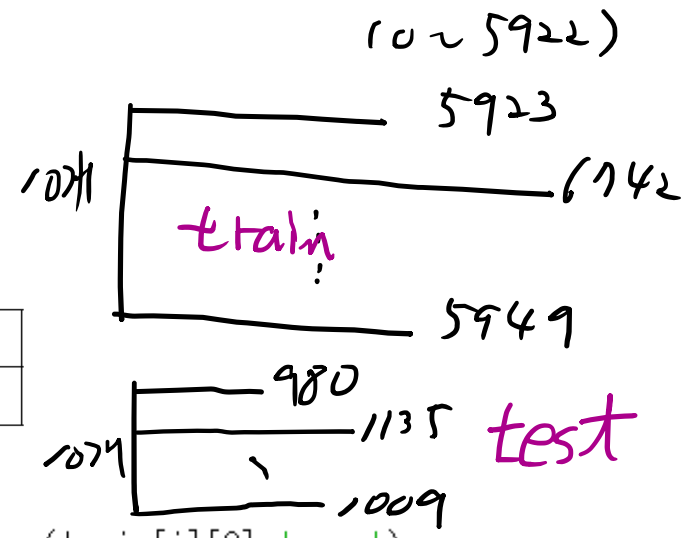
Dimensionality Reduction

MNIST Dataset

- LeCun 교수가 만들어 공개한 dataset
- 60,000 training data and 10,000 test data
- Handwritten digit images (size 28×28) with 0~1 values
- Each image can be flattened to 784 dimensional vector (1-D numpy array)

MNIST dataset

Train	5923	6742	5958	6131	5842	5421	5918	6265	5851	5949
test	980	1135	1032	1010	982	892	958	1028	974	1009




```python
import numpy as np
import matplotlib.pyplot as plt
import pickle
import pdb

def init_data():
    with open('train.bin','rb') as f1:
        train=pickle.load(f1)

    with open('test.bin','rb') as f2:
        test=pickle.load(f2)

    return train,test


train,test=init_data() # read data file:train.bin, test.bin

print(len(train))
print(len(train[0]))
print(train[0][0].shape)
print(len(test))
print(len(test[0]))
print(test[0][0].shape)
```

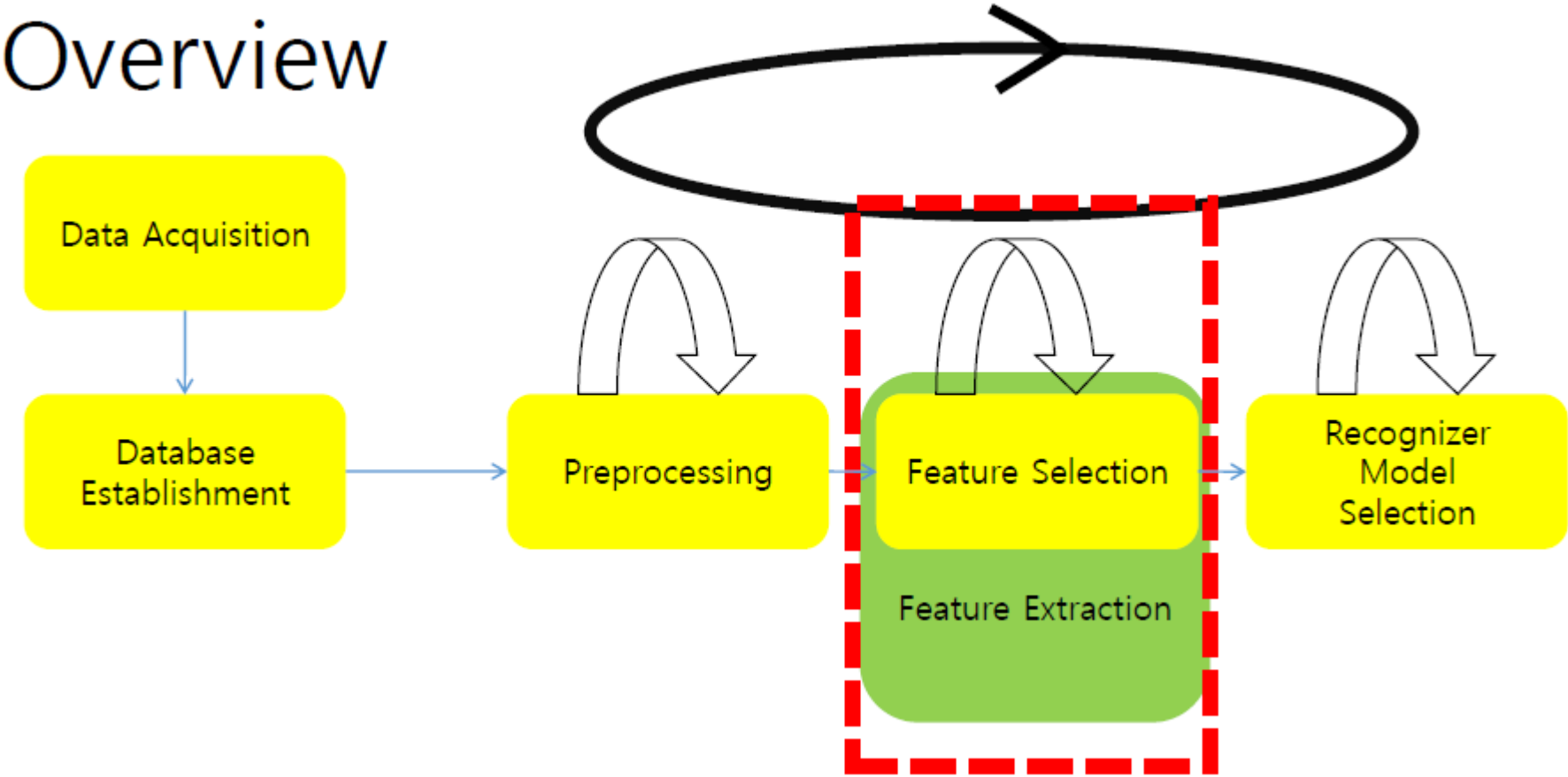
```
10
5923
(28, 28)
10
980
(28, 28)
```

```python
for i in range(10):
    plt.subplot(2,5,i+1),plt.imshow(train[i][0],'gray')
    plt.axis('off')
    print(len(train[i]))

plt.show()
```

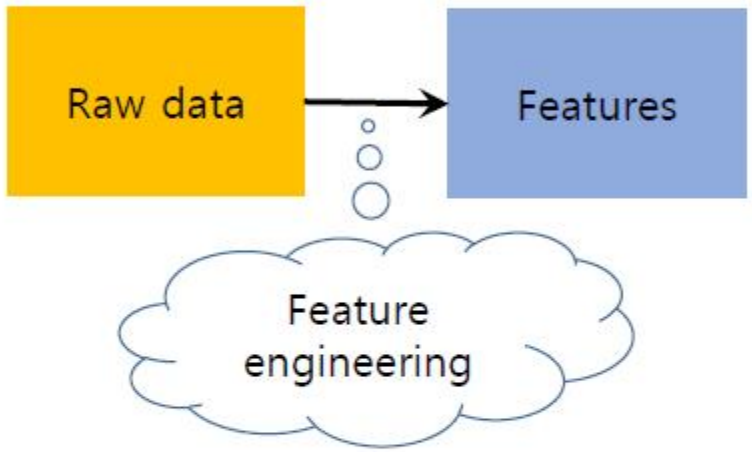
```
5923
6742
5958
6131
5842
5421
5918
6265
5851
5949
```

0MNIST.py

Overview

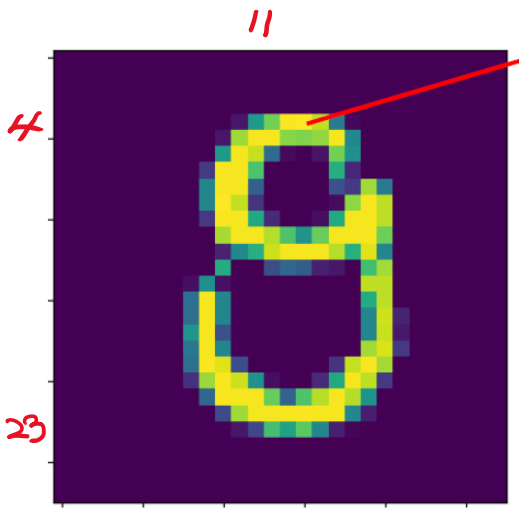

Data Acquisition
Database Establishment
Preprocessing
Feature Selection
Feature Extraction
Recognizer Model Selection

Features



인식기는 만능이 아니다!!!

Features for Number Recognition



(y, x)

```
(array([ 4,  4,  4,  4,  4,  4,  4,  4,  5,  5,  5,  5,  5,  5,  5,  5,  5,
         6,  6,  6,  6,  6,  6,  6,  6,  6,  7,  7,  7,  7,  7,  7,  8,  8,  8,
         8,  8,  8,  8,  8,  9,  9,  9,  9,  9,  9,  9,  9, 10, 10, 10, 10,
        10, 10, 10, 10, 10, 10, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11,
        12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 13, 13, 13, 13, 13, 13,
        13, 13, 14, 14, 14, 14, 14, 15, 15, 15, 15, 15, 16, 16, 16, 16, 16,
        16, 17, 17, 17, 17, 17, 17, 18, 18, 18, 18, 18, 18, 18, 19, 19, 19, 19,
        19, 19, 19, 19, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 21, 21,
        21, 21, 21, 21, 21, 21, 21, 21, 22, 22, 22, 22, 22, 22, 22, 22,
        22, 22, 23, 23, 23, 23, 23, 23, 23, 23], dtype=int64), array([11, 12, 13,
14, 15, 16, 17, 18, 10, 11, 12, 13, 14, 15, 16, 17, 18,
        10, 11, 12, 13, 14, 16, 17, 18,  9, 10, 11, 12, 17, 18,  9, 10, 11,
        12, 17, 18, 19, 20,  9, 10, 11, 12, 17, 18, 19, 20,  9, 10, 11, 12,
        13, 16, 17, 18, 19, 20, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20,
        10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 10, 13, 14, 15, 16, 17,
        19, 20,  8,  9, 10, 19, 20,  8,  9, 10, 19, 20,  8,  9, 10, 19, 20,
        21,  8,  9, 10, 19, 20, 21,  8,  9, 10, 19, 20, 21,  8,  9, 10, 11,
        17, 18, 19, 20,  8,  9, 10, 11, 12, 13, 16, 17, 18, 19, 20,  9, 10,
        11, 12, 13, 14, 15, 16, 17, 18, 19, 10, 11, 12, 13, 14, 15, 16, 17,
        18, 19, 11, 12, 13, 14, 15, 16, 17, 18], dtype=int64))
```

0이 아닌 위치의 좌표 값

```
def feat1(trainSet, testSet):

    trS1=len(trainSet); trS2=len(trainSet[0])
    teS1=len(testSet); teS2=len(testSet[0])

    trainSetf=np.zeros((trS1*trS2,5))
    testSetf=np.zeros((teS1*teS2,5))

    for i in range(trS1):
        for j in range(trS2):
            imsi=trainSet[i][j]
            imsi=np.where(imsi!=0)
            pdb.set_trace()
```

- mean(y) mean(x) var(y) var(xy) var(x)

functionsRev.py

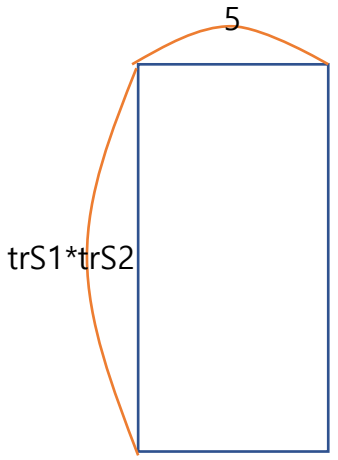
```
def feat1(trainSet, testSet):

    trS1=len(trainSet); trS2=len(trainSet[0])
    teS1=len(testSet); teS2=len(testSet[0])

    trainSetf=np.zeros((trS1*trS2,5))
    testSetf=np.zeros((teS1*teS2,5))

    for i in range(trS1):
        for j in range(trS2):
            imsi=trainSet[i][j]
            imsi=np.where(imsi!=0)
            imsi2=np.mean(imsi,1)
            imsi3=np.cov(imsi)
            trainSetf[i*trS2+j,:]=np.array([imsi2[0],imsi2[1], imsi3[0,0], imsi3[0,1], imsi3[1,1]])


    for i in range(teS1):
        for j in range(teS2):
            imsi=testSet[i][j]
            imsi=np.where(imsi!=0)
            imsi2=np.mean(imsi,1)
            imsi3=np.cov(imsi)
            testSetf[i*teS2+j,:]=np.array([imsi2[0],imsi2[1], imsi3[0,0], imsi3[0,1], imsi3[1,1]])

    return trainSetf, testSetf


def data_ready1(train,test,k=300):
    trainSet=[]
    testSet=[]
#    pdb.set_trace() #중단점 표시
    for i in range(10):
        trainSet.append(train[i][0:k])
        testSet.append(test[i][0:100])
    return trainSet,testSet
```

trS1=10, trS2=300
teS1=10, teS2=100

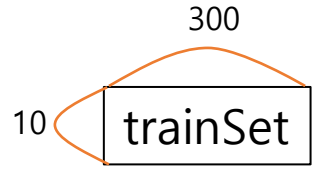
0이 아닌 위치의 좌표 값

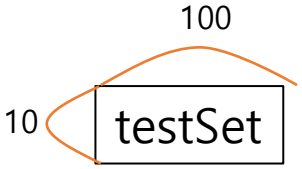
각 좌표 별 평균

mean(y) mean(x) var(y) var(xy) var(x)



trainSetf





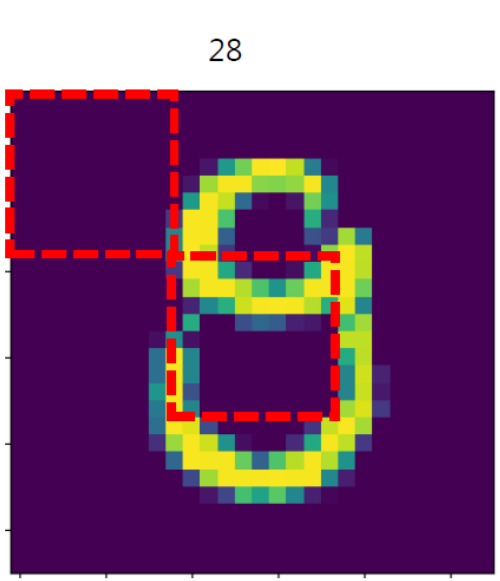
3featSel1.py

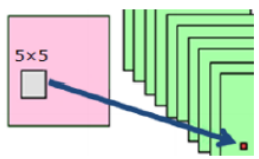
```
import numpy as np
import matplotlib.pyplot as plt
import pickle
import time
import functionsRev as fs
import pdb

t1=time.time()
#pdb.set_trace()
train, test=fs.init_data()
trainSet, testSet=fs.data_ready1(train,test)
trainSetf1, testSetf1=fs.feat1(trainSet, testSet)
```

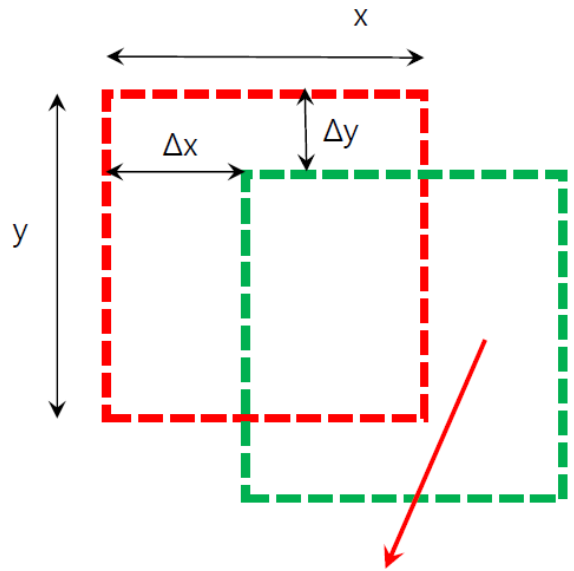
3featSel1.py

```
import numpy as np
import matplotlib.pyplot as plt
import pickle
import time
import functionsRev as fs
import pdb

t1=time.time()
#pdb.set_trace()
train, test=fs.init_data()
trainSet, testSet=fs.data_ready1(train,test)
trainSetf1, testSetf1=fs.feat1(trainSet, testSet)
###trainSetf2, testSetf2=fs.lda(trainSetf1, testSetf1, 15)

k=10
result=fs.knn(trainSetf1, testSetf1, k)
acc, pre, rec, f1=fs.calcMeasure(result)
t2=time.time()
print(t2-t1)
print('k=',k)
print('Acc=',acc.mean())
print('F1=',f1.mean())
```

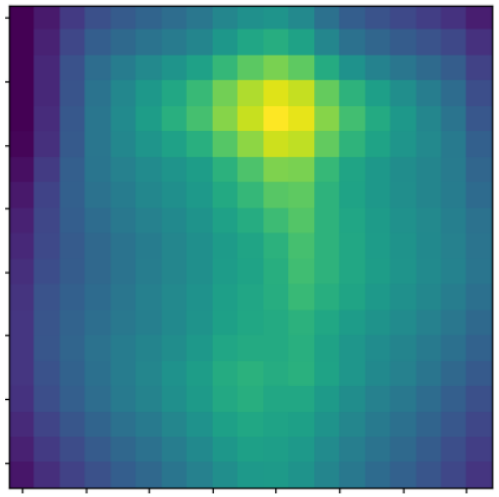
Features for Number Recognition







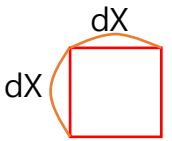
Sum()?
Mean()?



x=y=10, Δx=Δy=1

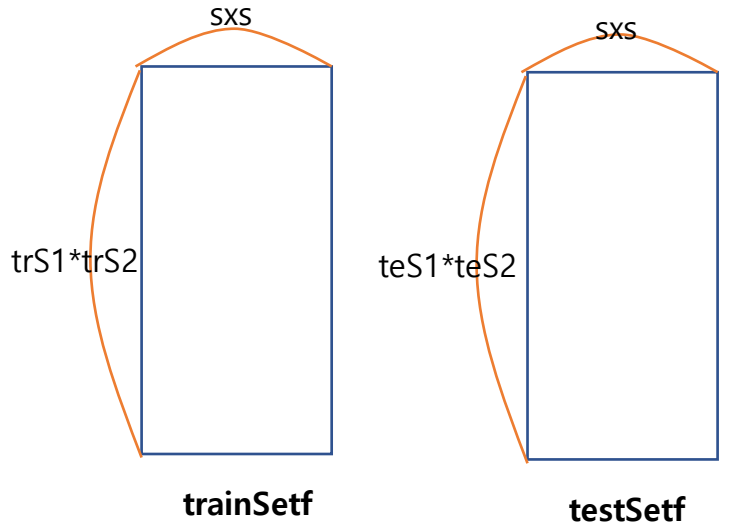
functionsRev.py

```
def feat2(trainSet, testSet, dX):
    size=trainSet[0][0].shape[0]; s=size-dX+1
    trS1=len(trainSet); trS2=len(trainSet[0])
    teS1=len(testSet); teS2=len(testSet[0])
    trainImsi=np.zeros((trS1*trS2,s,s)); testImsi=np.zeros((teS1*teS2,s,s))
    trainSetf=np.zeros((trS1*trS2,s*s)); testSetf=np.zeros((teS1*teS2,s*s))

    for i in range(trS1):
        for j in range(trS2):
            imsi=trainSet[i][j]
            for ii in range(s):
                for jj in range(s):
                    trainImsi[i*trS2+j,ii,jj]=imsi[ii:dX+ii,jj:dX+jj].sum()
            trainSetf[i*trS2+j,:]=trainImsi[i*trS2+j,::].flatten()

    for i in range(teS1):
        for j in range(teS2):
            imsi=testSet[i][j]
            for ii in range(s):
                for jj in range(s):
                    testImsi[i*teS2+j,ii,jj]=imsi[ii:dX+ii,jj:dX+jj].sum()
            testSetf[i*teS2+j,:]=testImsi[i*teS2+j,::].flatten()

    return trainSetf, testSetf
```



Size 내의 값을 더하기

(s,s) 2차원 데이터를 sxs 1차원 데이터로..

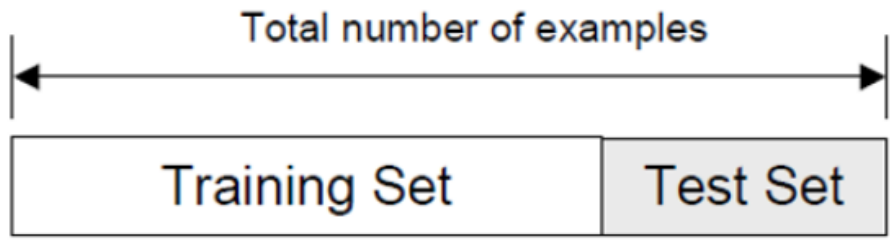


3featSel2.py

```
import numpy as np
import matplotlib.pyplot as plt
import pickle
import time
import functions as fs
import pdb

t1=time.time()
#pdb.set_trace()
train, test=fs.init_data()
trainSet, testSet=fs.data_ready1(train,test)

dX=20; k=10
trainSetf1, testSetf1=fs.feat2(trainSet, testSet, dX)
result=fs.knn(trainSetf1, testSetf1, k)
acc, pre, rec, f1=fs.calcMeasure(result)
t2=time.time()
print(t2-t1)
print('k=',k,' dX=',dX)
print('Acc=',acc.mean())
print('F1=',f1.mean())
```

Feature selection

- training에서 읽어들인 각 클래스별 300개의 데이터를 training set으로, test에서 읽어들인 각 클래스별 100개의 데이터를 test set으로 구분하라.



- 앞서 작성한 feat1과 knn을 사용하여 인식율을 확인해보자.
- 앞서 작성한 feat2과 knn을 사용하여 인식율을 확인해보자.

Feature selection

- k를 1, 5, 15, 50, 100, 200, ~~500~~으로 늘려가며, 아래 표를 완성시키라. Feat1 사용, 3featSel1.py 실행

F1	Total	c0	c1	c2	c3	c4	c5	c6	c7	c8	c9
기존											
K=1											
K=5											
K=15											
K=50											
K=100											
K=200											

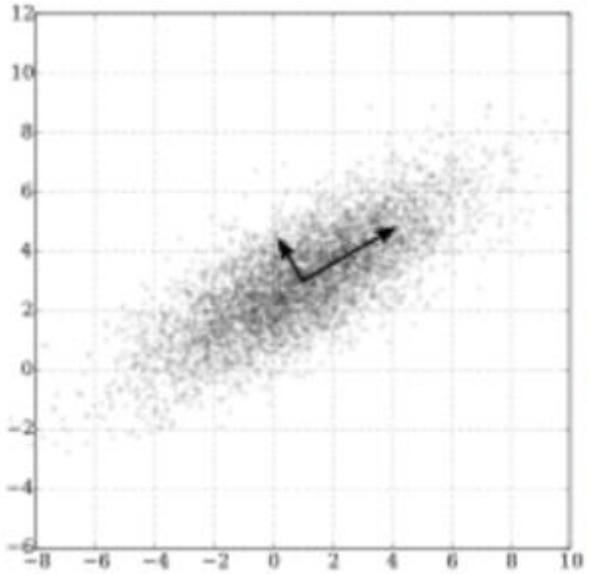
Feature selection

Feat2 사용, 3featSel2.py 실행

- k를 1, 5, 15, 50, 100, 200, 으로 dx를 3, 5, 10, 15, 20으로 늘려가며, 아래 표를 완성시키라.

F1	dX=3	dX=5	dX=10	dX=15	dX=20
K=1					
K=5					
K=15					
K=50					
K=100					
K=200					

PCA(Principal Component Analysis)



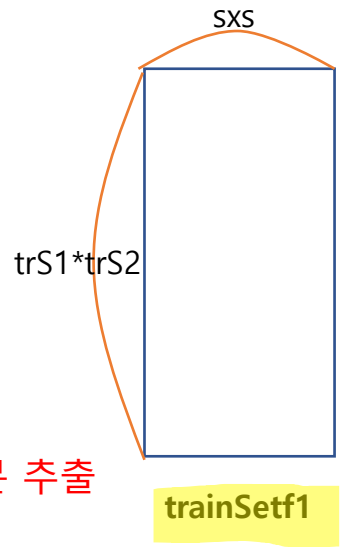
PCA of a multivariate Gaussian distribution centered at (1,3) with a standard deviation of 3 in roughly the (0.866, 0.5) direction and of 1 in the orthogonal direction. The vectors shown are the eigenvectors of the covariance matrix scaled by the square root of the corresponding eigenvalue, and shifted so their tails are at the mean.

functionsRev.py

```
def pca(trainSet, testSet, k):

    imsi=np.cov(trainSet.T)
    # L,V=np.linalg.eig(imsi)
    U,s,V=np.linalg.svd(imsi)
    #print(s)
    #plt.plot(s);plt.show()
    PC=U[:,np.argsort(s)[::-1]][:,:k]

    trainSetf=trainSet.dot(PC)
    testSetf=testSet.dot(PC)

    return trainSetf, testSetf
```

PCA 성분 추출

Eigenvalue(s) 크기의 역순으로 k개의 eigenvector를 선정

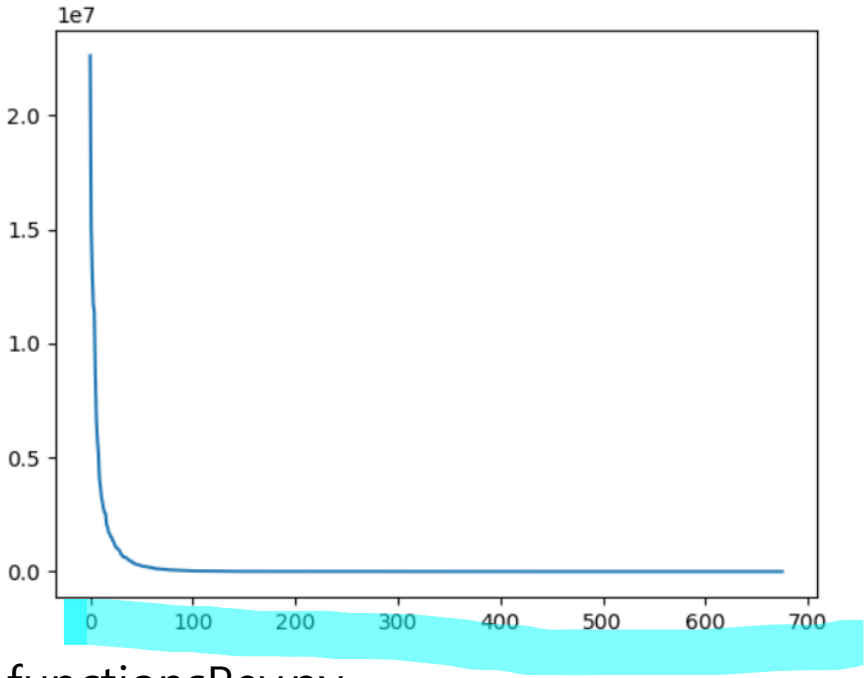


S=28-dX+1

4featSel3PCAF2.py

```
import numpy as np
import matplotlib.pyplot as plt
import pickle
import time
import functionsRev as fs
import pdb

#t1=time.time()
#pdb.set_trace()
train, test=fs.init_data()
trainSet, testSet=fs.data_ready1(train,test)

dx=3
trainSetf1, testSetf1=fs.feat2(trainSet, testSet, dx)
##t1=time.time()
##result=fs.knn(trainSetf1, testSetf1, k=10)
##t2=time.time()
##print(t2-t1)
##acc, pre, rec, f1=fs.calcMeasure(result)
##print('Acc=',acc.mean())
##print('F1=',f1.mean())

trainSetf2, testSetf2=fs.pca(trainSetf1, testSetf1,25)
t1=time.time()
result=fs.knn(trainSetf2, testSetf2, k=10)
t2=time.time()
print(t2-t1)
acc, pre, rec, f1=fs.calcMeasure(result)
print('PCA_Acc=',acc.mean())
print('PCA_F1=',f1.mean())
```

PCA(Principal Component Analysis)

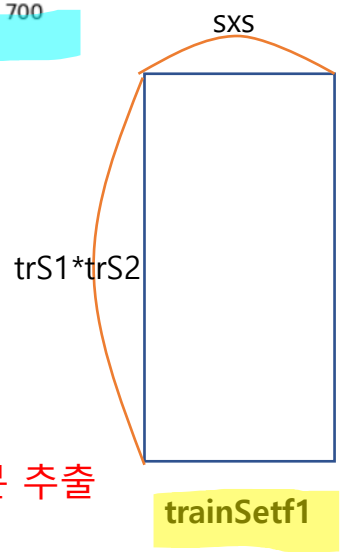


functionsRev.py

```
def pca(trainSet, testSet, k):

    imsi=np.cov(trainSet.T)
    # L,V=np.linalg.eig(imsi)
    U,s,V=np.linalg.svd(imsi)
    #print(s)
    #plt.plot(s);plt.show()
    PC=U[:,np.argsort(s)[::-1]][:,:k]

    trainSetf=trainSet.dot(PC)
    testSetf=testSet.dot(PC)

    return trainSetf, testSetf
```

PCA 성분 추출

Eigenvalue(s) 크기의 역순으로 k개의 eigenvector를 선정



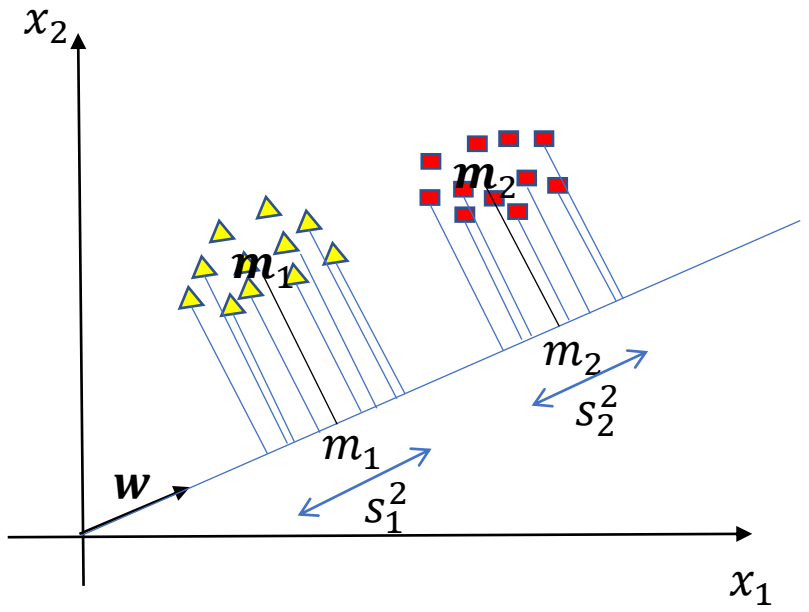
S=28-dX+1

4featSel3PCAF2.py

```
import numpy as np
import matplotlib.pyplot as plt
import pickle
import time
import functionsRev as fs
import pdb

#t1=time.time()
#pdb.set_trace()
train, test=fs.init_data()
trainSet, testSet=fs.data_ready1(train,test)

dx=3
trainSetf1, testSetf1=fs.feat2(trainSet, testSet, dx)
##t1=time.time()
##result=fs.knn(trainSetf1, testSetf1, k=10)
##t2=time.time()
##print(t2-t1)
##acc, pre, rec, f1=fs.calcMeasure(result)
##print('Acc=',acc.mean())
##print('F1=',f1.mean())

trainSetf2, testSetf2=fs.pca(trainSetf1, testSetf1,25)
t1=time.time()
result=fs.knn(trainSetf2, testSetf2, k=10)
t2=time.time()
print(t2-t1)
acc, pre, rec, f1=fs.calcMeasure(result)
print('PCA_Acc=',acc.mean())
print('PCA_F1=',f1.mean())
```

LDA(Linear Discriminant Analysis)

- Fisher's idea: Fisher's linear discriminant
  - Maximize a function that will give a large separation between the projected class means
  - While also giving a small variance within each class, thereby minimizing the class overlap



functionsRev.py

```
def lda(trainSet, testSet, k):
    trS1=len(trainSet)
    trS2=len(trainSet[0])
    trS3=int(trS1/10)

    covMat=np.zeros((10,trS2,trS2))
    meanV=np.zeros((10,trS2))

    for i in range(10):
        covMat[i,::]=np.cov(trainSet[i*trS3:(i+1)*trS3,:].T)
        meanV[i,:]=np.mean(trainSet[i*trS3:(i+1)*trS3,:],0)

    meanC=np.mean(meanV,0)
    Sb=(meanV-meanC).T.dot(meanV-meanC)
    Sw=covMat.sum(0)
    imsi=np.linalg.pinv(Sw).dot(Sb)
    # L,V=np.linalg.eig(imsi)
    U,s,V=np.linalg.svd(imsi)
    PC=U[:,np.argsort(s)[::-1]][:,:k]
    trainSetf=trainSet.dot(PC)
    testSetf=testSet.dot(PC)

    return trainSetf, testSetf
```

$\mathbf{S_W^{-1}S_B}$

Within-class scatter matrix for $\mathcal{C}_i$ is

$$\mathbf{S_i} = \sum_{\mathbf{t}} \mathbf{r_i^t}(\mathbf{x^t} - \mathbf{m_i})(\mathbf{x^t} - \mathbf{m_i})^{\mathbf{T}}$$

where $r_i^t = 1$ if $x^t \in C_i$ and 0 otherwise.

The total within-class scatter is

$$\mathbf{S_W} = \sum_{\mathbf{i=1}}^{\mathbf{K}} \mathbf{S_i}$$

The between-class scatter matrix is

$$\mathbf{S_B} = \sum_{\mathbf{i=1}}^{\mathbf{K}} \mathbf{N_i}(\mathbf{m_i} - \mathbf{m})(\mathbf{m_i} - \mathbf{m})^{\mathbf{T}}$$

where $N_i = \sum_t r_i^t$

* The largest eigenvectors of $\mathbf{S_W^{-1}S_B}$ are the solution

4featSel3PCAF2.py

```
import numpy as np
import matplotlib.pyplot as plt
import pickle
import time
import functionsRev as fs
import pdb

#t1=time.time()
#pdb.set_trace()
train, test=fs.init_data()
trainSet, testSet=fs.data_ready1(train,test)

dx=3
trainSetf1, testSetf1=fs.feat2(trainSet, testSet, dx)
##t1=time.time()
##result=fs.knn(trainSetf1, testSetf1, k=10)
##t2=time.time()
##print(t2-t1)
##acc, pre, rec, f1=fs.calcMeasure(result)
##print('Acc=',acc.mean())
##print('F1=',f1.mean())

trainSetf2, testSetf2=fs.lda(trainSetf1, testSetf1,7)
t1=time.time()
result=fs.knn(trainSetf2, testSetf2, k=10)
t2=time.time()
print(t2-t1)
acc, pre, rec, f1=fs.calcMeasure(result)
print('PCA_Acc=',acc.mean())
print('PCA_F1=',f1.mean())
```

4주MNIST/Homework2.xlsx

Homework#2

Template Matching				k-NN(300/digit)				k-NN(300/digit)-feature selection1			
#template	F1	Accuracy			F1	Accuracy			F1	Accuracy	
#10				k=1				k=1			
#50				k=5				k=5			
#100				k=15				k=15			
#200				k=50				k=50			
#300				k=100				k=100			
#500				k=200				k=200			
#1000											

k-NN(300/digit)-feature selection2							k-NN(300/digit)-feature selection2					
F1	dx=3	dx=5	dx=10	dx=15	dx=20		Acc	dx=3	dx=5	dx=10	dx=15	dx=20
k=1							k=1					
k=5							k=5					
k=15							k=15					
k=50							k=50					
k=100							k=100					
k=200							k=200					

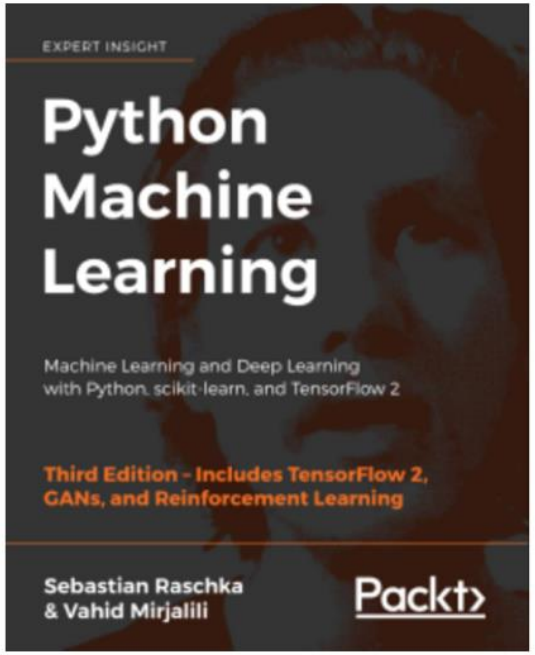
F1	k-NN	FeatSel1	FeatSel2	FeatSel2	FeatSel2
			dx=?	PCA 45	LDA 7
k=1					
k=5					
k=15					
k=50					
k=100					
k=200					

Accuracy	k-NN	FeatSel1	FeatSel2	FeatSel2	FeatSel2
			dx=?	PCA 45	LDA 7
k=1					
k=5					
k=15					
k=50					
k=100					
k=200					

Python Machine Learning Third Edition
Machine Learning and Deep Learning with Python, scikit-learn and Tensorflow 2

CHAPTER 5 Compressing Data via Dimensionality Reduction
Feature selection for dimensionality reduction: Feature Extraction (PCA, LDA)

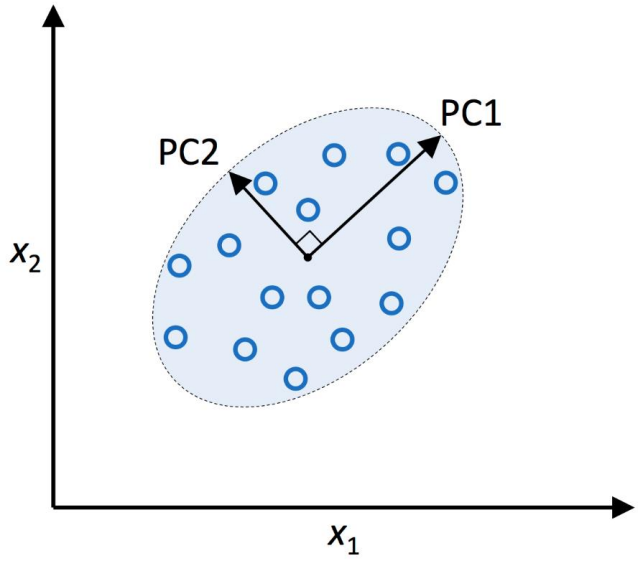
unsupervised dimensionality reduction via principal component analysis
The main steps behind PCA
Extracting the principal components step by step
Total and explained variance
Feature transformation
Principal component analysis in scikit-learn
Supervised data compression via linear discriminant analysis
The inner workings of LDA
Computing the scatter matrices
Selecting linera discriminants for the new feature subspace
Projecting samples onto the new feature space
LDA via scikit-learn



Unsupervised dimensionality reduction via principal component analysis

- Feature Extraction
  - ✓ Transform or project the data onto a new feature space
  - ✓ An approach to data compression with the goal of maintaining most of the relevant information
  - ✓ Improve the predictive performance by reducing the curse of dimensionality

The main steps behind principal component analysis(PCA)



1. Standardize the d-dimensional dataset
2. Construct the covariance matrix
3. Decompose the covariance matrix into eigenvectors and eigenvalues
4. Sort the eigenvalues by decreasing order to rank the corresponding eigenvectors
5. Select k eigenvectors which corresponds to the k largest eigenvalues
6. Construct a projection matrix W from the top k eigenvectors
7. Transform the d-dimensional dataset X using the projection matrix W

Extracting the principal components step by step

1. Standardize the d-dimensional dataset
2. Construct the covariance matrix
3. Decompose the covariance matrix into eigenvectors and eigenvalues

ch05tmp1.py

```
import pandas as pd

df_wine = pd.read_csv('https://archive.ics.uci.edu/ml/'
                      'machine-learning-databases/wine/wine.data',
                      header=None)

df_wine.columns = ['Class label', 'Alcohol', 'Malic acid', 'Ash',
                   'Alcalinity of ash', 'Magnesium', 'Total phenols',
                   'Flavanoids', 'Nonflavanoid phenols', 'Proanthocyanins',
                   'Color intensity', 'Hue',
                   'OD280/OD315 of diluted wines', 'Proline']

df_wine.head()

from sklearn.model_selection import train_test_split

X, y = df_wine.iloc[:, 1:].values, df_wine.iloc[:, 0].values

X_train, X_test, y_train, y_test =     train_test_split(X, y, test_size=0.3,
                     stratify=y,
                     random_state=0)

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train_std = sc.fit_transform(X_train)
X_test_std = sc.transform(X_test)

import numpy as np
cov_mat = np.cov(X_train_std.T)
eigen_vals, eigen_vecs = np.linalg.eig(cov_mat)

print('₩nEigenvalues ₩n%s' % eigen_vals)
```

1

```
Eigenvalues
[4.84274532 2.41602459 1.54845825 0.96120438 0.84166161 0.6620634
 0.51828472 0.34650377 0.3131368  0.10754642 0.21357215 0.15362835
 0.1808613 ]
```

Total and explained variance

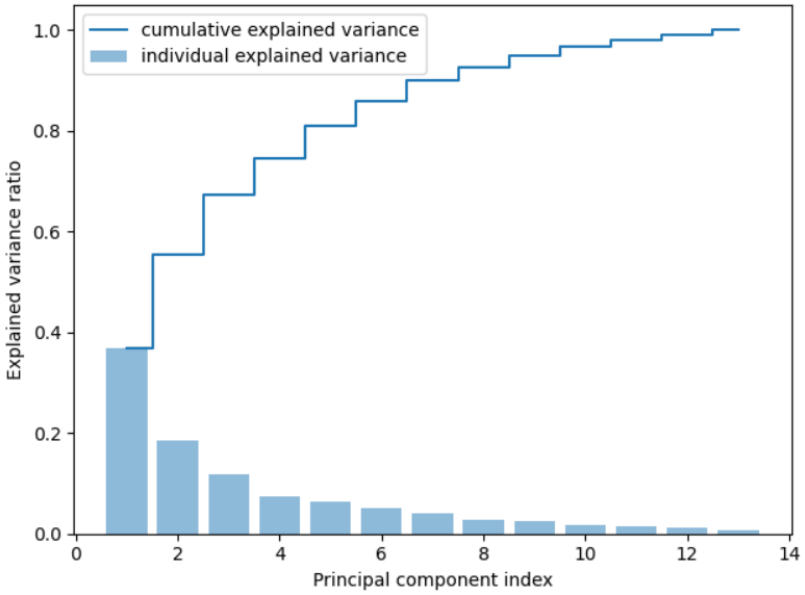
4. Sort the eigenvalues by decreasing order to rank the corresponding eigenvectors

ch05tmp1.py

```
tot = sum(eigen_vals)
var_exp = [(i / tot) for i in sorted(eigen_vals, reverse=True)]
cum_var_exp = np.cumsum(var_exp)

import matplotlib.pyplot as plt


plt.bar(range(1, 14), var_exp, alpha=0.5, align='center',
        label='individual explained variance')
plt.step(range(1, 14), cum_var_exp, where='mid',
         label='cumulative explained variance')
plt.ylabel('Explained variance ratio')
plt.xlabel('Principal component index')
plt.legend(loc='best')
plt.tight_layout()
# plt.savefig('images/05_02.png', dpi=300)
plt.show()
```

Variance explained ratios of eigenvalues

$$\frac{\lambda_j}{\sum_{j=1}^{d}\lambda_j} \quad \text{where } \lambda_j : \text{j-th eigenvalue}$$



Feature transformation

5. Select k eigenvectors which corresponds to the k largest eigenvalues
6. Construct a projection matrix W from the top k eigenvectors
7. Transform the d-dimensional dataset X using the projection matrix W

ch05tmp1.py

```
eigen_pairs = [(np.abs(eigen_vals[i]), eigen_vecs[:, i])
               for i in range(len(eigen_vals))]

# Sort the (eigenvalue, eigenvector) tuples from high to low
eigen_pairs.sort(key=lambda k: k[0], reverse=True)

w = np.hstack((eigen_pairs[0][1][:, np.newaxis],
               eigen_pairs[1][1][:, np.newaxis]))
print('Matrix W:\n', w)

X_train_std[0].dot(w)

X_train_pca = X_train_std.dot(w)
colors = ['r', 'b', 'g']
markers = ['s', 'x', 'o']

for l, c, m in zip(np.unique(y_train), colors, markers):
    plt.scatter(X_train_pca[y_train == l, 0],
                X_train_pca[y_train == l, 1],
                c=c, label=l, marker=m)

plt.xlabel('PC 1')
plt.ylabel('PC 2')
plt.legend(loc='lower left')
plt.tight_layout()
# plt.savefig('images/05_03.png', dpi=300)
plt.show()
```

Make a list of (eigenvalue, eigenvector) tuples

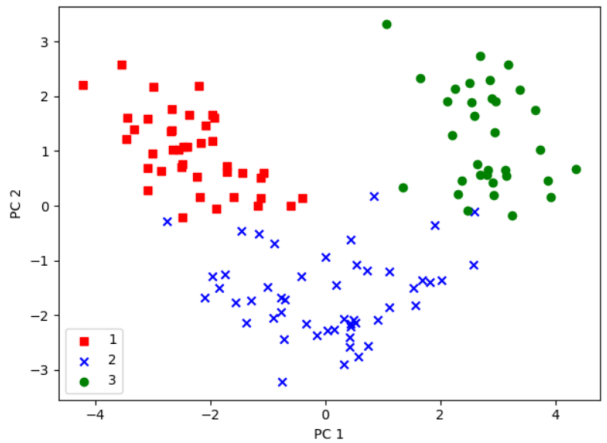
W:13x2

x′=xW
x:1x13

eigenvalue index

X′=XW

```
Matrix W:
 [[-0.13724218  0.50303478]
 [ 0.24724326  0.16487119]
 [-0.02545159  0.24456476]
 [ 0.20694508 -0.11352904]
 [-0.15436582  0.28974518]
 [-0.39376952  0.05080104]
 [-0.41735106 -0.02287338]
 [ 0.30572896  0.09048885]
 [-0.30668347  0.00835233]
 [ 0.07554066  0.54977581]
 [-0.32613263 -0.20716433]
 [-0.36861022 -0.24902536]
 [-0.29669651  0.38022942]]
```



Principal component analysis in scikit-learn

- How to use the PCA class implemented in scikit-learn

ch05tmp2.py

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

df_wine = pd.read_csv('https://archive.ics.uci.edu/ml/'
                      'machine-learning-databases/wine/wine.data',
                      header=None)

df_wine.columns = ['Class label', 'Alcohol', 'Malic acid', 'Ash',
                   'Alcalinity of ash', 'Magnesium', 'Total phenols',
                   'Flavanoids', 'Nonflavanoid phenols', 'Proanthocyanins',
                   'Color intensity', 'Hue',
                   'OD280/OD315 of diluted wines', 'Proline']

df_wine.head()

from sklearn.model_selection import train_test_split

X, y = df_wine.iloc[:, 1:].values, df_wine.iloc[:, 0].values

X_train, X_test, y_train, y_test =     train_test_split(X, y, test_size=0.3,
                     stratify=y,
                     random_state=0)

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train_std = sc.fit_transform(X_train)
X_test_std = sc.transform(X_test)
```

Principal component analysis in scikit-learn

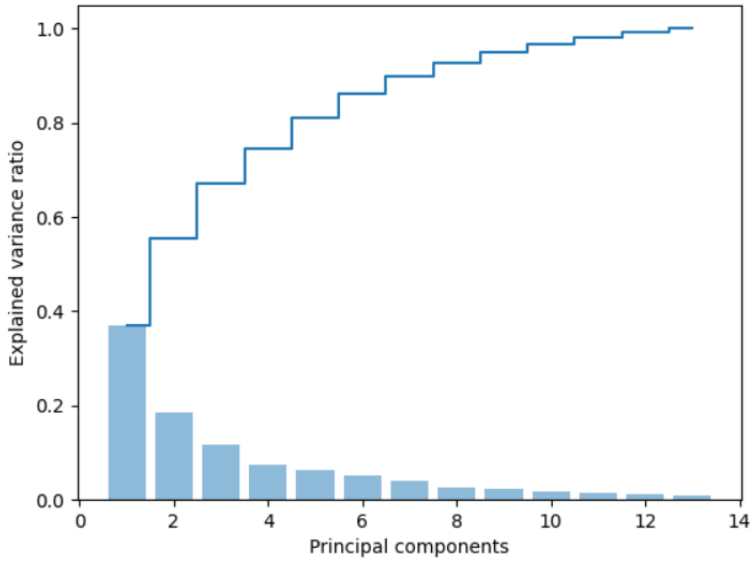
ch05tmp2.py

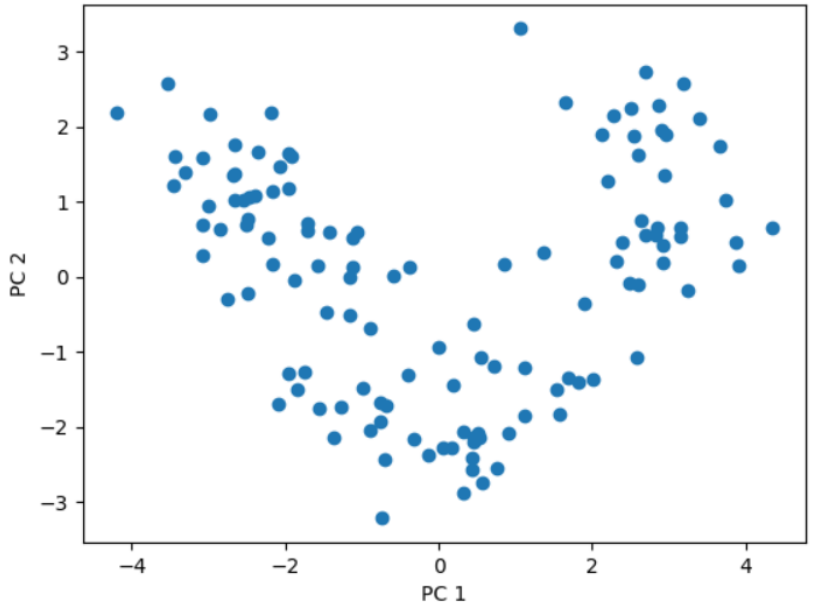
```
from matplotlib.colors import ListedColormap

def plot_decision_regions(X, y, classifier, resolution=0.02):

    # setup marker generator and color map
    markers = ('s', 'x', 'o', '^', 'v')
    colors = ('red', 'blue', 'lightgreen', 'gray', 'cyan')
    cmap = ListedColormap(colors[:len(np.unique(y))])

    # plot the decision surface
    x1_min, x1_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    x2_min, x2_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx1, xx2 = np.meshgrid(np.arange(x1_min, x1_max, resolution),
                           np.arange(x2_min, x2_max, resolution))
    Z = classifier.predict(np.array([xx1.ravel(), xx2.ravel()]).T)
    Z = Z.reshape(xx1.shape)
    plt.contourf(xx1, xx2, Z, alpha=0.4, cmap=cmap)
    plt.xlim(xx1.min(), xx1.max())
    plt.ylim(xx2.min(), xx2.max())

    # plot class samples
    for idx, cl in enumerate(np.unique(y)):
        plt.scatter(x=X[y == cl, 0],
                    y=X[y == cl, 1],
                    alpha=0.6,
                    c=cmap(idx),
                    edgecolor='black',
                    marker=markers[idx],
                    label=cl)
```

Principal component analysis in scikit-learn

ch05tmp2.py

```
from sklearn.decomposition import PCA

pca = PCA()
X_train_pca = pca.fit_transform(X_train_std)
pca.explained_variance_ratio_


# In[15]:


plt.bar(range(1, 14), pca.explained_variance_ratio_, alpha=0.5, align='center')
plt.step(range(1, 14), np.cumsum(pca.explained_variance_ratio_), where='mid')
plt.ylabel('Explained variance ratio')
plt.xlabel('Principal components')

plt.show()

pca = PCA(n_components=2)
X_train_pca = pca.fit_transform(X_train_std)
X_test_pca = pca.transform(X_test_std)


# In[17]:


plt.scatter(X_train_pca[:, 0], X_train_pca[:, 1])
plt.xlabel('PC 1')
plt.ylabel('PC 2')
plt.show()

from sklearn.linear_model import LogisticRegression

pca = PCA(n_components=2)
X_train_pca = pca.fit_transform(X_train_std)
X_test_pca = pca.transform(X_test_std)

lr = LogisticRegression()
lr = lr.fit(X_train_pca, y_train)
```

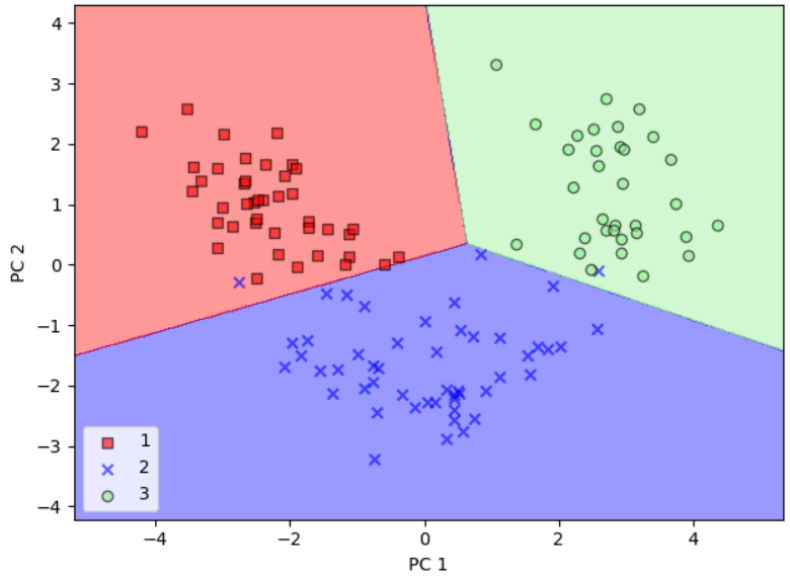


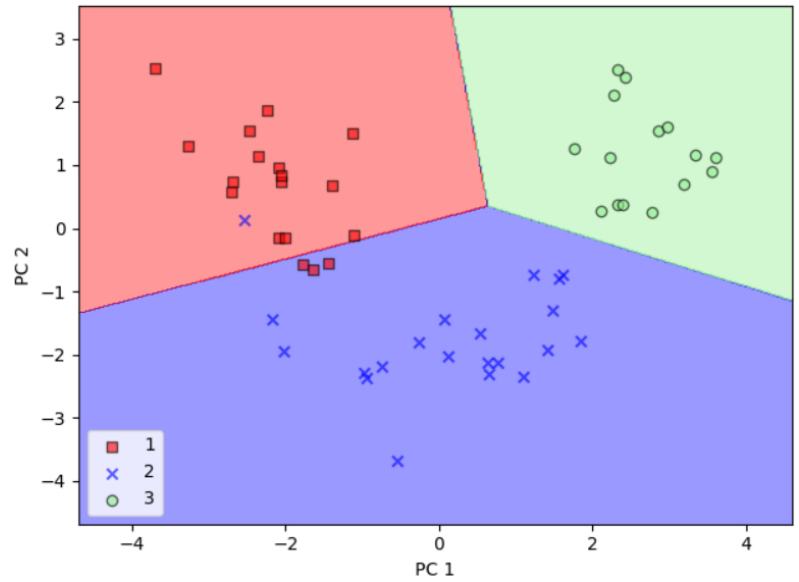


Principal component analysis in scikit-learn

ch05tmp2.py

```
# In[20]:


plot_decision_regions(X_train_pca, y_train, classifier=lr)
plt.xlabel('PC 1')
plt.ylabel('PC 2')
plt.legend(loc='lower left')
plt.tight_layout()
# plt.savefig('images/05_04.png', dpi=300)
plt.show()

# In[21]:


plot_decision_regions(X_test_pca, y_test, classifier=lr)
plt.xlabel('PC 1')
plt.ylabel('PC 2')
plt.legend(loc='lower left')
plt.tight_layout()
# plt.savefig('images/05_05.png', dpi=300)
plt.show()

# In[22]:


pca = PCA(n_components=None)
X_train_pca = pca.fit_transform(X_train_std)
pca.explained_variance_ratio_
```

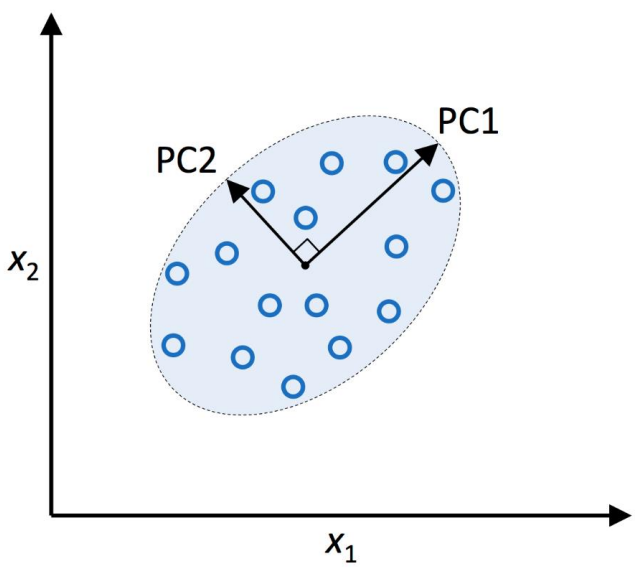


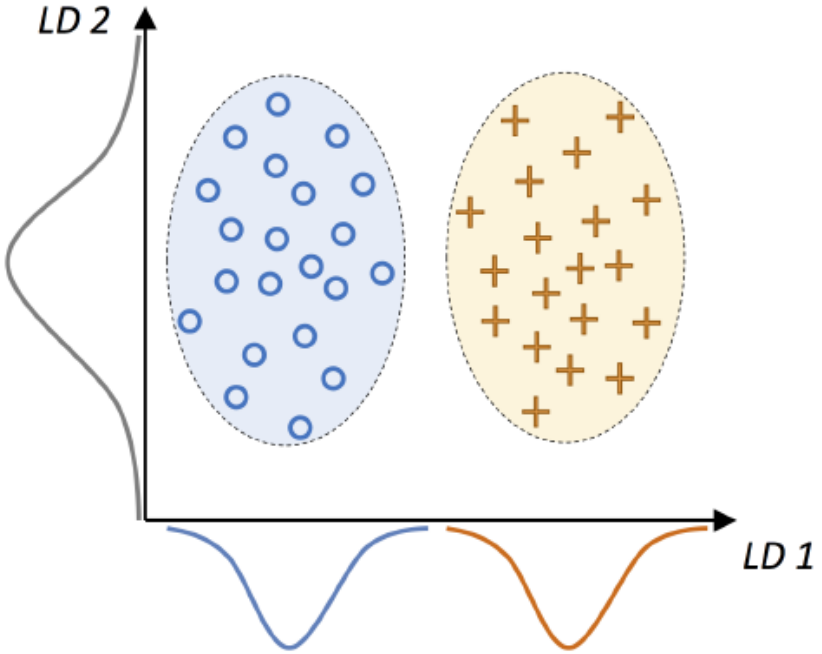


Supervised data compression via linear discriminant analysis

- Linear Discriminant Analysis
  - ✓ Find the feature subspace that optimizes class separability

PCA versus LDA





The inner workings of linear discriminant analysis(LDA)

1. Standardize the d-dimensional dataset
2. For each class, compute the d-dimensional mean vector
3. Construct the between-scatter matrix S_B and the within-class scatter matrix S_W
4. Compute the eigenvectors and corresponding eigenvalues of the matrix $S_W^{-1}\ S_B$
5. Sort the eigenvalues by decreasing order to rank the corresponding eigenvectors
6. Choose the k eigenvectors which corresponds to the k largest eigenvalues to construct a *dxk* dimensional transformation matrix W
7. Project the samples onto te new feature subspace using the projection matrix W

Computing the scatter matrices

ch05tmp3.py

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

df_wine = pd.read_csv('https://archive.ics.uci.edu/ml/'
                      'machine-learning-databases/wine/wine.data',
                      header=None)

df_wine.columns = ['Class label', 'Alcohol', 'Malic acid', 'Ash',
                   'Alcalinity of ash', 'Magnesium', 'Total phenols',
                   'Flavanoids', 'Nonflavanoid phenols', 'Proanthocyanins',
                   'Color intensity', 'Hue',
                   'OD280/OD315 of diluted wines', 'Proline']

df_wine.head()
```

```
from sklearn.model_selection import train_test_split

X, y = df_wine.iloc[:, 1:].values, df_wine.iloc[:, 0].values

X_train, X_test, y_train, y_test =     train_test_split(X, y, test_size=0.3,
                     stratify=y,
                     random_state=0)

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train_std = sc.fit_transform(X_train)
X_test_std = sc.transform(X_test)
```

Computing the scatter matrices

ch05tmp3.py

```
np.set_printoptions(precision=4)

mean_vecs = []
for label in range(1, 4):
    mean_vecs.append(np.mean(X_train_std[y_train == label], axis=0))
    print('MV %s: %s\n' % (label, mean_vecs[label - 1]))

# Compute the within-class scatter matrix:

# In[25]:


d = 13 # number of features
S_W = np.zeros((d, d))
for label, mv in zip(range(1, 4), mean_vecs):
    class_scatter = np.zeros((d, d))  # scatter matrix for each class
    for row in X_train_std[y_train == label]:
        row, mv = row.reshape(d, 1), mv.reshape(d, 1)  # make column vectors
        class_scatter += (row - mv).dot((row - mv).T)
    S_W += class_scatter                             # sum class scatter matrices

print('Within-class scatter matrix: %sx%s' % (S_W.shape[0], S_W.shape[1]))

# Better: covariance matrix since classes are not equally distributed:

# In[26]:


print('Class label distribution: %s'
      % np.bincount(y_train)[1:])
```

$$\boldsymbol{m}_i = \frac{1}{n_i} \sum_{\boldsymbol{x} \in D_i}^{c} \boldsymbol{x}$$

$$\boldsymbol{S}_i = \sum_{\boldsymbol{x} \in D_i}^{c} (\boldsymbol{x} - \boldsymbol{m}_i)(\boldsymbol{x} - \boldsymbol{m}_i)^T$$

$$\boldsymbol{S}_W = \sum_{i=1}^{c} \boldsymbol{S}_i$$

$$\sum_i = \frac{1}{n_i} \boldsymbol{S}_W = \frac{1}{n_i} \sum_{\boldsymbol{x} \in D_i}^{c} (\boldsymbol{x} - \boldsymbol{m}_i)(\boldsymbol{x} - \boldsymbol{m}_i)^T$$

Computing the scatter matrices

ch05tmp3.py

```
# In[27]:


d = 13  # number of features
S_W = np.zeros((d, d))
for label, mv in zip(range(1, 4), mean_vecs):
    class_scatter = np.cov(X_train_std[y_train == label].T)
    S_W += class_scatter
print('Scaled within-class scatter matrix: %sx%s' % (S_W.shape[0],
                                                     S_W.shape[1]))


# Compute the between-class scatter matrix:

# In[28]:


mean_overall = np.mean(X_train_std, axis=0)
d = 13  # number of features
S_B = np.zeros((d, d))
for i, mean_vec in enumerate(mean_vecs):
    n = X_train[y_train == i + 1, :].shape[0]
    mean_vec = mean_vec.reshape(d, 1)  # make column vector
    mean_overall = mean_overall.reshape(d, 1)  # make column vector
    S_B += n * (mean_vec - mean_overall).dot((mean_vec - mean_overall).T)

print('Between-class scatter matrix: %sx%s' % (S_B.shape[0], S_B.shape[1]))
```

$$\sum_i = \frac{1}{n_i} \boldsymbol{S}_W = \frac{1}{n_i} \sum_{\boldsymbol{x} \in D_i}^{c} (\boldsymbol{x} - \boldsymbol{m}_i)(\boldsymbol{x} - \boldsymbol{m}_i)^T$$

$$\boldsymbol{S}_B = \sum_{i=1}^{c} n_i (\boldsymbol{m}_i - \boldsymbol{m})(\boldsymbol{m}_i - \boldsymbol{m})^T$$

Selecting linear discriminants for the new feature subspace

4. Compute the eigenvectors and corresponding eigenvalues of the matrix $S_W^{-1} S_B$
5. Sort the eigenvalues by decreasing order to rank the corresponding eigenvectors
6. Choose the k eigenvectors which corresponds to the k largest eigenvalues to construct a *dxk* dimensional transformation matrix W
7. Project the samples onto te new feature subspace using the projection matrix W

ch05tmp3.py

```
# Solve the generalized eigenvalue problem for the matrix $S_W^{-1}S_B$:

# In[29]:


eigen_vals, eigen_vecs = np.linalg.eig(np.linalg.inv(S_W).dot(S_B))


# Make a list of (eigenvalue, eigenvector) tuples
eigen_pairs = [(np.abs(eigen_vals[i]), eigen_vecs[:, i])
               for i in range(len(eigen_vals))]

# Sort the (eigenvalue, eigenvector) tuples from high to low
eigen_pairs = sorted(eigen_pairs, key=lambda k: k[0], reverse=True)

# Visually confirm that the list is correctly sorted by decreasing eigenvalues

print('Eigenvalues in descending order:₩n')
for eigen_val in eigen_pairs:
    print(eigen_val[0])
```

```
Eigenvalues in descending order:

349.61780890599414
172.76152218979394
4.083480689889729e-14
2.842170943040401e-14
2.6148565436328254e-14
2.6148565436328254e-14
1.802389952469644e-14
1.802389952469644e-14
9.732182461883806e-15
9.732182461883806e-15
3.2709312372580184e-15
3.259777620466185e-15
3.259777620466185e-15
```

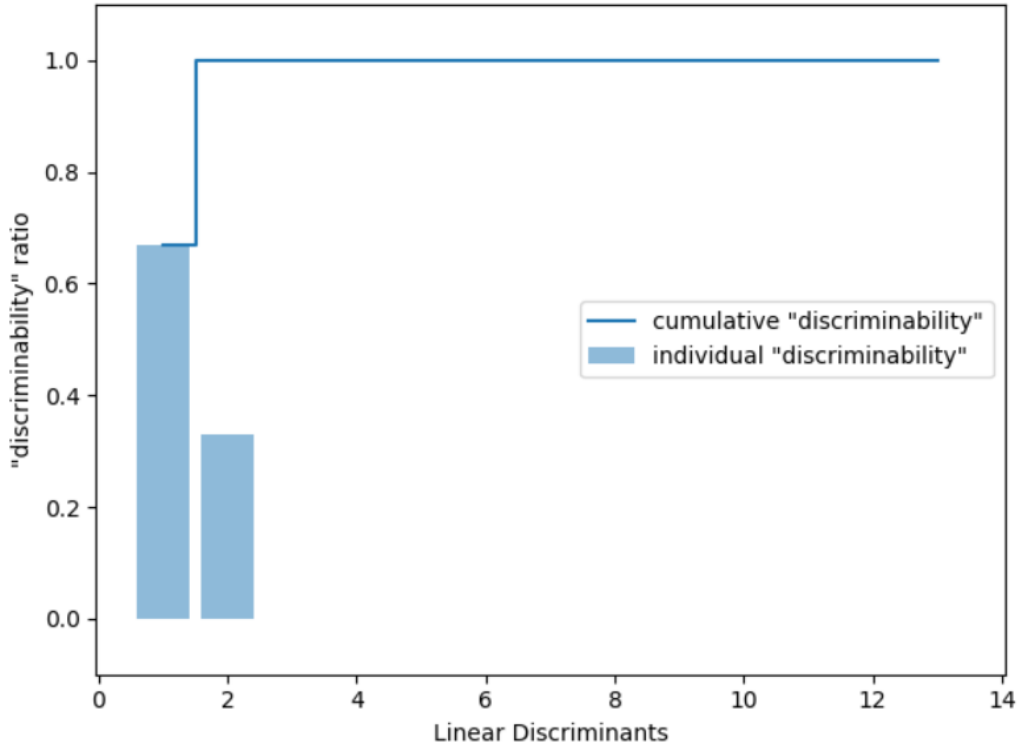
Selecting linear discriminants for the new feature subspace

6. Choose the k eigenvectors which corresponds to the k largest eigenvalues to construct a *dxk* dimensional transformation matrix W

ch05tmp3.py

```
tot = sum(eigen_vals.real)
discr = [(i / tot) for i in sorted(eigen_vals.real, reverse=True)]
cum_discr = np.cumsum(discr)

plt.bar(range(1, 14), discr, alpha=0.5, align='center',
        label='individual "discriminability"')
plt.step(range(1, 14), cum_discr, where='mid',
         label='cumulative "discriminability"')
plt.ylabel('"discriminability" ratio')
plt.xlabel('Linear Discriminants')
plt.ylim([-0.1, 1.1])
plt.legend(loc='best')
plt.tight_layout()
# plt.savefig('images/05_07.png', dpi=300)
plt.show()


# In[32]:


w = np.hstack((eigen_pairs[0][1][:, np.newaxis].real,
               eigen_pairs[1][1][:, np.newaxis].real))
print('Matrix W:₩n', w)
```

```
Matrix W:
 [[-0.1481 -0.4092]
 [ 0.0908 -0.1577]
 [-0.0168 -0.3537]
 [ 0.1484  0.3223]
 [-0.0163 -0.0817]
 [ 0.1913  0.0842]
 [-0.7338  0.2823]
 [-0.075  -0.0102]
 [ 0.0018  0.0907]
 [ 0.294  -0.2152]
 [-0.0328  0.2747]
 [-0.3547 -0.0124]
 [-0.3915 -0.5958]]
```

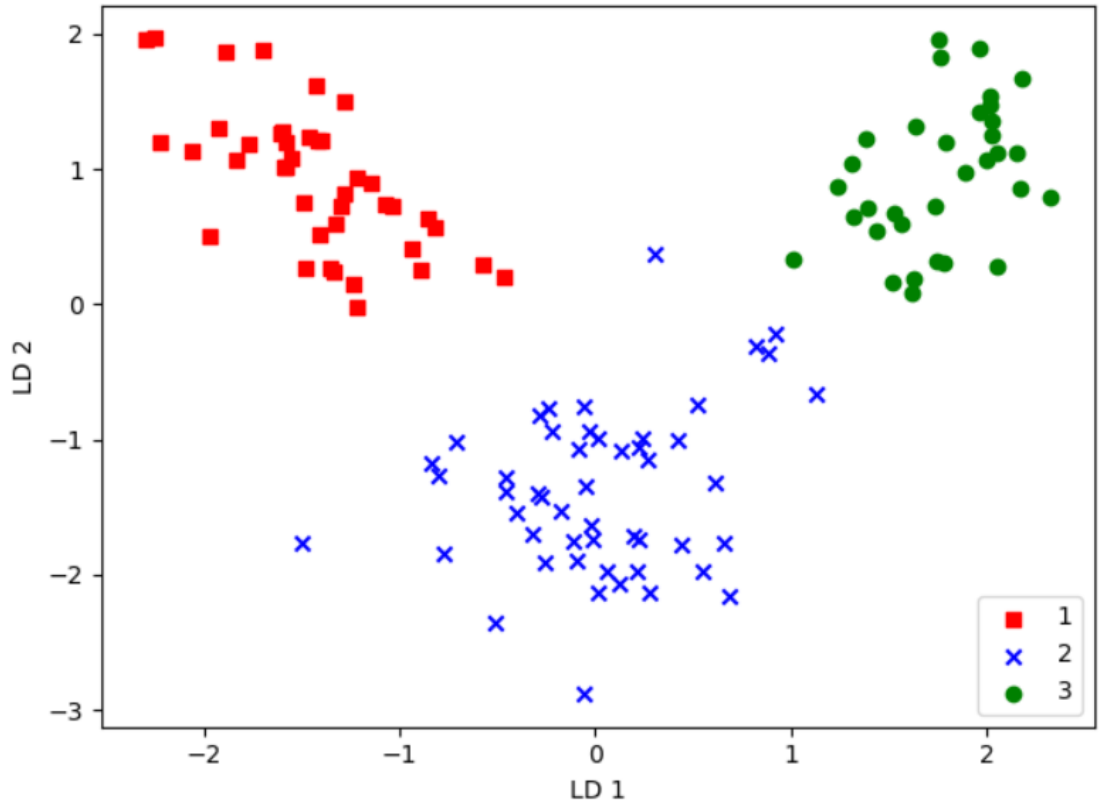


Projecting samples onto the new feature space

7. Project the samples onto te new feature subspace using the projection matrix W $X' = XW$

ch05tmp3.py

```
X_train_lda = X_train_std.dot(w)
colors = ['r', 'b', 'g']
markers = ['s', 'x', 'o']

for l, c, m in zip(np.unique(y_train), colors, markers):
    plt.scatter(X_train_lda[y_train == l, 0],
                X_train_lda[y_train == l, 1] * (-1),
                c=c, label=l, marker=m)

plt.xlabel('LD 1')
plt.ylabel('LD 2')
plt.legend(loc='lower right')
plt.tight_layout()
# plt.savefig('images/05_08.png', dpi=300)
plt.show()
```



LDA via scikit-learn

ch05tmp3.py

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA

lda = LDA(n_components=2)
X_train_lda = lda.fit_transform(X_train_std, y_train)


# In[35]:


from sklearn.linear_model import LogisticRegression
lr = LogisticRegression()
lr = lr.fit(X_train_lda, y_train)

plot_decision_regions(X_train_lda, y_train, classifier=lr)
plt.xlabel('LD 1')
plt.ylabel('LD 2')
plt.legend(loc='lower left')
plt.tight_layout()
# plt.savefig('images/05_09.png', dpi=300)
plt.show()


# In[36]:


X_test_lda = lda.transform(X_test_std)

plot_decision_regions(X_test_lda, y_test, classifier=lr)
plt.xlabel('LD 1')
plt.ylabel('LD 2')
plt.legend(loc='lower left')
plt.tight_layout()
# plt.savefig('images/05_10.png', dpi=300)
plt.show()
```