동국대학교
dongguk university

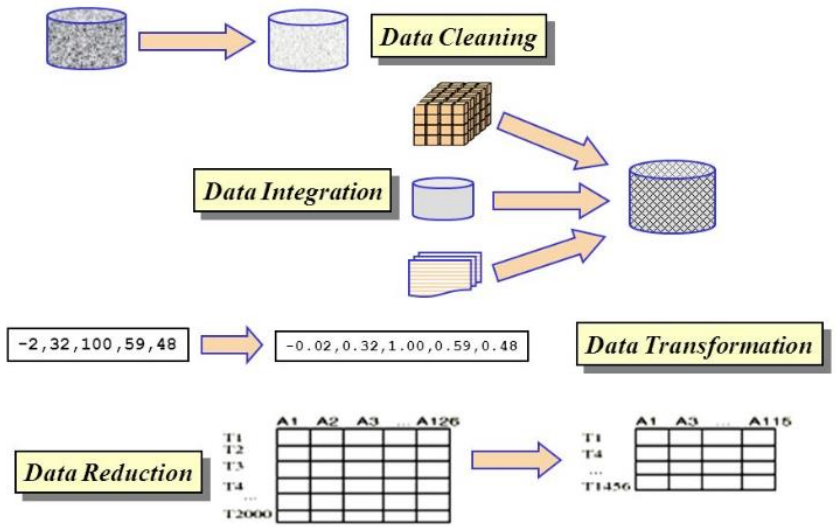
Data Preprocessing

Machine Learning

(PM chap 4, 5)

Data Preprocessing : Build Good Training Sets

- Building Good Training Sets
  - Integration
  - Cleaning - Dealing with missing data, outliers
  - Transformation - Handling categorical data, bringing features onto the same scale
  - Reduction - Selecting meaningful features, dimensionality reduction



Data Preprocessing : Build Good Training Sets

- Example
  - Titanic dataset – predict ‘survived’

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S
...	...	...	...	...	...	...	...	...	...	...	...	...
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13.0000	NaN	S
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.0000	B42	S
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.4500	NaN	S
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.0000	C148	C
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.7500	NaN	Q

Data Preprocessing : Build Good Training Sets

- Sklearn.preprocessing
  - Encoding, scaling, standardizing, etc.
- Sklearn.decomposition
  - Matrix decomposition algorithms, including PCA

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

※ fit() using training data

Missing Values

- Pandas - NaN
- Eliminating samples or features with missing values. dropna(axis=0)

	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

- Imputing missing values. 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

Missing Values

- Imputing missing values using scikit-learn
  - impute.SimpleImputer()

```
# get X array from df dataframe
X = df.values
print(X)
```

```
[[ 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. ]]
```

```
from sklearn.impute import SimpleImputer

# replace missing values, encoded as np.nan, using the mean value of the columns
imr = SimpleImputer(missing_values=np.nan, strategy='mean')
imr.fit(X)

imputed_X = imr.transform(X)
print(imputed_X)
```

```
[[ 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.        ]]
```

Categorical Values

- Encoding class labels - pandas
  - .map(<dictionary>)

```
label_mapping = {'class1': 0, 'class2': 1, 'class3': 2 }
pdf['classlabel'] = pdf['classlabel'].map(label_mapping)
```

- Encoding class labels - scikit-learn
  - preprocessing.LabelEncoder()

```
enc = LabelEncoder()
sdf['classlabel'] = enc.fit_transform(sdf['classlabel'])
```

	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

	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

Categorical Values

- Encoding features
  - Ordinal feature - Order information should be encoded CORRECTLY

	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

	color	size	price	classlabel
0	1.0	1.0	10.1	0
1	2.0	0.0	13.5	0
2	0.0	2.0	15.3	2
3	2.0	1.0	14.5	1

X
(L < M < XL)

  - Nominal feature - Order information should NOT be encoded

	color	size	price	classlabel
0	1.0	1.0	10.1	0
1	2.0	0.0	13.5	0
2	0.0	2.0	15.3	2
3	2.0	1.0	14.5	1

X
(no order between colors)

Categorical Values

- Encoding ordinal features - pandas
  - .map(<dictionary>)

```
size_mapping = {'M': 0, 'L': 1, 'XL': 2 }
pdf['size'] = pdf['size'].map(size_mapping)
```

- Encoding ordinal features – scikit learn
  - preprocessing.OrdinalEncoder()

```
enc = OrdinalEncoder(categories=[['M', 'L', 'XL']])
sdf[['size']] = enc.fit_transform(sdf[['size']])
```

	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

	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

o

Categorical Values

- One-hot encoding of nominal features - pandas
  - .get_dummies()

```
pd.get_dummies(pdf, columns=['color'])
```

- One-hot encoding of nominal features - scikit learn
  - OneHotEncoder()

```
enc = OneHotEncoder(sparse=False)

color_enc = enc.fit_transform(sdf[['color']])
sdf_color_enc = pd.DataFrame(color_enc, columns=['color0', 'color1', 'color2',])
sdf = pd.concat([sdf_color_enc, sdf[['size', 'price', 'classlabel']]],axis=1)
```

	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

	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

o

Transformation of Numerical Values

- Brining features onto the same scale
- Normalization
  - Change all values in the range [0, 1]

$$x_{new} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

- Standardization
  - Transform all values to have zero mean and unit variance

$$x_{new} = \frac{x - \mu}{\sigma}$$

Transformation of Numerical Values

- Example - Wine Dataset
  - From 13 features → predict class label (1, 2, 3)

```
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']
```

	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
0	1	14.23	1.71	2.43	15.6	127	2.80	3.06	0.28	2.29	5.64	1.04	3.92	1065
1	1	13.20	1.78	2.14	11.2	100	2.65	2.76	0.26	1.28	4.38	1.05	3.40	1050
2	1	13.16	2.36	2.67	18.6	101	2.80	3.24	0.30	2.81	5.68	1.03	3.17	1185
3	1	14.37	1.95	2.50	16.8	113	3.85	3.49	0.24	2.18	7.80	0.86	3.45	1480
4	1	13.24	2.59	2.87	21.0	118	2.80	2.69	0.39	1.82	4.32	1.04	2.93	735

Transformation of Numerical Values

- Normalization, standardization (using training data)
  - MinMaxScaler(), StandardScaler()

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

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
```

```
mms = MinMaxScaler()
X_train_norm = mms.fit_transform(X_train)
X_test_norm = mms.transform(X_test)
```

```
[[  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.  ]]
```

```
[[ 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] = 1.0
```

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

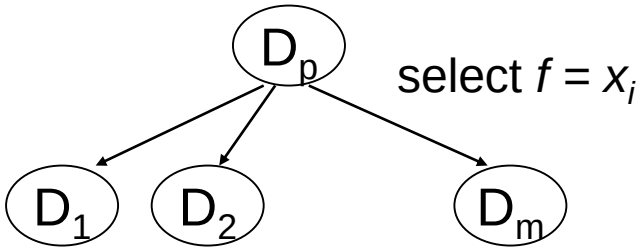
```
[[  13.62    4.95    2.35   20.     92.  ]
 [  13.76    1.53    2.7    19.5   132.  ]
 [  13.73    1.5     2.7    22.5   101.  ]]
```

```
[[ 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
  - **feature_importances_** in Decision tree
  - The importance of a feature is computed as the (normalized) total reduction of the criterion brought by that feature

    ➡ Use features with high importance



$$IG(D_p, f) = I(D_p) - \sum_{j=1}^{m} \frac{N_j}{N_p} I(D_j)$$

Feature Selection

- Example
  - Learn the tree

```
tree = DecisionTreeClassifier(random_state=0)
tree.fit(X_train, y_train)
```

  - Get the feature importances

```
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]]))
```

```
[0.01745058 0.         0.         0.02392638 0.         0.
 0.39714318 0.01635992 0.         0.10565781 0.         0.
 0.43946214]
[12  6  9  3  0  7 11 10  8  5  4  2  1]
 1. Proline                        0.439462
 2. Flavanoids                     0.397143
 3. Color intensity                0.105658
 4. Alcalinity of ash              0.023926
 5. Alcohol                        0.017451
 6. Nonflavanoid phenols           0.016360
 7. OD280/OD315 of diluted wines   0.000000
 8. Hue                            0.000000
 9. Proanthocyanins                0.000000
10. Total phenols                  0.000000
11. Magnesium                      0.000000
12. Ash                            0.000000
13. Malic acid                     0.000000
```

Feature Selection

- Select meaningful features using regularization
  - L1 and L2 regularization as penalties against model complexity
  - L2 regularization is one approach to reduce the complexity of a model by penalizing large individual weights

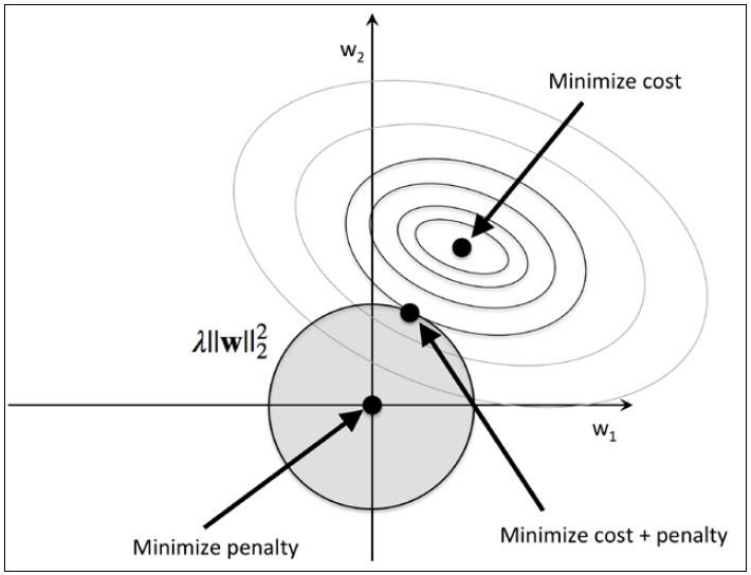
$$L_2: \|\mathbf{w}\|_2^2 = \sum_{j=1}^{m} w_j^2 \quad \Rightarrow \quad w_j = w_j - \alpha \frac{\partial J}{\partial w_j} - \alpha\lambda w_j$$

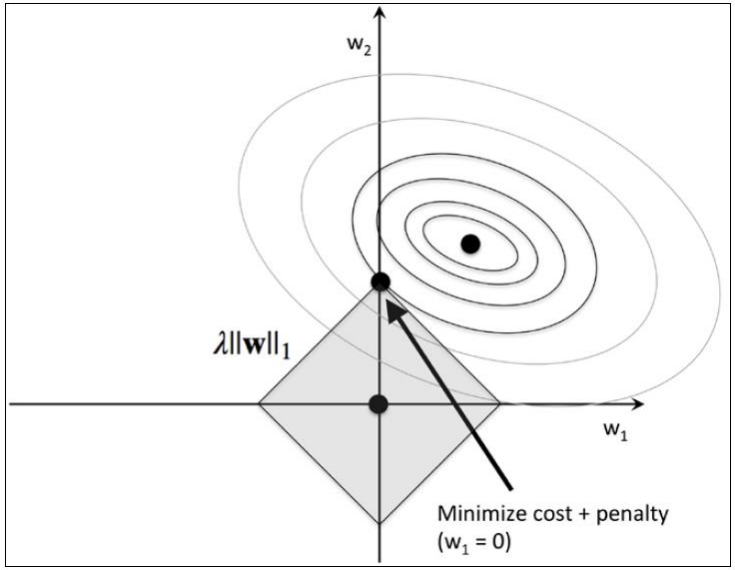
  - Another approach to reduce the model complexity is L1 regularization. With L1, small weight goes to 0.

$$L_1: \|\mathbf{w}\|_1 = \sum_{j=1}^{m} |w_j| \quad \Rightarrow \quad w_j = w_j - \alpha \frac{\partial J}{\partial w_j} - \alpha\lambda$$

Feature Selection

- Select meaningful features using regularization





L_2 regularization

$$Cost = J(w) + \lambda \sum_{j=0}^{n} w_j^2$$

L_1 regularization

$$Cost = J(w) + \lambda \sum_{j=0}^{n} |w_j|$$

Feature Selection

- Example - Logistic regression with L1 regularization

```
# 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
```

```
# Learned parameters with L1
lr.coef_
```

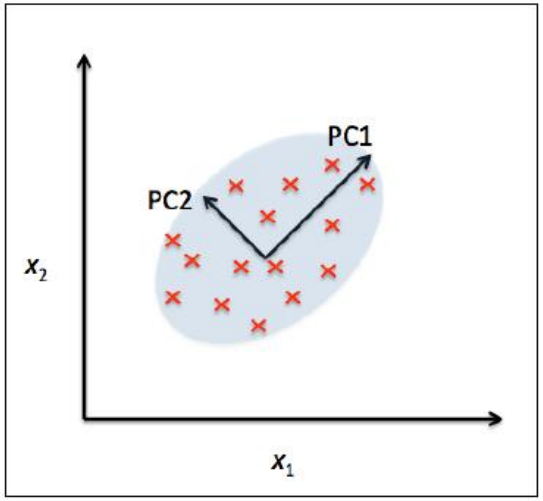
```
array([[ 1.24621325,  0.18053788,  0.74592794, -1.16359758,  0.        ,
         0.        ,  1.16054466,  0.        ,  0.        ,  0.        ,
         0.        ,  0.55619053,  2.50878509],
       [-1.53653546, -0.38772029, -0.99500573,  0.36495134, -0.0597017 ,
         0.        ,  0.66781059,  0.        ,  0.        , -1.9342868 ,
         1.23335332,  0.        , -2.23126477],
       [ 0.13547325,  0.16858336,  0.35732342,  0.        ,  0.        ,
         0.        , -2.43747781,  0.        ,  0.        ,  1.56369187,
        -0.81842988, -0.49300554,  0.        ]])
```

Feature Selection

- Sequential feature selection
  - Aims to reduce the dimensionality of the initial feature subspace with a minimum decay in performance of the classifier
  - Sequential Backward Selection(SBS) algorithm
    1. Initialize the algorithm with $k = d$, where d is the dimensionality of the full feature space X_d
    2. Determine the feature x^- that maximizes the criterion
       $x^- = \text{argmax} J(X_k - x)$ where $x \in X_k$
    3. Remove the feature x^- from the feature set:
       $X_{k-1} \coloneqq X_k - x^-; k \coloneqq k - 1$
    4. Terminate if k equals the number of desired features, if not, go to step 2

Dimensionality Reduction

- d features → k new features. k << d
- Principal Component Analysis(PCA)
  - PCA is linear transformation technique that is widely used for dimensionality reduction
  - PCA aims to find the principal components (directions of maximum variance) in high-dimensional data, and projects it to fewer dimensions
  - Example
    - x_1 and x_2 are the original feature axes, and PC1 and PC2 are the principal components of new axes



Dimensionality Reduction

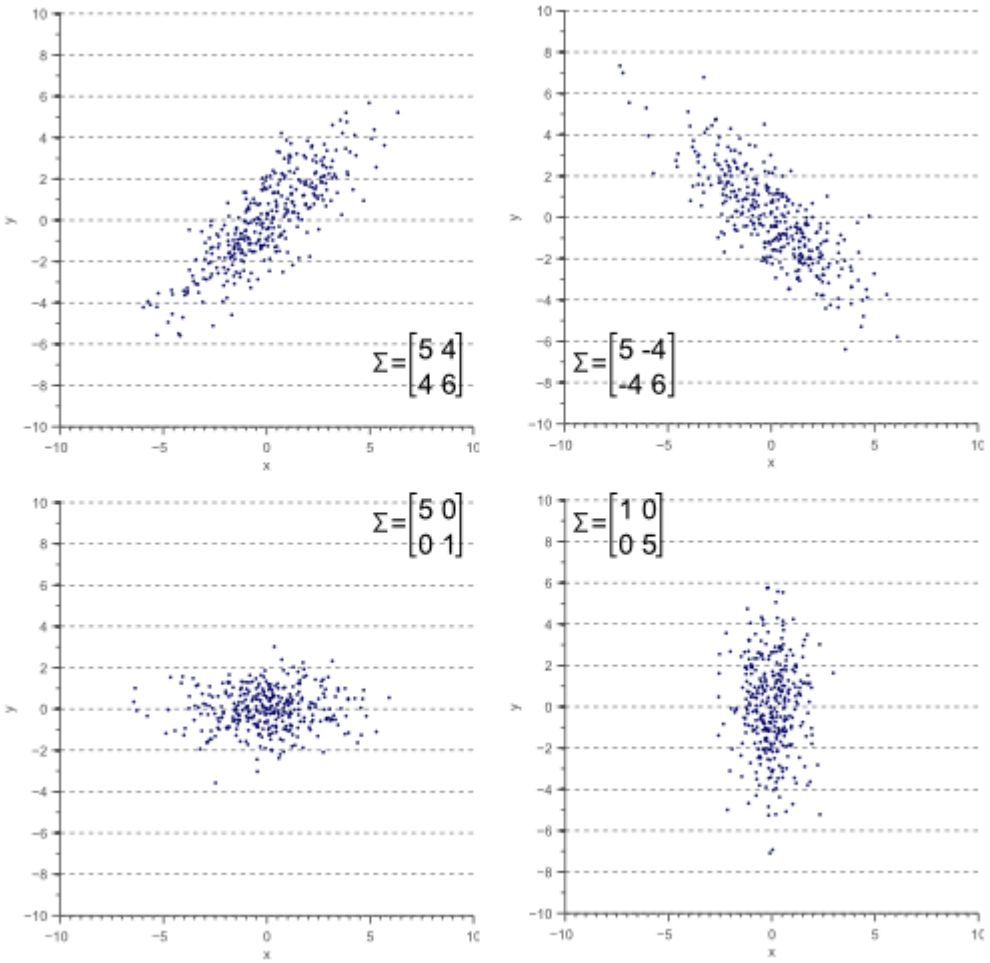
- Covariance
  - Covariance between two variables *x* and *y*

$$cov(x, y) = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{(n-1)}$$

- Covariance matrix of dataset *X*
  - $x = (x_1, x_2, \ldots, x_d)$

$$\sum = cov(X, X)$$

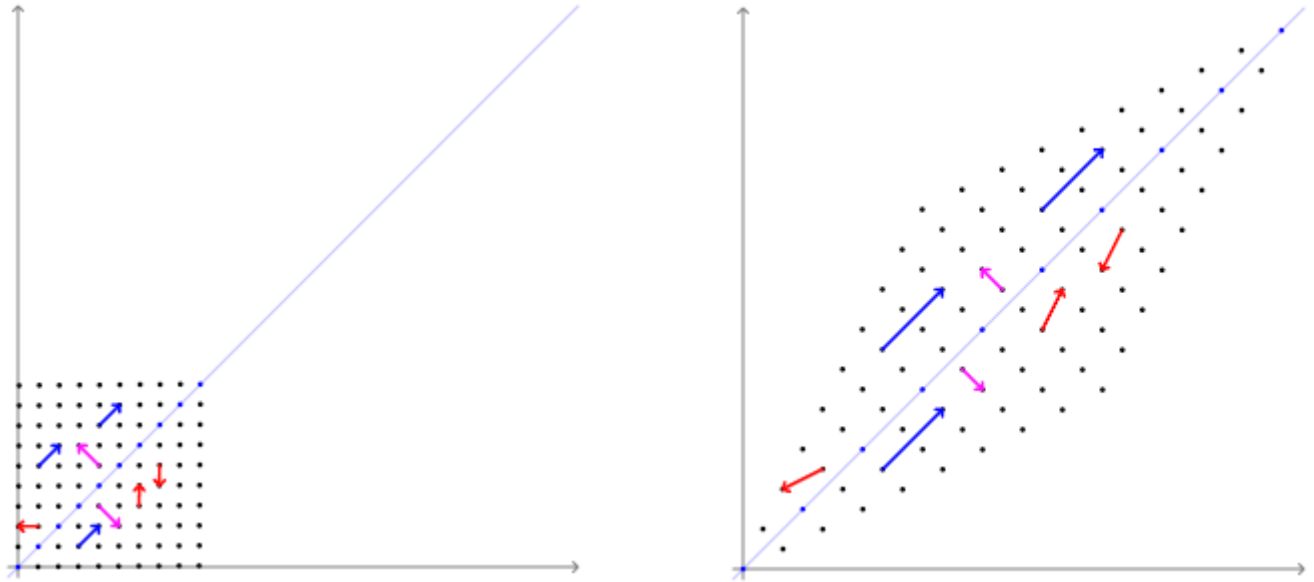
$$\begin{bmatrix} cov(x_1, x_1) & cov(x_1, x_2) \\ cov(x_2, x_1) & cov(x_2, x_2) \end{bmatrix}$$



Dimensionality Reduction

- Eigenvectors and eigenvalues
  - Unit vector *x* whose directions are not changed by the transformation *A*

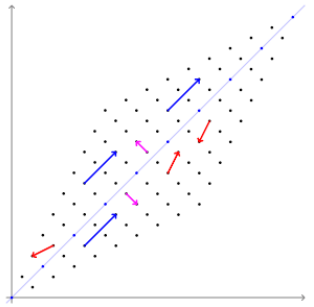
$$Ax = \lambda x$$



Dimensionality Reduction

- Example
  - Transformation $A = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$

$$x = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \rightarrow Ax = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$$



  - Let $Ax = \lambda x$
    then $(A - \lambda I)x = 0$

$$det(A - \lambda I) = det\left(\begin{bmatrix} 2-\lambda & 1 \\ 1 & 2-\lambda \end{bmatrix}\right) = 0$$

$$(2-\lambda)^2 - 1 = 0 \quad \rightarrow \quad \lambda = 1, 3$$

→ *Eigenvalues*

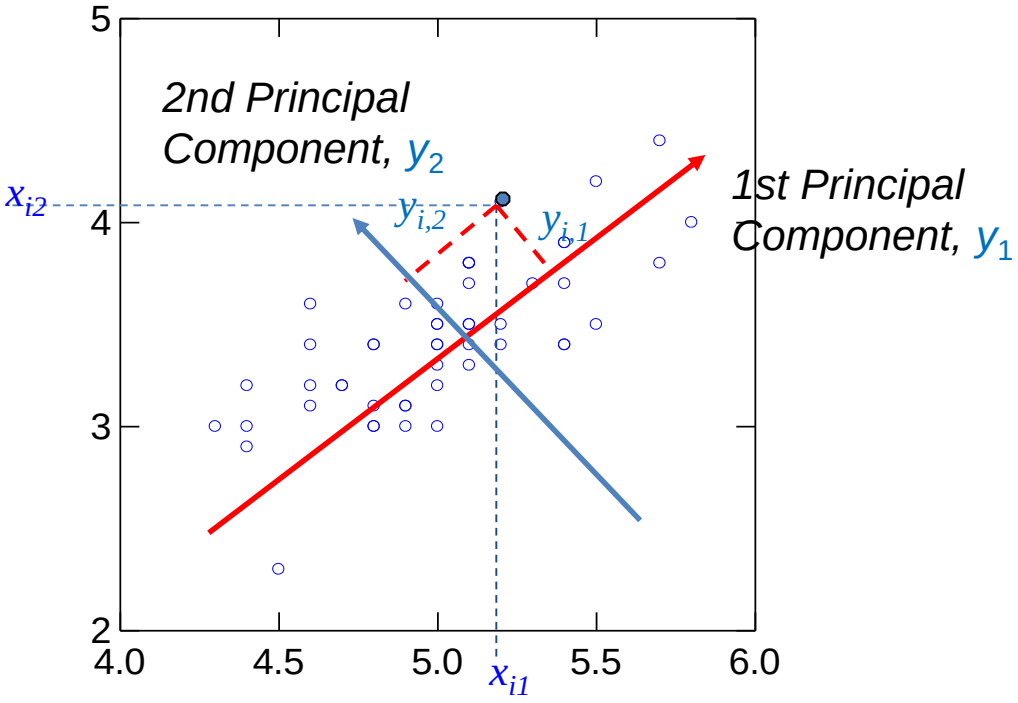
  - For $\lambda = 1$, $\begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = 1 \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \rightarrow \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$

    For $\lambda = 3$, $\begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = 3 \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \rightarrow \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$

→ *Eigenvectors*

Dimensionality Reduction

- Eigenvector of covariance matrix
  - The eigenvectors of covariance matrix $\sum$ correspond to principal components
  - The eigenvalues correspond to the variance explained by the principal components



Dimensionality Reduction with PCA

- PCA Algorithm

1. Standardize the d-dimensional dataset
2. Construct the covariance matrix $\sum (d \times d)$
3. Find its eigenvectors $(d \times d)$ and eigenvalues (d)
4. Select k eigenvectors that correspond to the k largest eigenvalues, where k is the dimensionality of the new feature subspace($k \leq d$)
5. Construct a projection matrix W $(d \times k)$ from the top k eigenvectors

$$W = [\, e_1, e_2, \ldots e_k \,]$$

6. Transform the d-dimensional data $\mathbf{x}$ to new k-dimensional data $\mathbf{x}'$ using the projection W

$$\mathbf{x}' = \mathbf{x}\, W$$
$$(k) = (d)(d \times k)$$

PCA Example

Data:

x_1	x_2
2.5	2.4
0.5	0.7
2.2	2.9
1.9	2.2
3.1	3.0
2.3	2.7
2.0	1.6
1.0	1.1
1.5	1.6
1.1	0.9

Zero mean:

x_1	x_2
0.69	0.49
-1.31	-1.21
0.39	0.99
0.09	0.29
1.29	1.09
0.49	0.79
0.19	-0.31
-0.81	-0.81
-0.31	-0.31
-0.71	-1.01

Covariance =

$$\begin{pmatrix} 0.67 & 0.62 \\ 0.62 & 0.72 \end{pmatrix}$$

PCA Example

Eigenvalues &
Eigenvectors :

1.28 0.05

$$\begin{pmatrix} -0.68 \\ -0.74 \end{pmatrix} \quad \begin{pmatrix} -0.74 \\ 0.68 \end{pmatrix}$$

New data:

x'_1	x'_2
-0.83	-0.18
1.79	0.15
-1.00	0.38
-0.28	0.13
-1.68	-0.21
-0.92	0.17
0.10	-0.35
1.15	0.05
0.44	0.02
1.23	-0.16

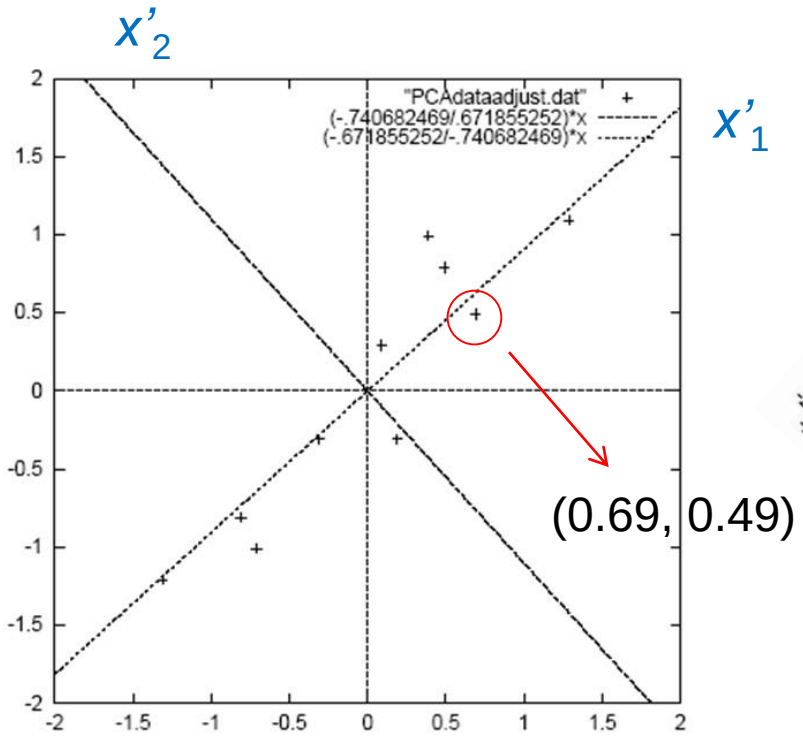
Reduced:

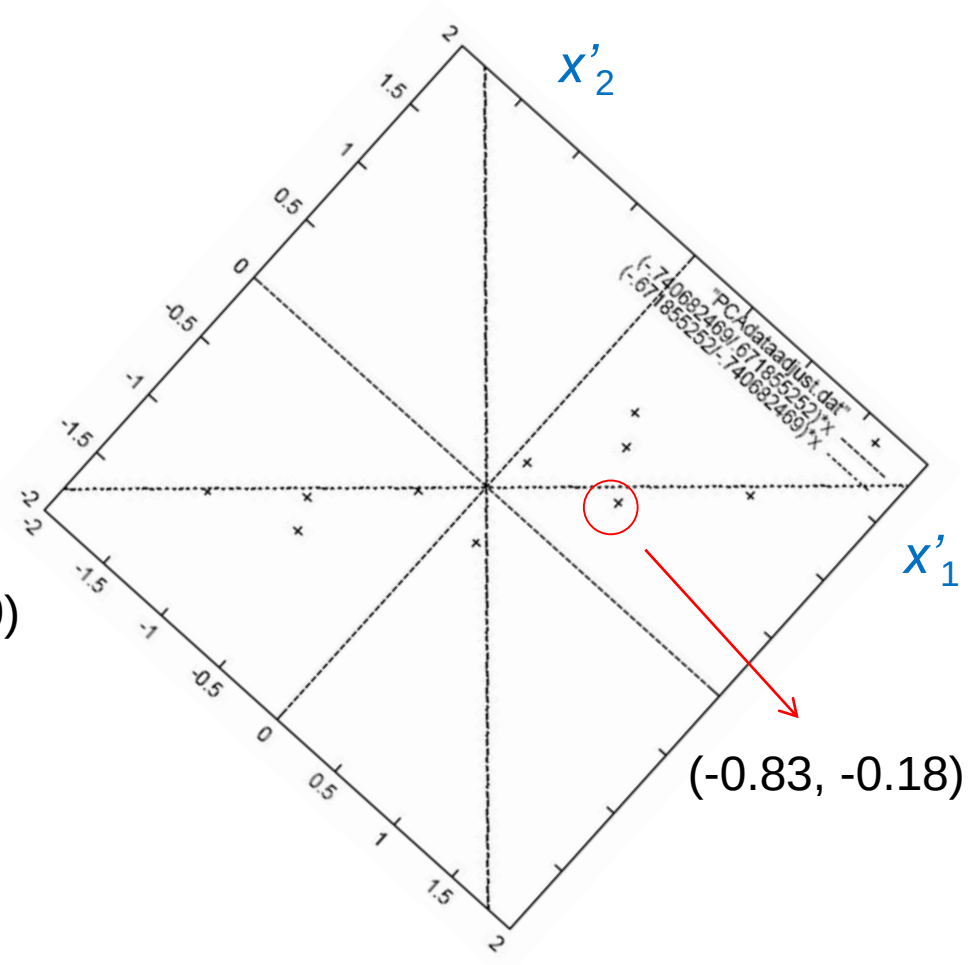
x'_1
-0.83
1.79
-1.00
-0.28
-1.68
-0.92
0.10
1.15
0.44
1.23

$$\mathbf{W} = [vector\ 1 \quad vector\ 2]$$

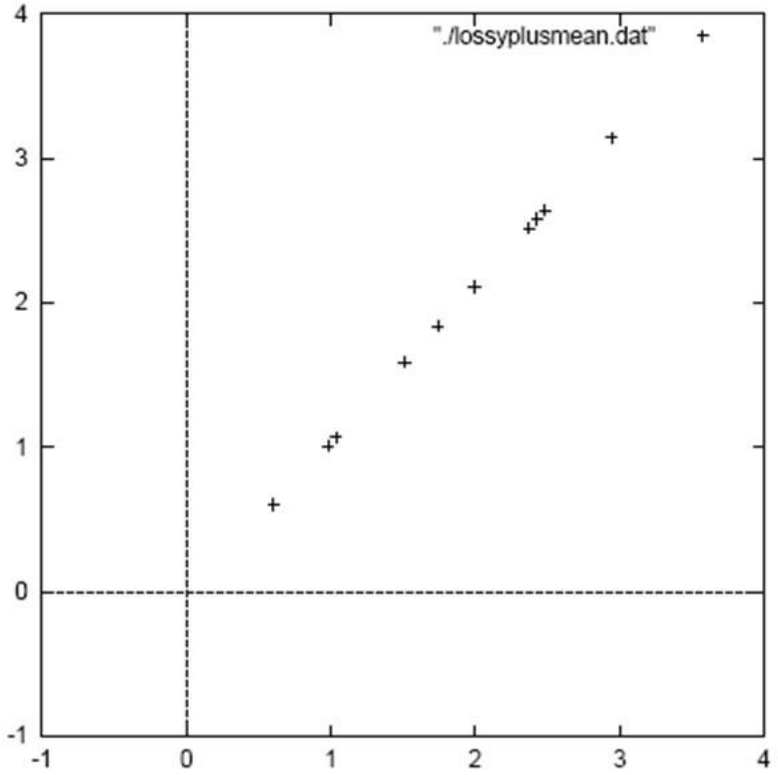
x'_1 = -0.68 x_1 - 0.74 x_2
x'_2 = -0.74 x_1 + 0.68 x_2

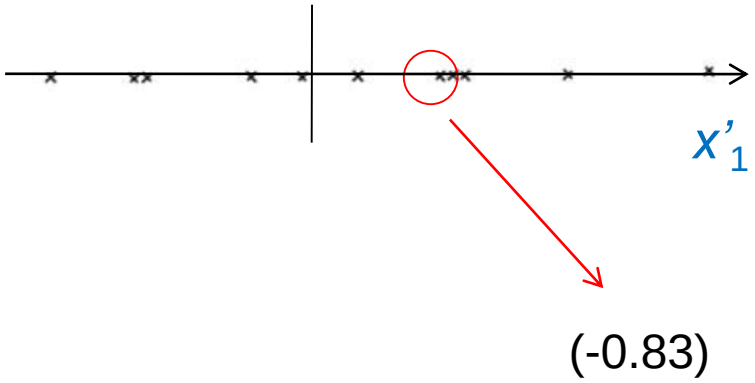
PCA Example


x'_2
x'_1
"PCAdataadjust.dat"
(-.740682469/.671855252)*x
(-.671855252/-.740682469)*x
(0.69, 0.49)


x'_2
x'_1
"PCAdataadjust.dat"
(-.740682469/.671855252)*x
(-.671855252/-.740682469)*x
(-0.83, -0.18)

PCA Example





PCA using Numpy

- X_train_std

```
print(X_train_std.shape)
print(X_train_std[:3])
```

```
(124, 13)
[[ 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]
```

- Covariance matrix - np.cov(X.T)

```
cov_mat = np.cov(X_train_std.T)
print(cov_mat.shape)
print(cov_mat)
```

```
(13, 13)
[[ 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]
 [ 0.17405351  0.08326463  1.00813008  0.46420355  0.29092834  0.18020384
   0.15537535  0.15918823 -0.00703776  0.20549146  0.00665422  0.02039019
   0.223495  ]
```

PCA using Numpy

- Eigenvalues and eigenvectors of covariance matrix - np.linalg.eig(A)

```
eigen_vals, eigen_vecs = np.linalg.eig(cov_mat)
```

```
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]
 [-2.54515927e-02  2.44564761e-01  6.77775667e-01 -1.08977111e-01
   1.60834991e-01 -3.89128239e-04 -1.32328045e-01 -1.77578177e-01
  -3.82496856e-01  1.42747511e-01  4.61583035e-01  1.67924873e-02
  -1.15542548e-01]
```

PCA using Numpy

- Projection to 2 dimension using PC1, PC2

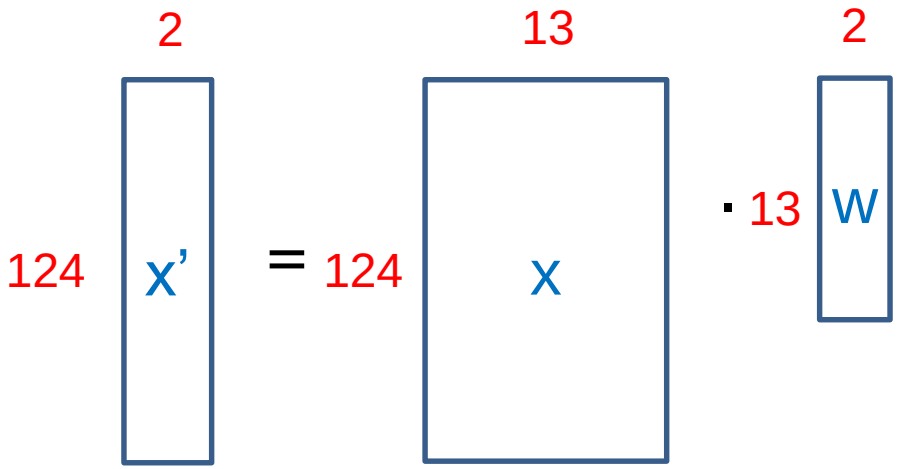
```
w = eigen_vecs[:, [0, 1]]
print(w)
```

```
[[-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]]
```

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

```
print(X_train_std[0])
print(X_train_pca[0])
```

```
[ 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]
[2.38299011 0.45458499]
```



PCA using Scikit-Learn

	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
0	1	14.23	1.71	2.43	15.6	127	2.80	3.06	0.28	2.29	5.64	1.04	3.92	1065
1	1	13.20	1.78	2.14	11.2	100	2.65	2.76	0.26	1.28	4.38	1.05	3.40	1050
2	1	13.16	2.36	2.67	18.6	101	2.80	3.24	0.30	2.81	5.68	1.03	3.17	1185

- PCA

```
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]
```

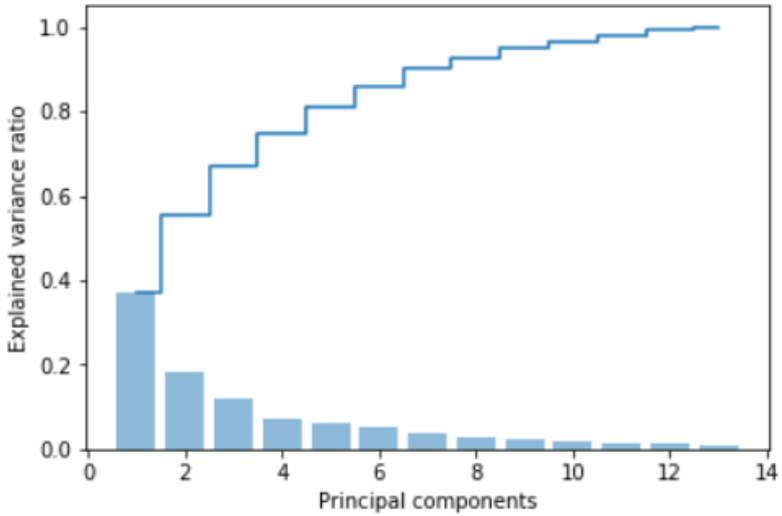
```
array([ 2.38299011,  0.45458499, -0.22703207,  0.57988399, -0.57994169,
       -1.73317476,  0.70180475,  0.21617248, -0.23666876, -0.16548767,
        0.29726982,  0.23489704, -0.40161994])
```

PCA using Scikit-Learn

- Explained variance ratio
  - The ratio between the variance of that principal component and the total variance

```
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])
```



PCA using Scikit-Learn

- Dimensionality reduction to 2 PC

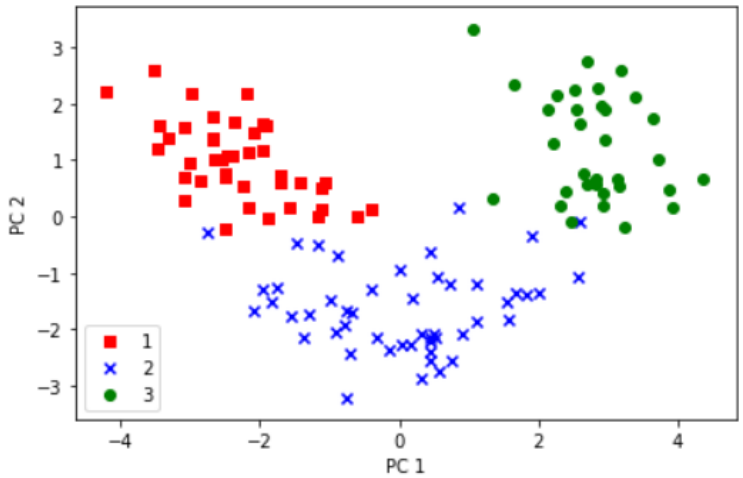
```
pca = PCA(n_components=2)
X_train_pca = pca.fit_transform(X_train_std)
X_test_pca = pca.transform(X_test_std)
```

```
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])
```



PCA using Scikit-Learn

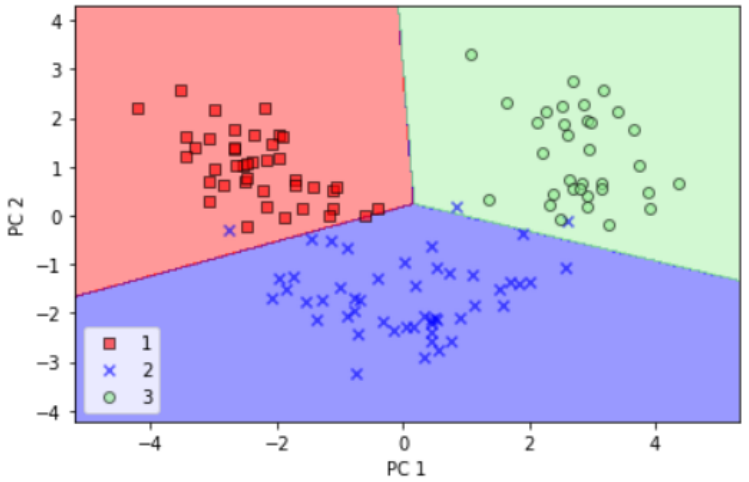
- Logistic regression with 2 dimensional data

```
from sklearn.linear_model import LogisticRegression

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

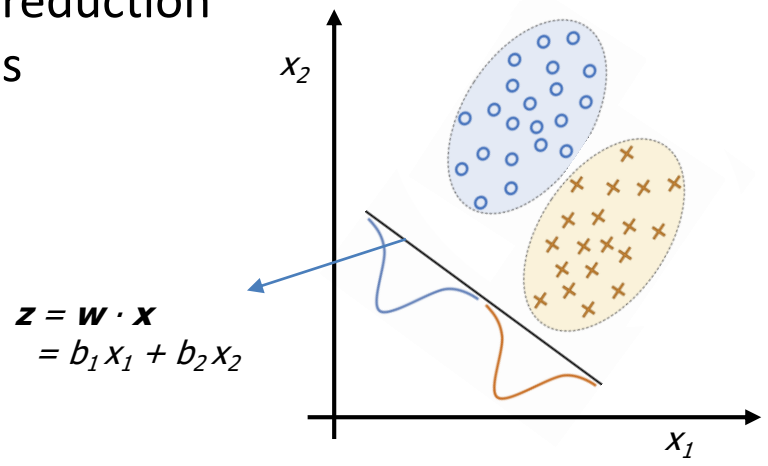
```
acc = lr.score(X_test_pca, y_test)
print("Test accuracy : %.4f" % acc)
```

```
Test accuracy : 0.9259
```



Dimensionality Reduction with LDA

- Linear discriminant analysis(LDA)
  - LDA finds a linear combination of features that separates classes well, so that it maximize the between-class scatter and minimize the within-class scatter
  - Assumption
    - Each class has normal distribution
    - Each class has same covariance structure
  - It can be used for dimensionality reduction by projection of data to that axis



Dimensionality Reduction with LDA

- Maximum separation
  - Original data in 2 classes have means $m_1, \quad m_2$

    → Projected data have means $w^T m_1, \quad w^T m_2$

    variances $\sum (w^T x - w^T m_1)^2, \ \sum (w^T x - w^T m_2)^2$
  - Between class distance

    $$(w^T m_1 - w^T m_2)^2 = w^T (m_1 - m_2)(m_1 - m_2)^T w = w^T S_B w$$
  - Within class variance

    $$\sum (w^T x - w^T m_1)^2 + \sum (w^T x - w^T m_2)^2$$
    $$= w^T \left(\sum (x - m_1)(x - m_1)^T + \sum (x - m_2)(x - m_2)^T \right) w = w^T S_w w$$
  - Maximize

    $$J(w) = \frac{Between\ class\ distance}{Within\ class\ variance} = \frac{w^T S_B w}{w^T S_w w}$$

Dimensionality Reduction with LDA

- Maximize $J(w) = \frac{w^T S_B w}{w^T S_w w}$

$$\frac{dJ(w)}{dw} = \frac{2S_B w}{(w^T S_w w)} - \frac{w^T S_B w \cdot 2S_w w}{(w^T S_w w)^2} = 0$$

$$2S_B w \cdot w^T S_w w \; - w^T S_B w \cdot 2S_w w = 0$$

$$S_w{}^{-1} S_B w = \frac{w^T S_B w}{w^T S_w w} w$$

$$S_w{}^{-1} S_B w = J(w) w$$

- ∴ w is eigenvector of $S_w{}^{-1} S_B$ ($Ax = \lambda x$)

- LDA using scikit-learn

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA

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

Dimensionality Reduction with LDA

- PCA vs. LDA
  - PCA is unsupervised dimensionality reduction
    - It does not require any class labels
    - PCA tries to find the axes where the data is most spread
  - LDA is supervised dimensionality reduction
    - It is used for datasets with class labels
    - LDA tries to separate classes
  - PCA performs better in case where number of samples per class is less, LDA performs better with large dataset having multiple classes
  - LDA is inappropriate when the classes do not have normal distribution, or the classes have different covariance structures