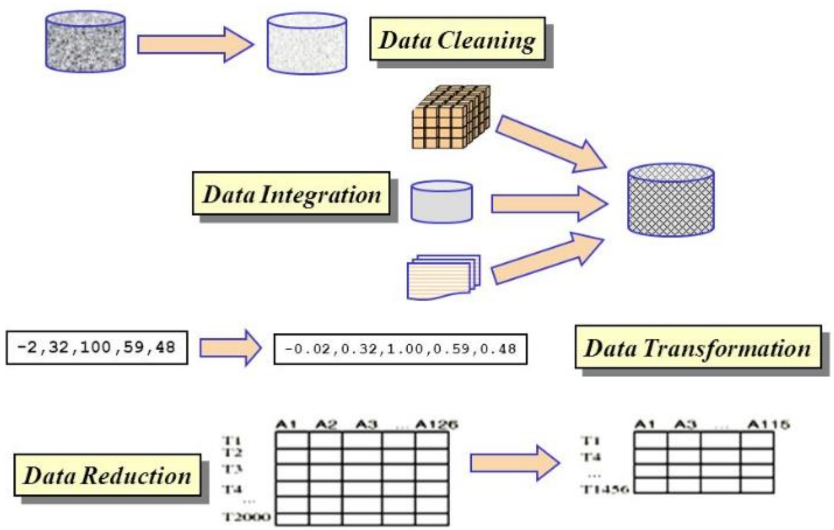
Data Preprocessing

Machine Learning

Data Preprocessing : Build Good Training Sets

- Building Good Training Sets
  - Dealing with missing data
  - Handling categorical data
  - Bringing features onto the same scale
  - Selecting meaningful features
  - Dimensionality reduction via PCA



Data Preprocessing : Build Good Training Sets

- sklearn.preprocessing
  - Encoding, scaling, standardizing, etc.
- sklearn.decomposition
  - Matrix decomposition algorithms, including PCA
- sklearn.impute (v0.20 ~)
  - Imputing

- Methods
  - fit() : fit the model for preprocessing
  - transform() : transform feature values
  - fit_transform() : fit + transform

※ fit() using training data

Dealing with missing data

- Load Sample data

```
import pandas as pd
df = pd.read_csv('sample-missing-data.csv')
df
```

	A	B	C	label
0	1	0.1	10.0	0
1	2	0.2	20.0	0
2	2	0.1	NaN	1
3	3	0.3	20.0	1
4	2	0.2	NaN	0

- Removing a feature

```
df.drop("A", axis=1)
```

	B	C	label
0	0.1	10.0	0
1	0.2	20.0	0
2	0.1	NaN	1
3	0.3	20.0	1
4	0.2	NaN	0

Dealing with missing data

- Removing data with missing values

```
# check missing values
df.isnull().sum()
```

```
A     0
B     0
C     2
label 0
dtype: int64
```

```
# remove rows that contain missing values
df.dropna(axis=0)
```

```
# remove colomns that contain missing values
df.dropna(axis=1)
```

Original data

	A	B	C	label
0	1	0.1	10.0	0
1	2	0.2	20.0	0
2	2	0.1	NaN	1
3	3	0.3	20.0	1
4	2	0.2	NaN	0

	A	B	C	label
0	1	0.1	10.0	0
1	2	0.2	20.0	0
3	3	0.3	20.0	1

	A	B	label
0	1	0.1	0
1	2	0.2	0
2	2	0.1	1
3	3	0.3	1
4	2	0.2	0

Dealing with missing data

- Imputing missing values - Pandas

```
# Impute using pandas
df.fillna(df.mean())
```

	A	B	C	label
0	1	0.1	10.0	0
1	2	0.2	20.0	0
2	2	0.1	NaN	1
3	3	0.3	20.0	1
4	2	0.2	NaN	0

	A	B	C	label
0	1	0.1	10.000000	0
1	2	0.2	20.000000	0
2	2	0.1	16.666667	1
3	3	0.3	20.000000	1
4	2	0.2	16.666667	0

Dealing with missing data

- Imputing missing values - Scikit-learn

```
# Impute using scikit-learn
# sklearn v0.18
# from sklearn.preprocessing import Imputer
# imr = Imputer(missing_values='NaN', strategy='mean', axis=0)

# sklearn >= v0.20
from sklearn.impute import SimpleImputer
imr = SimpleImputer(missing_values=np.nan, strategy='mean')

imr = imr.fit(df.values)
imputed_data = imr.transform(df.values)
imputed_data
```

```
array([[ 1. ,  0.1, 10. ,  0. ],
       [ 2. ,  0.2, 20. ,  0. ],
       [ 2. ,  0.1,  nan,  1. ],
       [ 3. ,  0.3, 20. ,  1. ],
       [ 2. ,  0.2,  nan,  0. ]])
```

→

```
array([[ 1.        ,  0.1       , 10.        ,  0.        ],
       [ 2.        ,  0.2       , 20.        ,  0.        ],
       [ 2.        ,  0.1       , 16.66666667,  1.        ],
       [ 3.        ,  0.3       , 20.        ,  1.        ],
       [ 2.        ,  0.2       , 16.66666667,  0.        ]])
```

Handling categorical feature values

- Sample data

```
import pandas as pd

df = pd.DataFrame([['green', 'M', 10.1, 'class1'],
                   ['red', 'L', 13.5, 'class1'],
                   ['blue', 'XL', 15.3, 'class3'],
                   ['red', 'M', 14.5, 'class2']])

df.columns = ['color', 'size', 'price', 'classlabel']
df
```

	color	size	price	classlabel
0	green	M	10.1	class1
1	red	L	13.5	class1
2	blue	XL	15.3	class3
3	red	M	14.5	class2

Handling categorical feature values

- Encoding class labels - scikit-learn

```
from sklearn.preprocessing import LabelEncoder
# Label encoding using sklearn
le = LabelEncoder()
df['classlabel'] = le.fit_transform(df['classlabel'])
df
```

	color	size	price	classlabel
0	green	M	10.1	0
1	red	L	13.5	0
2	blue	XL	15.3	2
3	red	M	14.5	1

Handling categorical feature values

- Encoding ordinal features – pandas

```
# encoding ordinal features using pandas
size_mapping = {'M': 0, 'L': 1, 'XL': 2 }
df['size'] = df['size'].map(size_mapping)
df
```

	color	size	price	classlabel
0	green	0	10.1	0
1	red	1	13.5	0
2	blue	2	15.3	2
3	red	0	14.5	1

Handling categorical feature values

- One-hot encoding of nominal features – pandas

```
# one-hot encoding via pandas
pd.get_dummies(df, columns=['color'])
```

	size	price	classlabel	color_blue	color_green	color_red
0	0	10.1	0	0	1	0
1	1	13.5	0	0	0	1
2	2	15.3	2	1	0	0
3	0	14.5	1	0	0	1

Handling categorical feature values

- One-hot encoding of nominal features - scikit learn

```
# Convert pd.DataFrame object to np.array object
X = df[['color', 'size', 'price']].values
X
```

```
array([['green', 0, 10.1],
       ['red', 1, 13.5],
       ['blue', 2, 15.3],
       ['red', 0, 14.5]], dtype=object)
```

Handling categorical feature values

- One-hot encoding of nominal features - scikit learn

```
# sklearn v0.18
# Step 1) Convert 'color' feature data type string to integer
from sklearn.preprocessing import LabelEncoder
# label encoding
le = LabelEncoder()
X[:, 0] = le.fit_transform(X[:, 0])
print(X)

# Step 2) One-hot encode 'color' feature
from sklearn.preprocessing import OneHotEncoder
# one-hot encoding using scikit-learn
ohe = OneHotEncoder(categorical_features=[0], sparse=False)
X = ohe.fit_transform(X)
print(X)
```

```
[[1 0 10.1]
 [2 1 13.5]
 [0 2 15.3]
 [2 0 14.5]]
[[  0.    1.    0.    0.   10.1]
 [  0.    0.    1.    1.   13.5]
 [  1.    0.    0.    2.   15.3]
 [  0.    0.    1.    0.   14.5]]
```

Handling categorical feature values

- One-hot encoding of nominal features - scikit learn

```
# sklearn >= v0.20
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
# ColumnTransformer helps you select which column(feature) you want to transform
# In sklearn >= v0.20, OneHotEncoder() can directly transform string type
features
# ("process name", transformer, columns(np.array : column idx, pd.DataFrame :
column name ))
ct = ColumnTransformer([("Ohe", OneHotEncoder(), [0])],
                       remainder='passthrough')
X = ct.fit_transform(X)
X
```

```
array([[0.0, 1.0, 0.0, 0, 10.1],
       [0.0, 0.0, 1.0, 1, 13.5],
       [1.0, 0.0, 0.0, 2, 15.3],
       [0.0, 0.0, 1.0, 0, 14.5]], dtype=object)
```

Transformation of numerical feature values

- Loading Wine Dataset

```
import pandas as pd
import numpy as np
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']

print('Class labels', np.unique(df_wine['Class label']))
df_wine.head()
```

```
Class labels [1 2 3]
```

	Class label	Alcohol	Malic acid	Ash	Alcalinity of ash	Magnesium	Total phenols	Flavanoids	Nonflavanoid phenols	Prc
0	1	14.23	1.71	2.43	15.6	127	2.80	3.06	0.28	
1	1	13.20	1.78	2.14	11.2	100	2.65	2.76	0.26	
2	1	13.16	2.36	2.67	18.6	101	2.80	3.24	0.30	
3	1	14.37	1.95	2.50	16.8	113	3.85	3.49	0.24	
4	1	13.24	2.59	2.87	21.0	118	2.80	2.69	0.39	

Transformation of numerical feature values

- Get X and y. Splitting data into 70% training & 30% test

```
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,
                                                    random_state=0,
                                                    stratify=y)
```

Transformation of numerical feature values

- Normalization

```
from sklearn.preprocessing import MinMaxScaler
# transform to min 0, max 1
mms = MinMaxScaler()
X_train_norm = mms.fit_transform(X_train)
X_test_norm = mms.transform(X_test)
```

```
print(X_train[0:3, 0:5])
print('max[0] = %.2f \n' % X_train[:,0].max())
print(X_train_norm[0:3, 0:5])
print('max[0] = %.2f \n' % X_train_norm[:,0].max())
```

```
[[ 13.62   4.95   2.35  20.     92.   ]
 [ 13.76   1.53   2.7   19.5   132.   ]
 [ 13.73   1.5    2.7   22.5   101.   ]]
max[0] = 14.83

[[0.64619883 0.83201581 0.4248366  0.46236559 0.27160494]
 [0.6871345  0.15612648 0.65359477 0.43548387 0.7654321 ]
 [0.67836257 0.15019763 0.65359477 0.59677419 0.38271605]]
max[0] = 1.00
```

Transformation of numerical feature values

- Standardization

```python
from sklearn.preprocessing import StandardScaler
# transform to mean 0, variance 1
stdsc = StandardScaler()
X_train_std = stdsc.fit_transform(X_train)
X_test_std = stdsc.transform(X_test)
```

```python
print(X_train[0:3, 0:5])
print('mean[0] = %.2f \n' % X_train[:,0].mean())
print(X_train_std[0:3, 0:5])
print('mean[0] = %.2f \n' % X_train_std[:,0].mean())
```

```
[[ 13.62    4.95    2.35   20.      92.   ]
 [ 13.76    1.53    2.7    19.5    132.   ]
 [ 13.73    1.5     2.7    22.5    101.   ]]
mean[0] = 13.03

[[ 0.71225893  2.22048673 -0.13025864  0.05962872 -0.50432733]
 [ 0.88229214 -0.70457155  1.17533605 -0.09065504  2.34147876]
 [ 0.84585645 -0.73022996  1.17533605  0.81104754  0.13597904]]
mean[0] = 0.00
```

Feature Selection

- Select Meaningful Features using information gain

```
from sklearn.tree import DecisionTreeClassifier
tree = DecisionTreeClassifier(random_state=0)
tree.fit(X_train, y_train)
print('Training accuracy:', tree.score(X_train, y_train))
print('Test accuracy:', tree.score(X_test, y_test))
```

```
Training accuracy: 1.0
Test accuracy: 0.9444444444444444
```

```
feature_labels = df_wine.columns[1:]
importances = tree.feature_importances_
indices = np.argsort(importances)[::-1]
for f in range(X_train.shape[1]):
print("%2d. %-30s %f" % (f+1,
feature_labels[indices[f]],
importances[indices[f]]))
```

```
 1. Proline                        0.439462
 2. Flavanoids                     0.397143
 3. Color intensity                0.105658
 4. Alcalinity of ash              0.023926
 5. Alcohol                        0.017451
```

⋮

Feature Selection

- Select Meaningful Features using L1 regularization

```
from sklearn.linear_model import LogisticRegression

# Logistic regression with almost no regularization. Lamda = 0.0001
# C=inverse of lambda. Note that C=1.0 is the default
lr = LogisticRegression(penalty='l1', C=10000, solver='liblinear')
lr.fit(X_train_std, y_train)

print('Training accuracy:', lr.score(X_train_std, y_train))
print('Test accuracy:', lr.score(X_test_std, y_test))
```

```
Training accuracy: 1.0
Test accuracy: 1.0
```

Feature Selection

- Select Meaningful Features using L1 regularization

```
lr.coef_
```

```
array([[  7.77314758e+00,   1.91412612e+00,   3.99377441e+00,
         -6.24484378e+00,   7.47326622e-01,   2.20339655e-01,
          5.26986328e+00,   7.43521256e-01,   1.29756653e+00,
         -1.93150719e+00,  -1.65773896e+00,   3.47507659e+00,
          9.01387622e+00],
       [ -6.95043777e+00,  -3.22019911e+00,  -8.04217082e+00,
          4.41324186e+00,  -1.28124736e+00,  -4.13693560e-02,
          5.26677198e+00,   2.47284036e+00,  -8.25309639e-01,
         -1.94628666e+01,   6.05599053e+00,   9.79524942e-01,
         -1.65213728e+01],
       [  4.89432308e+00,   8.97270667e-01,   5.06763502e+00,
         -5.56383532e-03,  -2.00440234e-01,   1.46945513e+00,
         -1.02115043e+01,  -2.55706918e+00,  -2.31053380e+00,
          1.00472035e+01,  -7.58511750e+00,  -4.05718438e+00,
          2.66979146e-01]])
```

Feature Selection

- Select Meaningful Features using L1 regularization

```
from sklearn.linear_model import LogisticRegression

# Logistic regression with L1 regularization. Lamda = 1
lr = LogisticRegression(penalty='l1', C=1, solver='liblinear')
lr.fit(X_train_std, y_train)

print('Training accuracy:', lr.score(X_train_std, y_train))
print('Test accuracy:', lr.score(X_test_std, y_test))
```

```
Training accuracy: 1.0
Test accuracy: 1.0
```

Feature Selection

- Select Meaningful Features using L1 regularization

```
lr.coef_
```

```
array([[ 1.24599619,  0.18053863,  0.74458045, -1.16193969,  0.        ,
         0.        ,  1.16403191,  0.        ,  0.        ,  0.        ,
         0.        ,  0.55287851,  2.50953582],
       [-1.53733751, -0.38713713, -0.99521311,  0.36500237, -0.05944925,
         0.        ,  0.66802847,  0.        ,  0.        , -1.93439049,
         1.23344405,  0.        , -2.23141931],
       [ 0.13503233,  0.16979919,  0.35796836,  0.        ,  0.        ,
         0.        , -2.43231118,  0.        ,  0.        ,  1.56174978,
        -0.81730975, -0.49913568,  0.        ]])
```

```
lr.coef_[lr.coef_==0].shape
```

```
(16,)
```

Feature Selection

- Select Meaningful Features using L1 regularization

```
array([[  7.77314758e+00,   1.91412612e+00,   3.99377441e+00,
         -6.24484378e+00,   7.47326622e-01,   2.20339655e-01,
          5.26986328e+00,   7.43521256e-01,   1.29756653e+00,
         -1.93150719e+00,  -1.65773896e+00,   3.47507659e+00,
          9.01387622e+00],
       [ -6.95043777e+00,  -3.22019911e+00,  -8.04217082e+00,
          4.41324186e+00,  -1.28124736e+00,  -4.13693560e-02,
          5.26677198e+00,   2.47284036e+00,  -8.25309639e-01,
         -1.94628666e+01,   6.05599053e+00,   9.79524942e-01,
         -1.65213728e+01],
       [  4.89432308e+00,   8.97270667e-01,   5.06763502e+00,
         -5.56383532e-03,  -2.00440234e-01,   1.46945513e+00,
         -1.02115043e+01,  -2.55706918e+00,  -2.31053380e+00,
          1.00472035e+01,  -7.58511750e+00,  -4.05718438e+00,
          2.66979146e-01]])
```

```
array([[ 1.24599619,  0.18053863,  0.74458045, -1.16193969,  0.        ,
         0.        ,  1.16403191,  0.        ,  0.        ,  0.        ,
         0.        ,  0.55287851,  2.50953582],
       [-1.53733751, -0.38713713, -0.99521311,  0.36500237, -0.05944925,
         0.        ,  0.66802847,  0.        ,  0.        , -1.93439049,
         1.23344405,  0.        , -2.23141931],
       [ 0.13503233,  0.16979919,  0.35796836,  0.        ,  0.        ,
         0.        , -2.43231118,  0.        ,  0.        ,  1.56174978,
        -0.81730975, -0.49913568,  0.        ]])
```

Dimensionality Reduction

- Loading Wine dataset

```
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()
```

	Class label	Alcohol	Malic acid	Ash	Alcalinity of ash	Magnesium	Total phenols	Flavanoids	Nonflavan phen
0	1	14.23	1.71	2.43	15.6	127	2.80	3.06	0
1	1	13.20	1.78	2.14	11.2	100	2.65	2.76	0
2	1	13.16	2.36	2.67	18.6	101	2.80	3.24	0
3	1	14.37	1.95	2.50	16.8	113	3.85	3.49	0
4	1	13.24	2.59	2.87	21.0	118	2.80	2.69	0

Dimensionality Reduction

- Get X and y. Splitting data into 70% training & 30% test

```
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)
```

Dimensionality Reduction

- Principal Component Analysis using numpy

```
from sklearn.preprocessing import StandardScaler

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

```
X_train_std.shape
```

```
(124, 13)
```

```
import numpy as np
# covariance matrix of X
cov_mat = np.cov(X_train_std.T)
cov_mat
```

```
array([[ 1.00813008,  0.06709556,  0.17405351, -0.35439069,  0.26374703,
         0.29079481,  0.21835807, -0.08111974,  0.10436705,  0.54282846,
         0.05893536, -0.01797029,  0.6415292 ],
       [ 0.06709556,  1.00813008,  0.08326463,  0.26356776, -0.11349172,
        -0.33735555, -0.41035281,  0.33653916, -0.21602672,  0.17504154,
        -0.551593  , -0.40561695, -0.24089991],
                                   ⋮
```

```
cov_mat.shape
```

```
(13, 13)
```

Dimensionality Reduction

- Principal Component Analysis using numpy

```
# eigenvalues and eigenvectors of covariance matrix
eigen_vals, eigen_vecs = np.linalg.eig(cov_mat)
print('\nEigenvalues \n%s' % eigen_vals)
print('\nEigenvectors \n%s' % eigen_vecs)
```

```
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 ]

Eigenvectors
[[ -1.37242175e-01   5.03034778e-01  -1.37748734e-01  -3.29610003e-03
   -2.90625226e-01   2.99096847e-01   7.90529293e-02  -3.68176414e-01
   -3.98377017e-01  -9.44869777e-02   3.74638877e-01  -1.27834515e-01
    2.62834263e-01]
 [  2.47243265e-01   1.64871190e-01   9.61503863e-02   5.62646692e-01
    8.95378697e-02   6.27036396e-01  -2.74002014e-01  -1.25775752e-02
    1.10458230e-01   2.63652406e-02  -1.37405597e-01   8.06401578e-02
   -2.66769211e-01]
```

⋮

```
eigen_vecs.shape
```

```
(13, 13)
```


dongguk UNIVERSITY

Dimensionality Reduction

- Principal Component Analysis using numpy
  - Projection to new 13 dimensions

```
X_train_pca = X_train_std.dot(eigen_vecs)
```

```
X_train_std[0]
```

```
array([ 0.71225893,  2.22048673, -0.13025864,  0.05962872, -0.50432733,
       -0.52831584, -1.24000033,  0.84118003, -1.05215112, -0.29218864,
       -0.20017028, -0.82164144, -0.62946362])
```

```
X_train_pca[0]
```

```
array([ 2.38299011,  0.45458499, -0.22703207,  0.57988399, -0.57994169,
        1.73317476, -0.70180475, -0.21617248, -0.23666876,  0.40161994,
        0.16548767, -0.23489704, -0.29726982])
```

Dimensionality Reduction

- Principal Component Analysis using numpy
  - Projection to 2 dimensions corresponding to 2 largest eigenvalues

```
w = eigen_vecs[:, [0, 1]]
w
```

```
array([[-0.13724218,  0.50303478],
       [ 0.24724326,  0.16487119],
       [-0.02545159,  0.24456476],
       [ 0.20694508, -0.11352904],
       [-0.15436582,  0.28974518],
                  ⋮
```

```
X_train_pca = X_train_std.dot(w)
```

```
X_train_std[0]
```

```
array([ 0.71225893,  2.22048673, -0.13025864,  0.05962872, -0.50432733,
       -0.52831584, -1.24000033,  0.84118003, -1.05215112, -0.29218864,
       -0.20017028, -0.82164144, -0.62946362])
```

```
X_train_pca[0]
```

```
array([ 2.38299011,  0.45458499])
```

Dimensionality Reduction

- Principal Component Analysis using scikit-learn

```
from sklearn.decomposition import PCA

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

```
X_train_std[0]
```

```
array([ 0.71225893,  2.22048673, -0.13025864,  0.05962872, -0.50432733,
       -0.52831584, -1.24000033,  0.84118003, -1.05215112, -0.29218864,
       -0.20017028, -0.82164144, -0.62946362])
```

```
X_train_pca[0]
```

```
array([ 2.38299011,  0.45458499, -0.22703207,  0.57988399, -0.57994169,
        1.73317476, -0.70180475, -0.21617248, -0.23666876,  0.40161994,
        0.16548767, -0.23489704, -0.29726982])
```

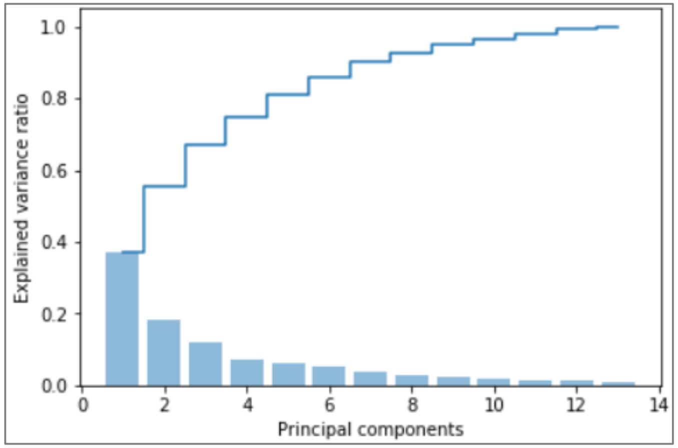
Dimensionality Reduction

- Explained variance ratio for each component

```
pca.explained_variance_ratio_
```

```
array([0.36951469, 0.18434927, 0.11815159, 0.07334252, 0.06422108,
       0.05051724, 0.03954654, 0.02643918, 0.02389319, 0.01629614,
       0.01380021, 0.01172226, 0.00820609])
```

```
import matplotlib.pyplot as plt
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()
```



dongguk UNIVERSITY

Dimensionality Reduction

- Projection to 2 dimensions

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

```
X_train_std[0]
```

```
array([ 0.71225893,  2.22048673, -0.13025864,  0.05962872, -0.50432733,
       -0.52831584, -1.24000033,  0.84118003, -1.05215112, -0.29218864,
       -0.20017028, -0.82164144, -0.62946362])
```

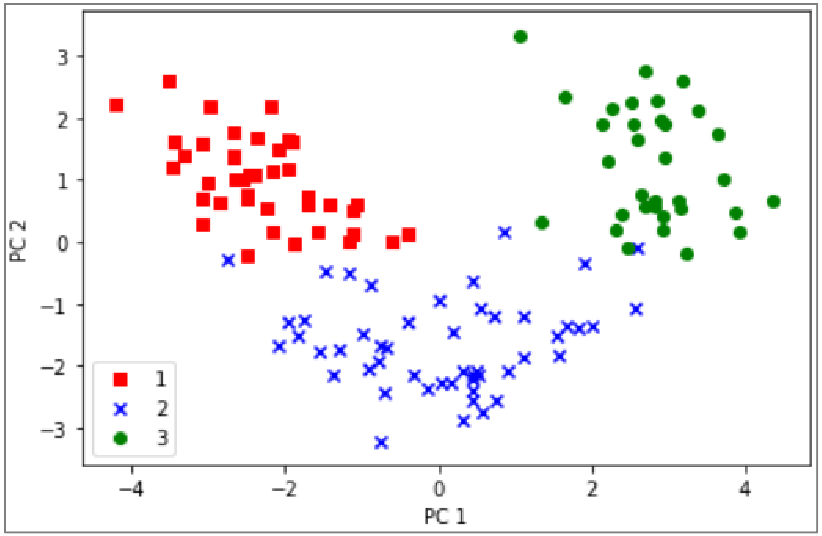
```
X_train_pca[0]
```

```
array([ 2.38299011,  0.45458499])
```

Dimensionality Reduction

- Visualizing data in 2 dimensions with PC1 and PC2

```
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.show()
```



Dimensionality Reduction

- Logistic regression with 2 dimensional data

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

```
acc = lr.score(X_train_pca, y_train)
print("Train accuracy : %.4f" % acc)
```

```
Train accuracy : 0.9839
```

```
acc = lr.score(X_test_pca, y_test)
print("Test accuracy : %.4f" % acc)
```

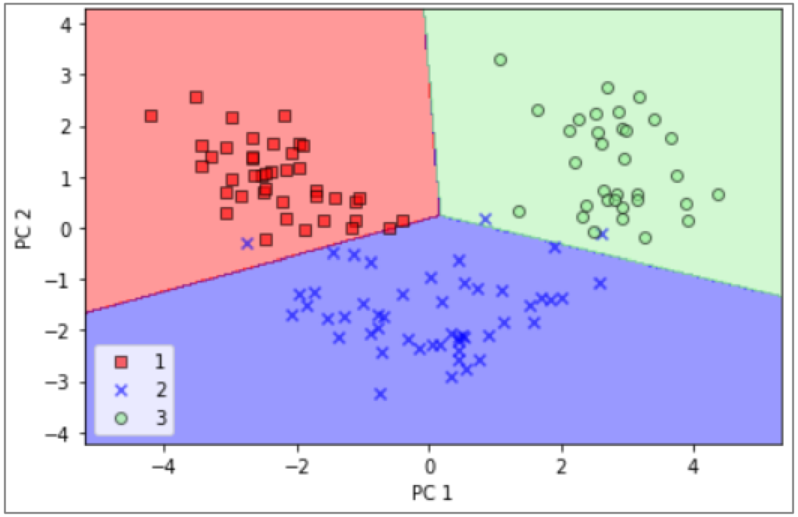
```
Test accuracy : 0.9259
```

Dimensionality Reduction

- Decision boundary

```
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.show()
```



Submit

- To make sure if you have completed this practice,
  Submit your practice file(Week08_givencode.ipynb) to e-class.
- **Deadline : tomorrow 11:59pm**
- Modify your ipynb file name as “Week08_StudentNum_Name.ipynb”
  Ex) **Week08_2020123456_홍길동.ipynb**
- You can upload this file without taking the quiz, but homework will be provided like a quiz every three weeks, so it is recommended to take the quiz as well.

Quiz : Kaggle Tutorial Competition

- Predict survival on the Titanic
  - Data Preprocessing
    1. Remove irrelevant features
    2. Convert categorical features to numerical
    3. Impute missing data
    4. Standardize numerical features
    5. Dimensionality Reduction by PCA on numerical features

  - Build and Evaluate Model
    - Logistic Regression
    - Decision Tree
    - Gaussian Naïve Bayes
    - K-Nearest Neighbors
    - Multilayer Neural Network

https://www.kaggle.com/c/titanic/data

Quiz : Kaggle Tutorial Competition

- Titanic: Machine Learning from Disaster
  - Data Dictionary (891 samples)

Variable	Definition	Key
Survived	Survival	0 = No, 1 = Yes
Pclass	Ticket class	1 = 1st, 2 = 2nd, 3 = 3rd
Sex	Sex	
Age	Age in years	
Sibsp	# of siblings / spouses aboard the Titanic	
Parch	# of parents / children aboard the Titanic	
Ticket	Ticket number	
Fare	Passenger fare	
Cabin	Cabin number	
Embarked	Port of Embarkation	C = Cherbourg Q = Queenstown S = Southampton

https://www.kaggle.com/c/titanic/data