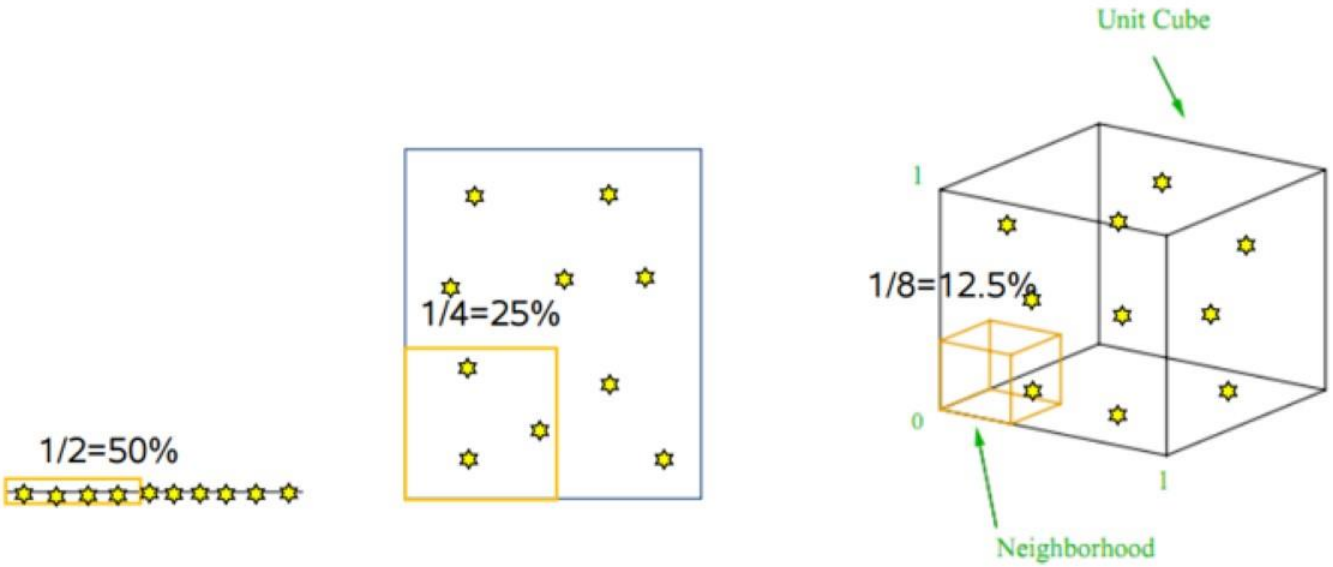
II. 머신러닝 모델

□ 차원축소

- 차원 축소의 필요성
  - 관측치의 수는 한정되어 있음
    - 각 변수의 50%영역에 해당하는 자료를 가지고 있다고 할 때,
    - 차원이 커질 수록 한정된 자료는 커진 차원의 패턴을 잘 설명하지 못함
    - 차원이 증가함에 따라 model complexity 가 기하급수적으로 높아짐



The Elements of Statistical Learning", Tibshirani and Friedman

□ 차원축소

- 상관계수가 높은 변수 중 일부를 분석에서 제외?
  - 정보의 손실 발생
  - 상관계수가 0.8이라고 하면, 0.2에 해당하는 정보는 버려지게 됨

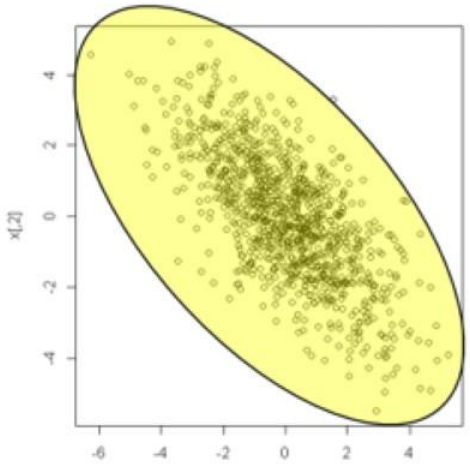
- 차원을 줄이면서 정보의 손실을 최소화하는 방법
  - Principal component을 활용

- 이외의 방법
  - 변수 선택법
  - penalty 기반 regression
  - convolutional neural network
  - drop out & bagging

- 차원축소 활용
  - 차원축소를 통해 좀더 데이터를 잘 설명할 수 있는 잠재적인 요소를 추출
  - 이미지 분류, 시맨틱, 토픽 분류 등

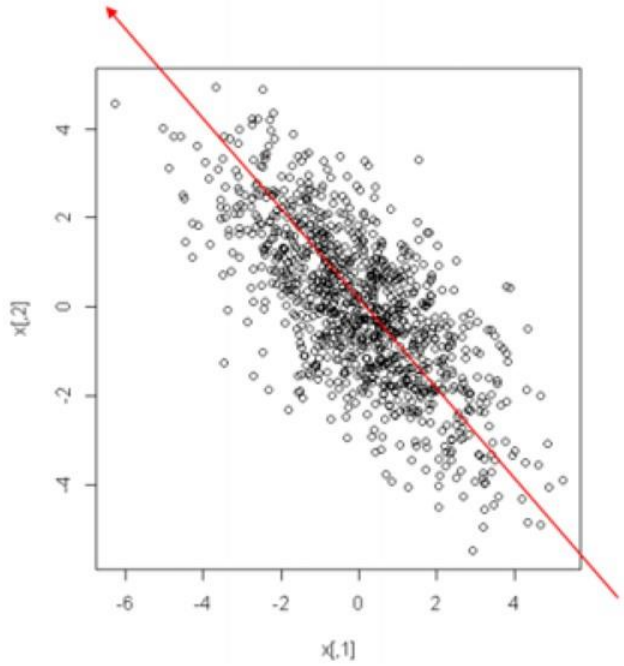
□ PCA

- 공분산 행렬의 개념
  - 공분산 행렬(Covariance matrix)의 정의
    - X_1, X_2 이 음의 상관관계를 가지므로, 둘의 공분산은 음수일 것. X가 centering 되어 있다면,
    - $$Cov\left(\mathbf{X} = \begin{bmatrix} X_1 \\ X_2 \end{bmatrix}\right) = \begin{bmatrix} Var(X_1) & Cov(X_1, X_2) \\ Cov(X_2, X_1) & Var(X_2) \end{bmatrix} = \begin{bmatrix} 3 & -2 \\ -2 & 3 \end{bmatrix}$$
    - X가 centering 되어 있다면, $Cov(\mathbf{X}) = (\mathbf{X}^T\mathbf{X})/(n-1)$
  - 점과 내적연산을 하는 경우, 점의 위치를 이동시켜 해당 공분산 구조와 비슷한 형태를 가지게 됨



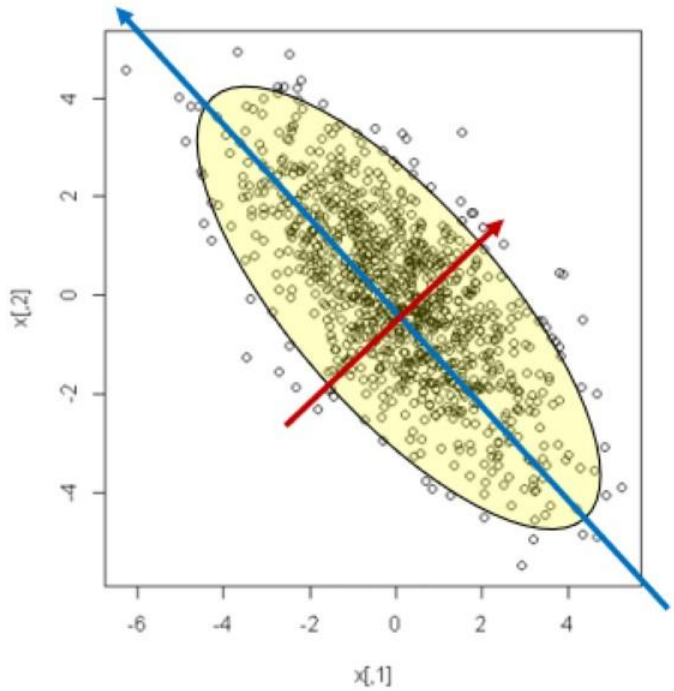
□ PCA

- Principal Components 의 개념
  - 차원을 줄이면서 정보의 손실을 최소화하는 방법
    - 더 적은 개수로 데이터를 충분히 잘 설명할 수 있는 새로운 축을 찾아냄



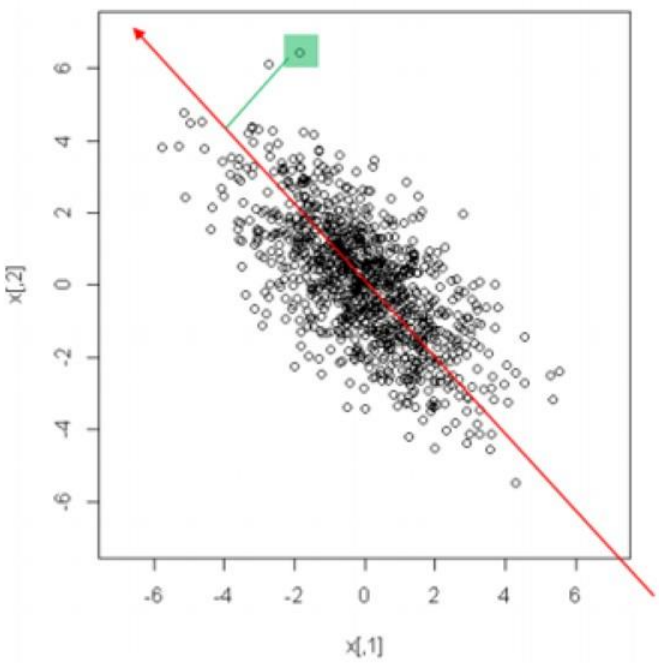
□ PCA

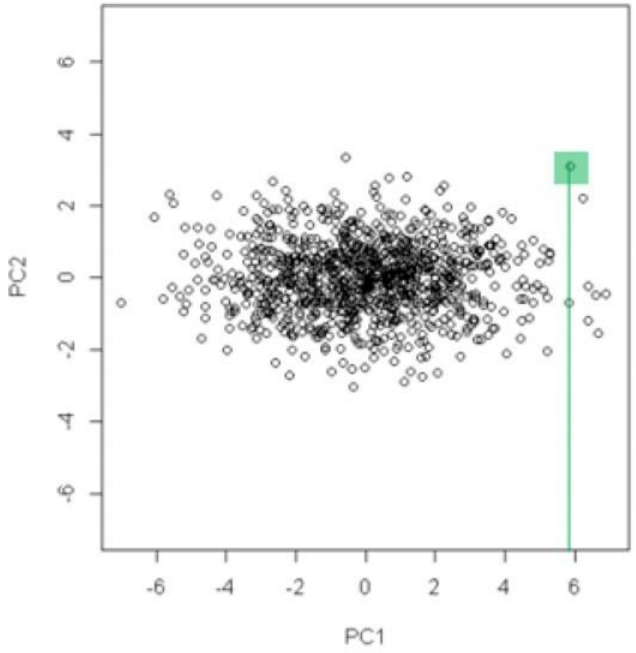
- Principal Components 의 개념
  - 공분산이 데이터의 형태를 변형시키는 방향의 축과 그것에 직교하는 축을 찾아내는 과정
  - 2차원의 경우 공분산이 나타내는 타원의 장축과 단축



□ PCA

- Principal Components = PC = PC score
  - 찾아낸 새로운 축에서의 좌표값을 의미
  - 새로운 축에 내린 정사영

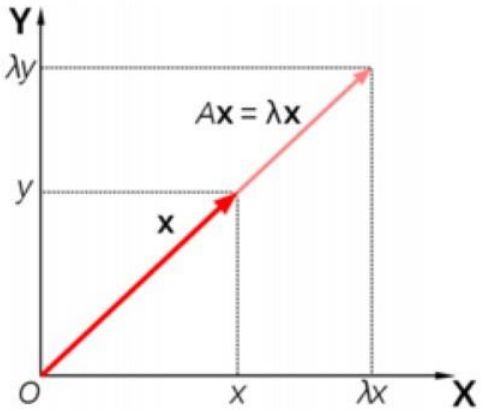




- PCA는 데이터의 **분산(variance)**을 최대한 보존하면서 서로 직교하는 새 기저(축)를 찾아, 고차원 공간의 표본들을 선형 연관성이 없는 저차원 공간으로 변환하는 기법

□ PCA

- eigen value, eigen vector
  - 정방행렬 A에 대하여,
  - 아래를 만족할 경우 v는 고유벡터(eigen vector) 이고 λ는 고유값(eigen value)이다.
    - $Av = \lambda v$
    - 아래와 같이 A는 v를 선형변환한다.



- Transpose and symmetric

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{bmatrix} \Rightarrow A' = A^T = A^t = \begin{bmatrix} a_{11} & a_{21} \\ a_{12} & a_{22} \\ a_{13} & a_{23} \end{bmatrix}$$

  - Symmetric 정의 : A = A^T 일때 A는 Symmetric 이다.

- digonal matrix

$$diag(a_1, \ldots, a_n) = \begin{bmatrix} a_1 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & a_n \end{bmatrix}$$

$$Orthonormal\ \ vectors:$$

$$\boldsymbol{q}_i{}^T \boldsymbol{q}_j = \begin{cases} 0 & if \quad i \neq j \\ 1 & if \quad i = j \end{cases}$$

□ PCA

- 행렬 분해

$$Cov\left(\mathbf{X} = \begin{bmatrix} X_1 \\ X_2 \end{bmatrix}\right) = \begin{bmatrix} Var(X_1) & Cov(X_1, X_2) \\ Cov(X_2, X_1) & Var(X_2) \end{bmatrix}$$

$$\mathrm{C} = P \sum P^T$$

P : $n \times n$ 직교행렬
Σ : $n \times n$ 정방행렬
P^T : 행렬 P의 전치행렬

$$C = [e_1 \dots e_n] \begin{bmatrix} \lambda_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \lambda_n \end{bmatrix} \begin{bmatrix} e_1^t \\ \dots \\ e_n^t \end{bmatrix}$$

- 즉, 데이터의 공분산 행렬이 고유벡터와 고유값으로 분해될 수 있고, 이렇게 분해된 고유벡터를 이용해서 입력데이터를 선형변환하는게 PCA

□ PCA

- 수학적 개념 _ Singular Value Decomposition (SVD)
  - Singular Value Decomposition (SVD)
  - n x p Matrix X를 아래와 같은 요소로 나누는 것을 SVD라 한다.
    - $\mathbf{X} = \mathbf{UDV^T}$
    - $\mathbf{U}$: n × p, $\mathbf{D}$: p × p, $\mathbf{V}$: p × p
    - $\mathbf{V^TV} = \mathbf{I}_p$, $\mathbf{U^TU} = \mathbf{I}_p$, $\mathbf{D}$: diagonal matrix
    - Column vectors of $\mathbf{V}$: eigen vectors of $\mathbf{X^TX}$
    - Diagonal entries of $\mathbf{D}$: eigen values of $\mathbf{X^TX}$
  - 위와 같이 SVD를 통하여, 임의의 matrix의 공분산 구조 행렬의 eigen vector, eigen value를 얻을 수 있다.
    - X이 centered 되어 있다면, $\mathbf{X^TX}$ 는 X의 공분산 구조임

□ PCA

- 수학적 개념 _ SVD와 eigen value, eigen vector 와의 연관성
  - $V = [v_1 \ \ldots \ v_p]$에서, $v_1 \ \ldots \ v_p$ 가 eigen vectors인 이유
    - SVD에 의해, $\mathbf{X} = \mathbf{UDV^T}$ 이므로,
      - $\mathbf{X}^T\mathbf{X} = \mathbf{V}\mathbf{D}^T\mathbf{U}^T\mathbf{U}\mathbf{D}\mathbf{V}^T = \mathbf{V}\mathbf{D}^2\mathbf{V}^T$

        $(\mathbf{X}^T\mathbf{X})\mathbf{V} = \mathbf{V}\mathbf{D}^2\mathbf{V}^T\mathbf{V} = \mathbf{V}\mathbf{D}^2$

        $(\mathbf{X}^T\mathbf{X})[v_1 \ \cdots \ v_p] = [v_1 \ \cdots \ v_p]\mathbf{D}^2 = [d_1^2 v_1 \ \cdots \ d_p^2 v_p]$
      - $(\mathbf{X}^T\mathbf{X})v_i = d_i^2 v_i$, $i=1,\ldots,p$
        - v_i: eigen vectors
        - d_i^2: eigen values

□ [실습] iris 차원축소

- Scikit-learn 에 내장되어 있는 데이터셋 iris 이용
  - 데이터 준비

```
import pandas as pd
import numpy as np

from sklearn import datasets
```

```
iris = datasets.load_iris()
```

```
x = iris.data[:,[0,2]] #꽃받침(sepal)의 길이와 꽃잎의 길이
y = iris.target
```

```
feature_name = [iris.feature_names[0],iris.feature_names[2]]
x_data = pd.DataFrame(x, columns = feature_name)
x_data.head()
```

	sepal length (cm)	petal length (cm)
0	5.1	1.4

```
y_data = pd.DataFrame(y, columns = ["target"])
y_data.head()
```

	target
0	0

□ [실습] iris 차원축소

- Scikit-learn 에 내장되어 있는 데이터셋 iris 이용
  - 모델생성

```
from sklearn.decomposition import PCA
```

```
pca = PCA(n_components=2) # 2개차원
pca.fit(x_data)
```

```
PCA(copy=True, iterated_power='auto', n_components=2, random_state=None,
    svd_solver='auto', tol=0.0, whiten=False)
```

  - 속성 확인, Pcscore계산

```
pca.explained_variance_  # 고유값
```

```
array([3.66189877, 0.1400726 ])
```

```
pca.components_   # 고유벡터
```

```
array([[ 0.39360585,  0.9192793 ],
       [-0.9192793 ,  0.39360585]])
```

```
PCscore = pca.transform(x_data) # PC score
PCscore[0:5]
```

```
array([[-2.46024094, -0.24479165],
       [-2.53896211, -0.06093579],
       [-2.70961121,  0.08355948],
       [-2.56511594,  0.25420858],
       [-2.49960153, -0.15286372]])
```

□ [실습] iris 차원축소

- Scikit-learn 에 내장되어 있는 데이터셋 iris 이용
  - PCscore 직접 구하기

```
eigens_v=pca.components_.transpose()
print(eigens_v)
```

```
[[ 0.39360585 -0.9192793 ]
 [ 0.9192793   0.39360585]]
```

```
mX=np.matrix(x)

for i in range(x.shape[1]):
    mX[:,i]=mX[:,i]-np.mean(x[:,i])

mX_df=pd.DataFrame(mX)
```

```
(mX*eigens_v)[0:5]
```

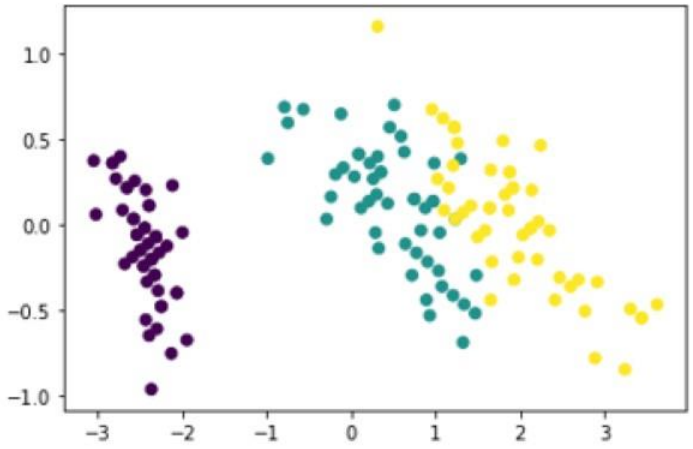
```
matrix([[-2.46024094, -0.24479165],
        [-2.53896211, -0.06093579],
        [-2.70961121,  0.08355948],
        [-2.56511594,  0.25420858],
        [-2.49960153, -0.15286372]])
```

□ [실습] iris 차원축소

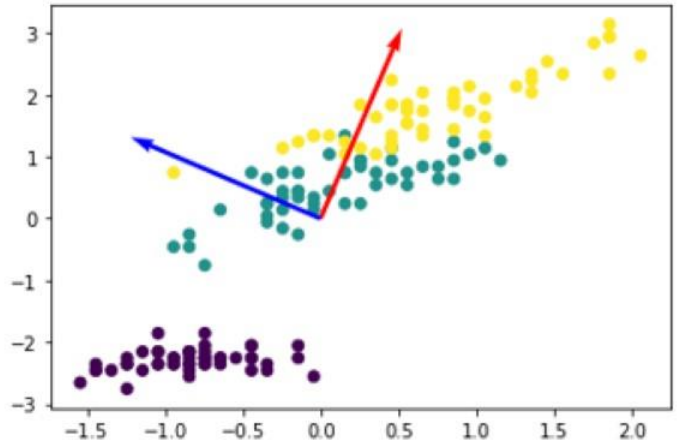
- Scikit-learn 에 내장되어 있는 데이터셋 iris 이용
  - 시각화

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

plt.scatter(PCscore[:,0],PCscore[:,1], c=y)
plt.show()
```




```
plt.scatter(mX_df[0],mX_df[1], c=y)
origin = [0], [0] # origin point
plt.quiver(*origin, eigens_v[0,:], eigens_v[1,:], color=['r','b'], scale=3)
plt.show()
```



□ [실습] iris 차원축소

- 로지스틱회귀분석
  - Scikit-learn 에 내장되어 있는 데이터셋 iris 이용
  - Iris 4개 특성 변수 모두 사용해 PCA 수행

```
x2 = iris.data

pca2 = PCA(n_components=4)
pca2.fit(x2)
```

```
PCA(copy=True, iterated_power='auto', n_components=4, random_state=None,
    svd_solver='auto', tol=0.0, whiten=False)
```

```
pca2.explained_variance_
```

```
array([4.22824171, 0.24267075, 0.0782095 , 0.02383509])
```

```
PCscore2 = pca2.transform(x2)[:,0:2]
```

□ [실습] iris 차원축소

- 로지스틱회귀분석
  - Scikit-learn 에 내장되어 있는 데이터셋 iris 이용
  - 원 데이터와 PCscore을 이용해 각각 모델 생성 후 성능 비교

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
```

```
clf = LogisticRegression(solver="sag", multi_class="multinomial",
                         max_iter=10000).fit(x2,y)
```

```
clf2 = LogisticRegression(solver="sag",multi_class="multinomial").fit(PCscore2,y)
```

```
y_pred2 = clf2.predict(PCscore2)
```

```
y_pred = clf.predict(x2)
```

```
confusion_matrix(y, y_pred) # x2
```

```
array([[50,  0,  0],
       [ 0, 47,  3],
       [ 0,  0, 50]], dtype=int64)
```

```
confusion_matrix(y, y_pred2) # PCscore2
```

```
array([[50,  0,  0],
       [ 0, 47,  3],
       [ 0,  2, 48]], dtype=int64)
```

□ [과제] 와인 차원축소

- 와인데이터의 전체 변수를 사용한 logistic regression 모델과 PCA로 차원을 축소한 모델의 성능 비교