Width']
 [[ 0.52106591  0.37741762]
 [-0.26934744  0.92329566]
 [ 0.5804131   0.02449161]
 [ 0.56485654  0.06694199]]
```

주성분변수 저장 및 부하 출력

```
iris_pca=pd.DataFrame(df_pca,index=iris.index,columns=['SL_PT','SW'])
iris_pca.head(3)
```

주성분변수 iris_pca에 'SL_PT','SW' 이름으로 데이터프레임 저장

	SL_PT	SW
0	-2.264703	0.480027
1	-2.080961	-0.674134
2	-2.364229	-0.341908

Fisher 판별분석 (그래프 그리기 위하여 차원을 2개로 하였음)

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
y=iris.iloc[:,5].values #문자열을 숫자화 하지만 문자정보 그대로 있음
da=LinearDiscriminantAnalysis(n_components=2)
model_da=da.fit(X,y)
```

집단을 가장 잘 판별하는 차원을 제공하는 차원 계산(n_components=2)
da.fit(X, y).transform(X)에는 차원 저장

```
import numpy as np
np.unique(y,return_counts=True)
```

```
(array(['setosa', 'versicolor', 'virginica'], dtype=object),
 array([50, 50, 50]))
```

축약 2차원(주성분과 동일함)정보를 iris_dim 데이터프레임에 'dim1','dim2' 이름으로 저장

```
iris_dim=pd.DataFrame(lda.fit(X, y).transform(X),
index=iris.index,columns=['dim1','dim2'])
```

Fisher 판별분석 집단 소속 사후확률 출력 및 판별집단

```
probs_da=model_da.predict_proba(X)
probs_da[49:52]
```

```
index=49(50번째)분꽃은 setosa 확률 1, 'versicolor'=0 'virginica'=0
index=50(51번째)분꽃은 setosa 확률 0, 'versicolor'=0.99 'virginica'=0.0001
index=51(52번째)분꽃은 setosa 확률 0, 'versicolor'=0.99 'virginica'=0.0007
```

```
array([[1.00000000e+00, 2.32225794e-20, 4.24175670e-40],
       [1.96973176e-18, 9.99889412e-01, 1.10587759e-04],
       [1.24287800e-19, 9.99257470e-01, 7.42529660e-04]])
```

```
y_pred=model_da.predict(X)
iris_da=pd.concat([iris_dim,pd.DataFrame(y_pred,index=iris.index,
columns=["result"])],axis=1)
iris_da.iloc[49:52,:]
```

	dim1	dim2	result
49	7.671967	-0.134894	setosa
50	-1.459275	0.028544	versicolor
51	-1.797706	0.484386	versicolor

위의 사후확률 결과대로 판별되어 저장되어 있음

Confusion matrix and Accuracy

```
from sklearn.metrics import confusion_matrix,
classification_report, precision_score
print(confusion_matrix(y_pred,y))
print(classification_report(y,y_pred, digits=3))
```

판별정확도 98%, setosa 100%, versicolor 98%(1개 오분류), virginica 96.1%(2개 오분류)

```
[[50  0  0]
 [ 0 48  1]
 [ 0  2 49]]
              precision    recall  f1-score   support

      setosa      1.000     1.000     1.000        50
  versicolor      0.980     0.960     0.970        50
   virginica      0.961     0.980     0.970        50

    accuracy                          0.980       150
   macro avg      0.980     0.980     0.980       150
weighted avg      0.980     0.980     0.980       150
```

원데이터, 주성분변수, 판별차원 및 판별집단 데이터 합치기

```
iris_pca_da=pd.concat([iris,iris_da,iris_pca],axis=1)
iris_pca_da.head(3)
```

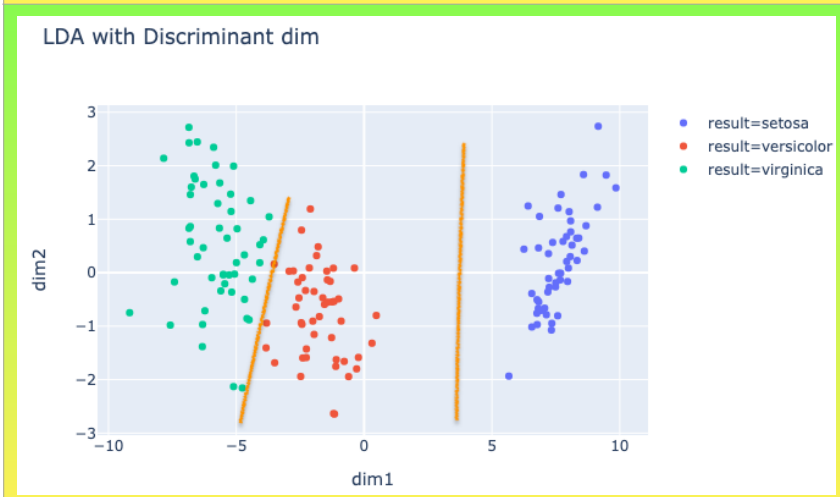
Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species	dim1	dim2	result	SL_PT	SW
5.1	3.5	1.4	0.2	setosa	8.061800	0.300421	setosa	-2.264703	0.480027
4.9	3.0	1.4	0.2	setosa	7.128688	-0.786660	setosa	-2.080961	-0.674134
4.7	3.2	1.3	0.2	setosa	7.489828	-0.265384	setosa	-2.364229	-0.341908

판별그래프 (주성분 변수)

```
import plotly.express as px
fig = px.scatter(iris_pca_da,x="SL_PT",y="SW",
color="result",title="LDA with PCA")
fig.show()
```

판별그래프 (차원축소 dim) 주성분 그래프와 정확하게 동일하다. 단지 x-차원 값의 부호만 다름

```
import plotly.express as px
fig = px.scatter(iris_pca_da,x="dim1",y="dim2",
color="result",title="LDA with Discriminant dim")
fig.show()
```



Fisher QDA

판별분석 결과 저장

```
import pandas as pd
iris=pd.read_csv('https://vincentarelbundock.github.io/Rdatasets/csv/datasets/iris.csv')
```

```
LDA와 달리 집단을 가장 잘 판별하는 차원을 제공하는 (n_components=2) 없음
da.fit(X, y).transform(X)도 제공하지 않음
```

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X=sc.fit_transform(iris.iloc[:,1:5])
from sklearn.discriminant_analysis import
QuadraticDiscriminantAnalysis
y=iris.iloc[:,5].values
da=QuadraticDiscriminantAnalysis()
model_da=da.fit(X,y)
```

사후 확률 및 판별집단 : 판별 정확도 98% (LDA 판별과 동일함, 사망, 생존 정확율)

```
probs_da=model_da.predict_proba(X)
probs_da[49:52]
```

```
y_pred=model_da.predict(X)
from sklearn.metrics import confusion_matrix,
classification_report, precision_score
print(confusion_matrix(y_pred,y))
print(classification_report(y,y_pred, digits=3))
```

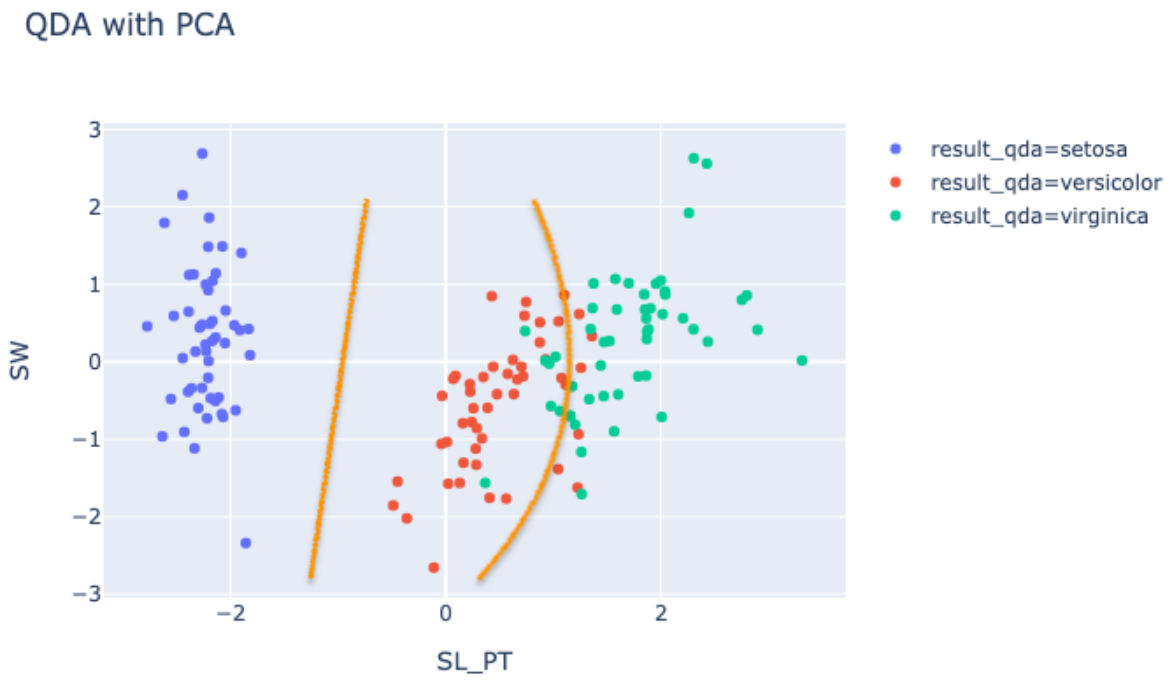
```
[[50  0  0]
 [ 0 48  1]
 [ 0  2 49]]
              precision    recall  f1-score   support

      setosa      1.000     1.000     1.000        50
  versicolor      0.980     0.960     0.970        50
   virginica      0.961     0.980     0.970        50

    accuracy                          0.980       150
   macro avg      0.980     0.980     0.980       150
weighted avg      0.980     0.980     0.980       150
```

판별결과 시각화

```
iris_da=pd.DataFrame(y_pred,index=iris.index,columns=["result_qda"])
iris_pca_da=pd.concat([iris,iris_pca,iris_da],axis=1)
iris_pca_da.head(3)
```

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species	SL_PT	SW	result_qda
5.1	3.5	1.4	0.2	setosa	-2.264703	0.480027	setosa
4.9	3.0	1.4	0.2	setosa	-2.080961	-0.674134	setosa
4.7	3.2	1.3	0.2	setosa	-2.364229	-0.341908	setosa

```
import plotly.express as px
fig = px.scatter(iris_pca_da,x="SL_PT",y="SW",
color="result_qda",title="QDA with PCA")
fig.show()
```



모집단 3개 이상 머신러닝 기법 활용

데이터

```
features=iris.iloc[:,[1,2,3,4]]
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X=pd.DataFrame(sc.fit_transform(features),columns=features.columns)
y=iris['Species']
X.head(3)
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width
0	-0.900681	1.019004	-1.340227	-1.315444
1	-1.143017	-0.131979	-1.340227	-1.315444
2	-1.385353	0.328414	-1.397064	-1.315444

LDA 방법과 달리 X는 데이터프레임을 그대로 사용한다. 하여 표준화한 배열을 다시 데이터프레임으로 만들어줘야 한다. 예측변수들의 단위가 상이한 경우에는 단위 표분화 작업이 필요하다.
분꽃 데이터는 길이, 넓이 모두 동일 단위이고 크기도 한자리로 동일하므로 굳이 표준화 할 필요는 없다. 하여, 다음의 머신러닝 기법 사례에서는 다음과 같이 원 데이터를 사용하였다.

```
X=iris.iloc[:,[1,2,3,4]]
y=iris['Species']
X.head(3)
```

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width
5.1	3.5	1.4	0.2
4.9	3.0	1.4	0.2
4.7	3.2	1.3	0.2

판별 대상 집단

```
9  y.value_counts()
```

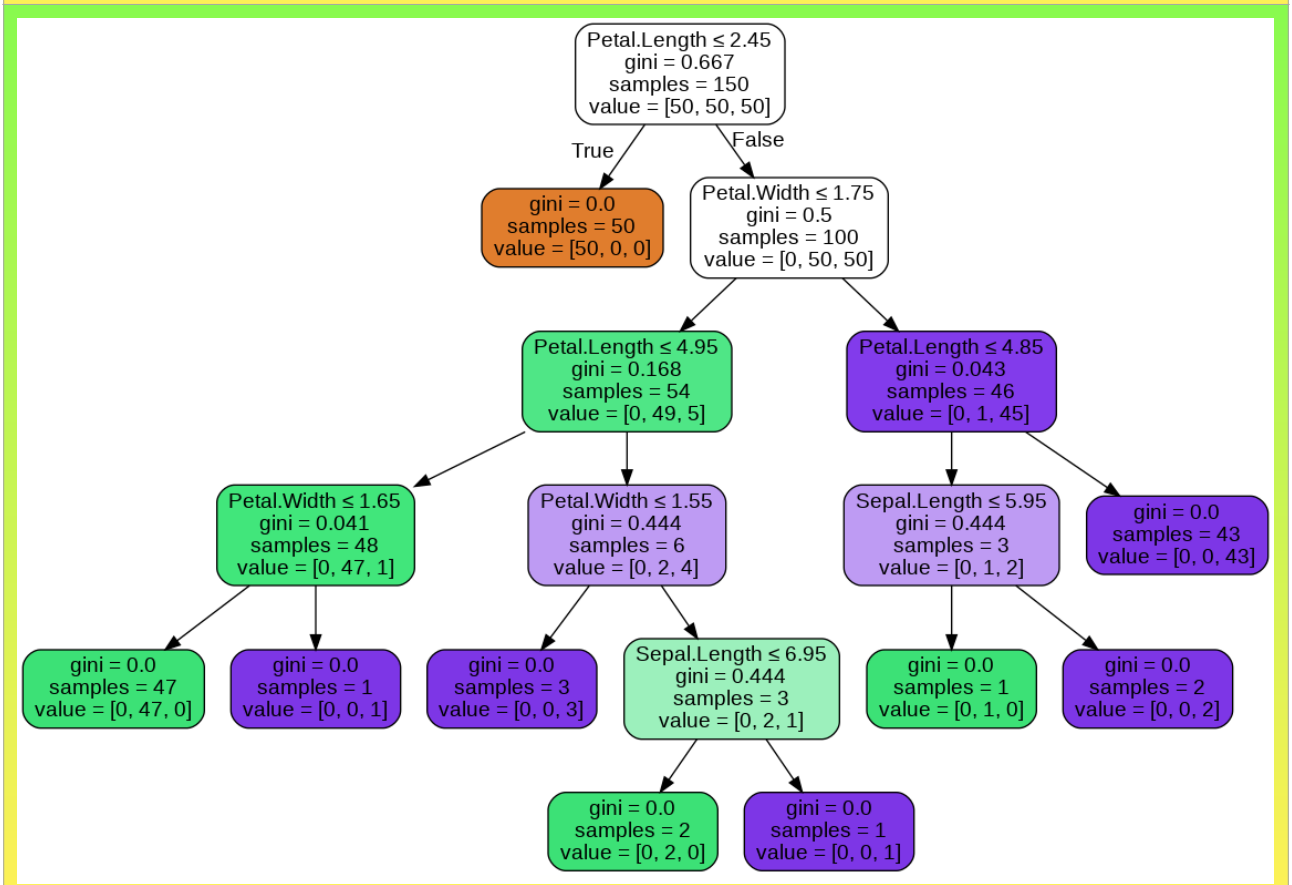
```
setosa        50
virginica     50
versicolor    50
```

의사결정나무

```
100.0
```

```
from sklearn.tree import DecisionTreeClassifier
decision_tree=DecisionTreeClassifier()
decision_tree.fit(X,y)
y_pred=decision_tree.predict(X)
y_prob=decision_tree.predict_proba(X)
round(decision_tree.score(X,y) * 100, 2)
```

```
from sklearn.tree import export_graphviz
from sklearn.externals.six import StringIO
from IPython.display import Image
import pydotplus
dot_data = StringIO()
export_graphviz(clf,out_file=dot_data,filled=True,rounded=True,
  special_characters=True,feature_names=X.columns)
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
graph.write_png('diabetes.png')
Image(graph.create_png())
```




```
decision_tree.predict_proba([[5,3,3,2]])#꽃받침 길이/넓이|꽃잎 길이/넓이
```

```
array([[0., 0., 1.]])
```

setosa 0%, versicolor 0%, virginica 100% -> virginica 판별

```
decision_tree.predict([[5,3,3,2]])
```

```
array(['virginica'
```

렌덤 포레스트 ⮕ 100.0

```
from sklearn.ensemble import RandomForestClassifier
random_forest=RandomForestClassifier(n_estimators=50)
random_forest.fit(X,y)
y_pred=random_forest.predict(X)
y_prob=random_forest.predict_proba(X)
random_forest.score(X,y)
```

```
random_forest.predict_proba([[5,3,3,2]]) #꽃받침 길이, 넓이, 꽃잎 길이, 넓이
```

```
array([[0.04, 0.46, 0.5 ]])
setosa 4%, versicolor 46%, virginica 50% -> virginica 판별
```

```
random_forest.predict([[5,3,3,2]])   array(['virginica']
```

K-neighbor 판별 ⮕ 96.67

```
from sklearn.neighbors import KNeighborsClassifier
knn=KNeighborsClassifier(n_neighbors=5)
knn.fit(X,y)
Y_pred = knn.predict(X)
round(knn.score(X,y) * 100, 2)
```

```
knn.predict_proba([[5,3,3,2]]) #꽃받침 길이, 넓이, 꽃잎 길이, 넓이
```

```
array([[0., 1., 0.]])
setosa 0%, versicolor 100%, virginica 0% -> versicolor 판별
```

```
knn.predict([[5,3,3,2]]).  ⮕  array(['versicolor']
```

SVM ⮡ 97.33

```
from sklearn import svm
SVM=svm.SVC(probability=True)
SVM.fit(X, y)
round(SVM.score(X,y) * 100, 2)
```

```
SVM.predict_proba([[5,3,3,2]]) #꽃받침 길이, 넓이, 꽃잎 길이, 넓이
```

```
array([[0.06947909, 0.91207031, 0.0184506 ]])
setosa 7%, versicolor 91%, virginica 2% -> versicolor 판별
```

```
SVM.predict([[5,3,3,2]]).  ⮡ array(['versicolor']
```

선형 SVM 96.67

Not convergence warning : 판별결과도 다른 방법과 상이함
(Warning 있는 경우 사용하지 말자)

```
from sklearn import svm
SVM_linear=svm.LinearSVC()
SVM_linear.fit(X, y)
round(SVM_linear.score(X,y) * 100, 2)
```

```
사후 확률 제공하지 않음
```

```
SVM_linear.predict([[5,3,3,2]]) #꽃받침 길이, 넓이, 꽃잎 길이, 넓이
⮡ array(['setosa']
```