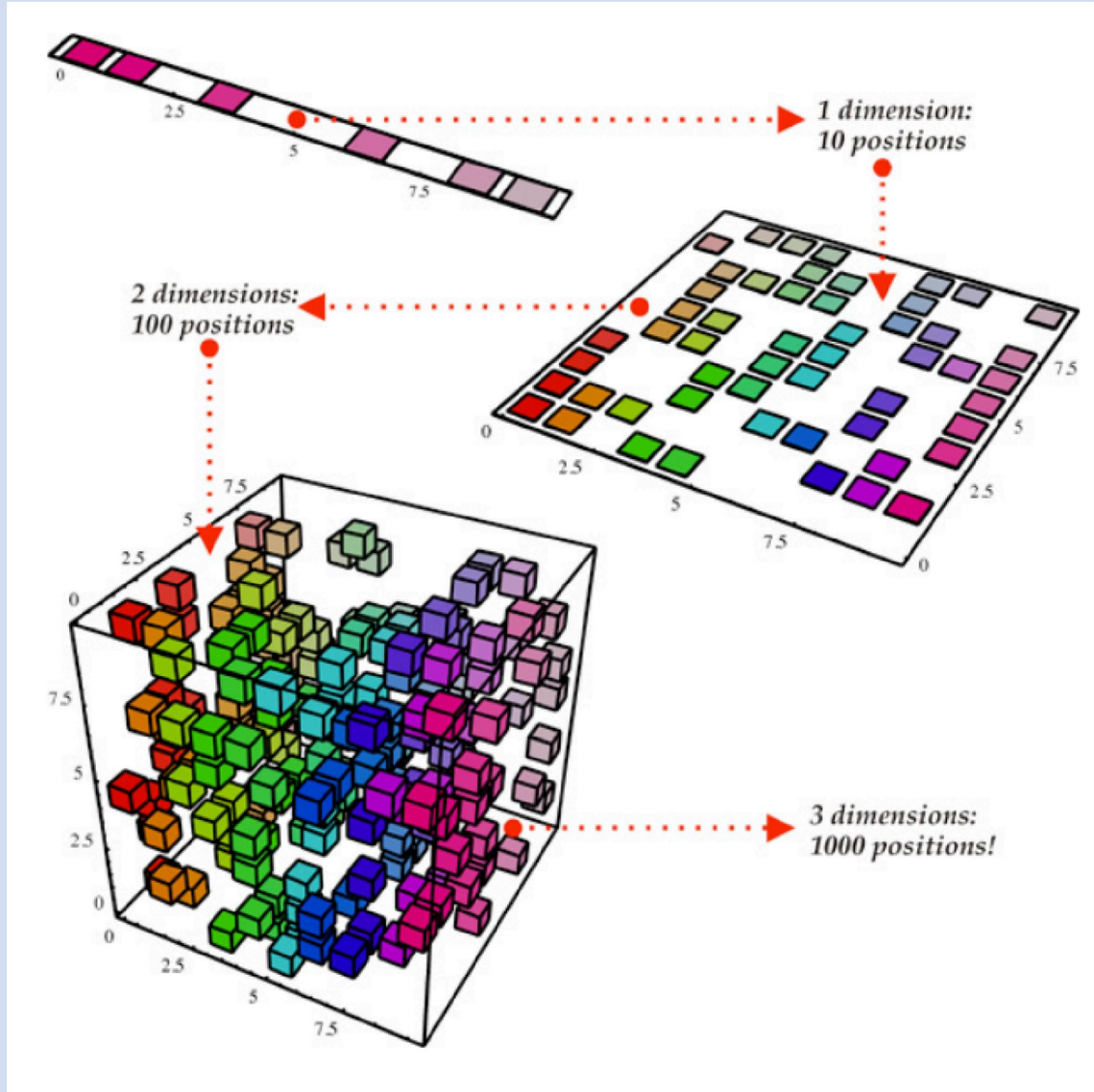
Data Processing and Dimensionality Reduction

Dr. Tran Anh Tuan

INTRODUCTION: What is Dimensionality Reduction?

Here are some of the benefits of applying dimensionality reduction to a dataset:

1. Space required to store the data is reduced as the number of dimensions comes down
2. Less dimensions lead to less computation/training time
3. Some algorithms do not perform well when we have a large dimensions. So reducing these dimensions needs to happen for the algorithm to be useful



INTRODUCTION:

Common Dimensionality Reduction Techniques

Dimensionality reduction can be done in two different ways:

1. By only keeping the most relevant variables from the original dataset (this technique is called feature selection)
2. By finding a smaller set of new variables, each being a combination of the input variables, containing basically the same information as the input variables (this technique is called dimensionality reduction)

1 Missing Value Ratio
2 Low Variance Filter
3 High Correlation Filter
4 Random Forest
5 Backward Feature Elimination
6 Forward Feature Selection
7 Factor Analysis
8 Principal Component Analysis
9 Independent Component Analysis

INTRODUCTION: Dimensionality Reduction

1. Missing Value Ratio

Data columns with too many missing values are unlikely to carry much useful information. Thus data columns with number of missing values greater than a given threshold can be removed. The higher the threshold, the more aggressive the reduction

2. Low Variance Filter

Similarly to the previous technique, data columns with little changes in the data carry little information. Thus all data columns with variance lower than a given threshold are removed. A word of caution: variance is range dependent; therefore normalization is required before applying this technique.

3. High Correlation filter

Data columns with very similar trends are also likely to carry very similar information. In this case, only one of them will suffice to feed the machine learning model. Here we calculate the correlation coefficient between numerical columns and between nominal columns as the Pearson's Product Moment Coefficient and the Pearson's chi square value respectively. Pairs of columns with correlation coefficient higher than a threshold are reduced to only one. A word of caution: correlation is scale sensitive; therefore column normalization is required for a meaningful correlation comparison.

$$\rho_{X,Y} = \frac{\operatorname{cov}(X,Y)}{\sigma_X \sigma_Y}$$

where:

- cov is the covariance
- σ_X is the standard deviation of X
- σ_Y is the standard deviation of Y

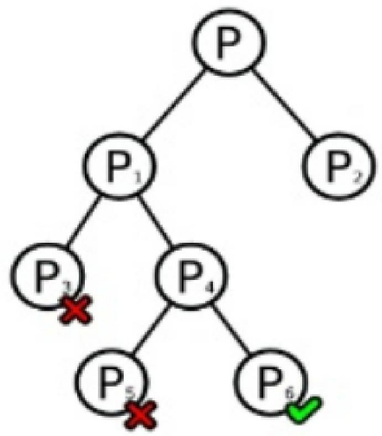
INTRODUCTION: Dimensionality Reduction

4. **Random Forest**

Decision Tree Ensembles, also referred to as random forests, are useful for feature selection in addition to being effective classifiers. One approach to dimensionality reduction is to generate a large and carefully constructed set of trees against a target attribute and then use each attribute's usage statistics to find the most informative subset of features

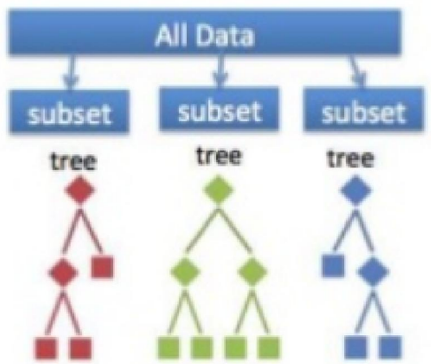
Decision Tree

- Decision tree builds classification models in the form of a tree structure.
- It breaks down a dataset into smaller and smaller subsets.



Random Forest

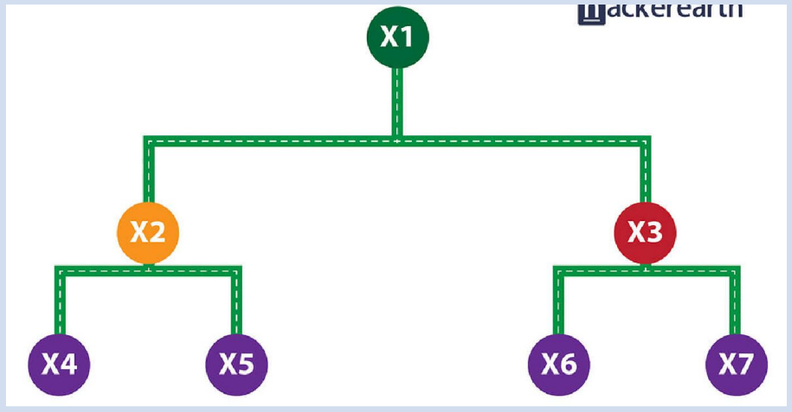
- Random Forest is an ensemble classifier made using many decision tree models.
- Ensemble models combine the results from different models.

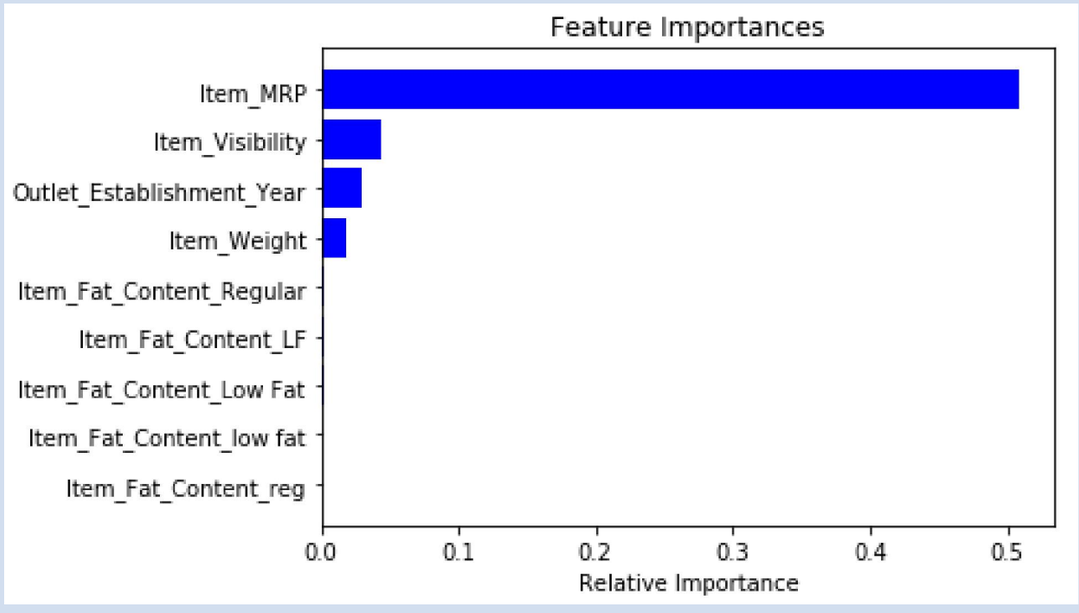


INTRODUCTION: Dimensionality Reduction

4. **Random Forest**

- Random forest consists of a number of decision trees. Every node in the decision trees is a condition on a single feature, designed to split the dataset into two so that similar response values end up in the same set. The measure based on which the (locally) optimal condition is chosen is called impurity.
- For classification, it is typically either Gini impurity or information gain/entropy and for regression trees it is variance. Thus when training a tree, it can be computed how much each feature decreases the weighted impurity in a tree. For a forest, the impurity decrease from each feature can be averaged and the features are ranked according to this measure.





INTRODUCTION: Dimensionality Reduction

5. Backward Feature Elimination

Follow the below steps to understand and use the 'Backward Feature Elimination' technique:

- We first take all the n variables present in our dataset and train the model using them
- We then calculate the performance of the model
- Now, we compute the performance of the model after eliminating each variable (n times), i.e., we drop one variable every time and train the model on the remaining n-1 variables
- We identify the variable whose removal has produced the smallest (or no) change in the performance of the model, and then drop that variable
- Repeat this process until no variable can be dropped

6. Forward Feature Selection

Instead of eliminating features, we try to find the best features which improve the performance of the model. This technique works as follows:

- We start with a single feature. Essentially, we train the model n number of times using each feature separately
- The variable giving the best performance is selected as the starting variable
- Then we repeat this process and add one variable at a time. The variable that produces the highest increase in performance is retained
- We repeat this process until no significant improvement is seen in the model's performance

INTRODUCTION: Dimensionality Reduction

SVM-RFE

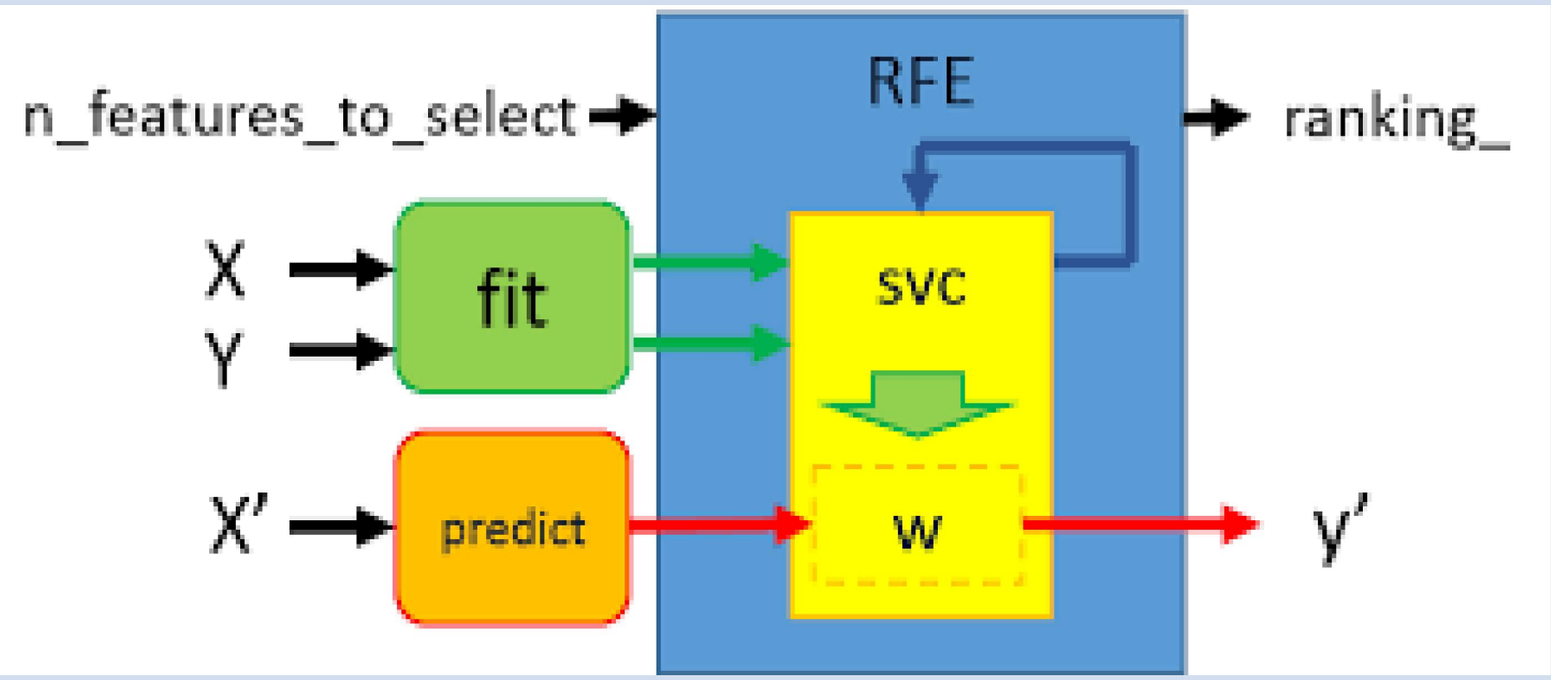
Linear binary classifier decision function

$$f(x_1,\ldots,x_N)=\sum_{i=1}^{N} w_i x_i + b$$

$$|w_i| = \text{score of variable } x_i$$

Recursive Feature Elimination (SVM-RFE)

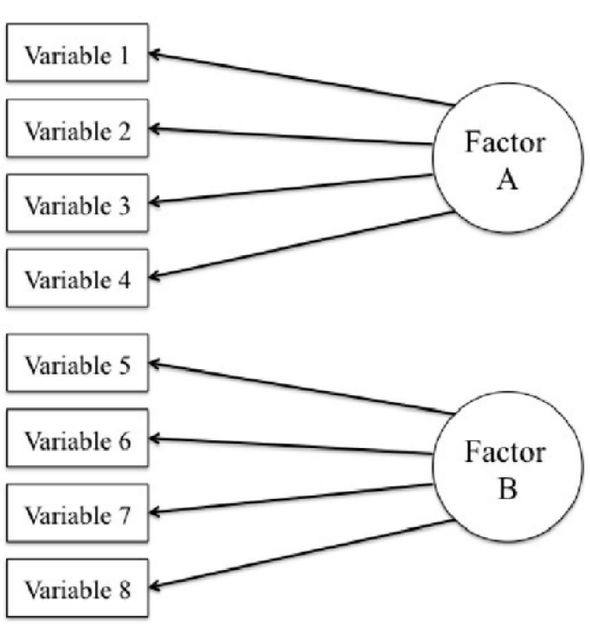
- at each iteration:
  1) eliminate threshold% of variables with lower score
  2) recompute scores of remaining variables

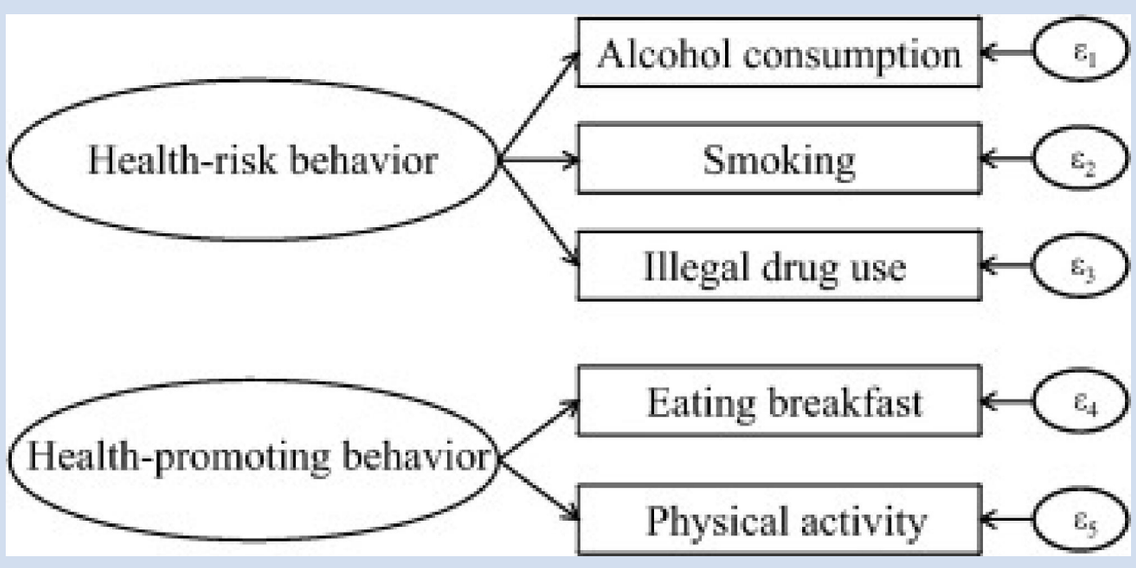


INTRODUCTION: Dimensionality Reduction

7. Factor Analysis

- Suppose we have two variables: Income and Education. These variables will potentially have a high correlation as people with a higher education level tend to have significantly higher income, and vice versa.
- In the Factor Analysis technique, variables are grouped by their correlations, i.e., all variables in a particular group will have a high correlation among themselves, but a low correlation with variables of other group(s). Here, each group is known as a factor. These factors are small in number as compared to the original dimensions of the data. However, these factors are difficult to observe.



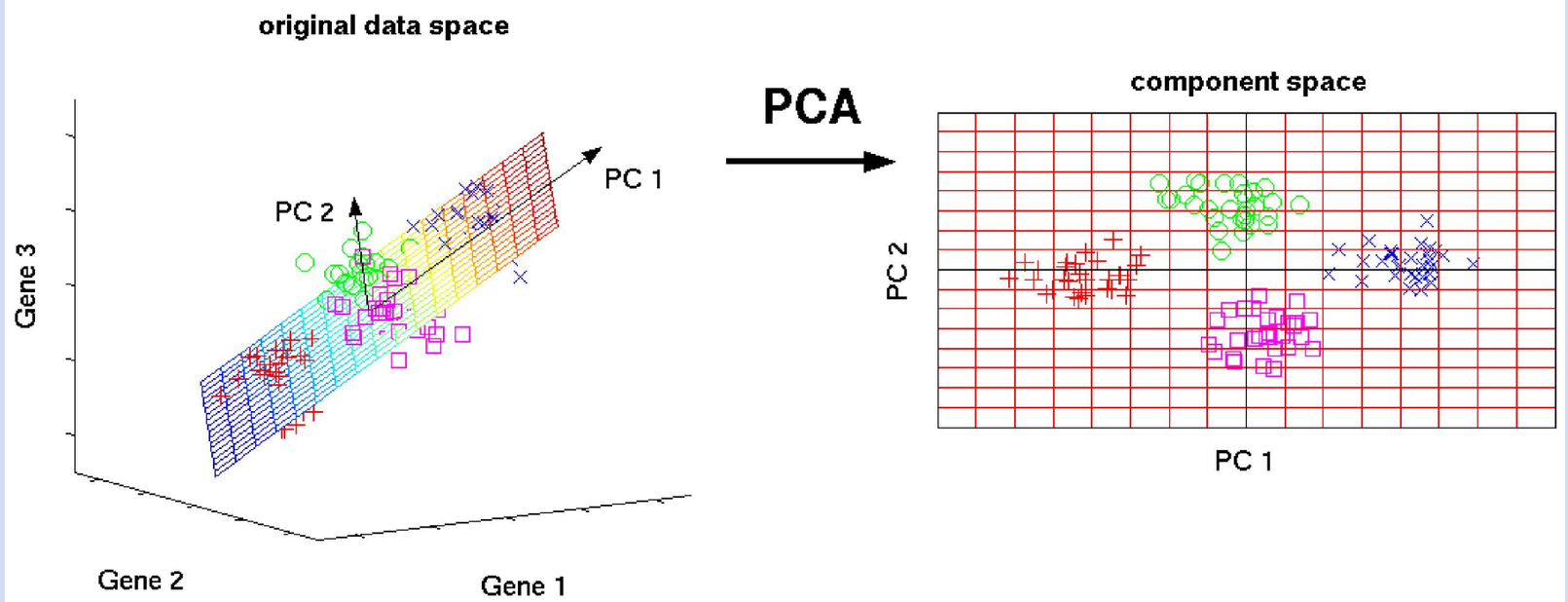


INTRODUCTION: Dimensionality Reduction

8. Principal Component Analysis (PCA)

PCA is a technique which helps us in extracting a new set of variables from an existing large set of variables. These newly extracted variables are called Principal Components. Below are some of the key points you should know about PCA before proceeding further:

- A principal component is a linear combination of the original variables
- Principal components are extracted in such a way that the first principal component explains maximum variance in the dataset



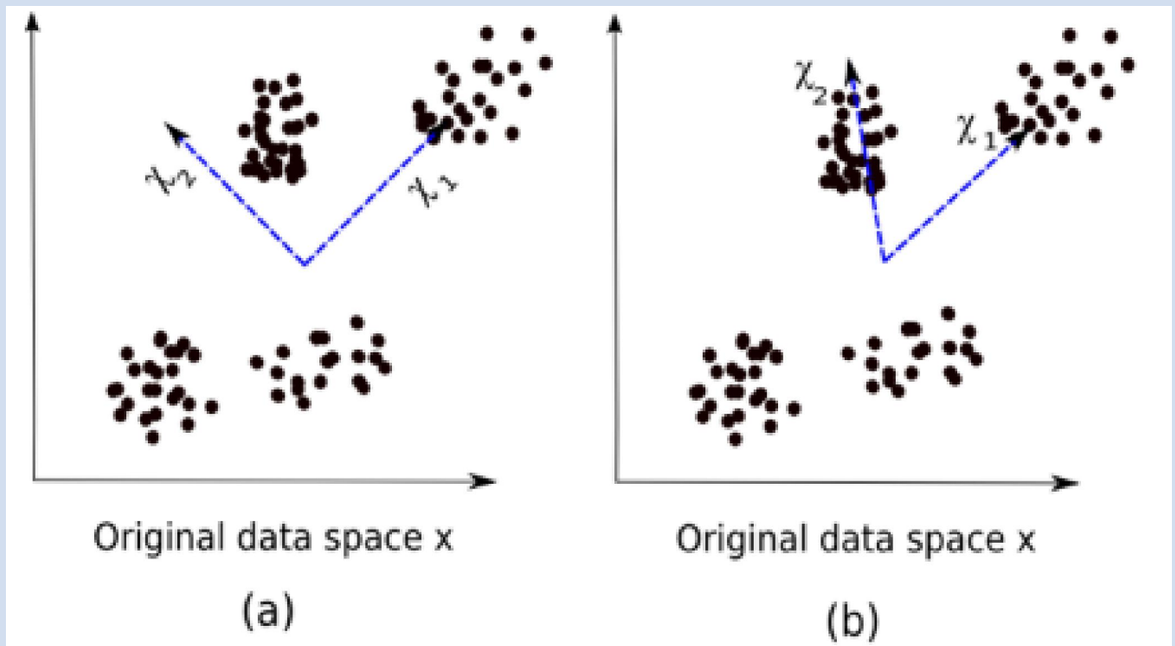
A Summary of the PCA Approach

- Standardize the data.
- Obtain the Eigenvectors and Eigenvalues from the covariance matrix or correlation matrix, or perform Singular Vector Decomposition.
- Sort eigenvalues in descending order and choose the k eigenvectors that correspond to the k largest eigenvalues where k is the number of dimensions of the new feature subspace ($k \leq d$).
- Construct the projection matrix $\mathbf{W}$ from the selected k eigenvectors.
- Transform the original dataset $\mathbf{X}$ via $\mathbf{W}$ to obtain a k-dimensional feature subspace $\mathbf{Y}$.

INTRODUCTION: Dimensionality Reduction

9. Independent Component Analysis

- Independent Component Analysis (ICA) is based on information-theory and is also one of the most widely used dimensionality reduction techniques. The major difference between PCA and ICA is that PCA looks for uncorrelated factors while ICA looks for independent factors.
- This algorithm assumes that the given variables are linear mixtures of some unknown latent variables. It also assumes that these latent variables are mutually independent, i.e., they are not dependent on other variables and hence they are called the independent components of the observed data.



(a) represents the PCA results while image
(b) represents the ICA results on the same dataset.

INTRODUCTION: Dimensionality Reduction

Summarizing the LDA approach in 5 steps

Listed below are the 5 general steps for performing a linear discriminant analysis; we will explore them in more detail in the following sections.

1. Compute the d-dimensional mean vectors for the different classes from the dataset.
2. Compute the scatter matrices (in-between-class and within-class scatter matrix).
3. Compute the eigenvectors ($\boldsymbol{e}_1, \boldsymbol{e}_2, \ldots, \boldsymbol{e}_d$) and corresponding eigenvalues ($\boldsymbol{\lambda}_1, \boldsymbol{\lambda}_2, \ldots, \boldsymbol{\lambda}_d$) for the scatter matrices.
4. Sort the eigenvectors by decreasing eigenvalues and choose k eigenvectors with the largest eigenvalues to form a $d \times k$ dimensional matrix $\boldsymbol{W}$ (where every column represents an eigenvector).
5. Use this $d \times k$ eigenvector matrix to transform the samples onto the new subspace. This can be summarized by the matrix multiplication: $\boldsymbol{Y} = \boldsymbol{X} \times \boldsymbol{W}$ (where $\boldsymbol{X}$ is a $n \times d$-dimensional matrix representing the n samples, and $\boldsymbol{y}$ are the transformed $n \times k$-dimensional samples in the new subspace).

INTRODUCTION: Dimensionality Reduction

The **within-class scatter** matrix S_W is computed by the following equation:

$$S_W = \sum_{i=1}^{c} S_i$$

where

$$S_i = \sum_{\boldsymbol{x} \in D_i}^{n} (\boldsymbol{x} - \boldsymbol{m}_i)\,(\boldsymbol{x} - \boldsymbol{m}_i)^T$$

(scatter matrix for every class)

and $\boldsymbol{m}_i$ is the mean vector

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

The **between-class scatter** matrix S_B is computed by the following equation:

$$S_B = \sum_{i=1}^{c} N_i (\boldsymbol{m}_i - \boldsymbol{m})(\boldsymbol{m}_i - \boldsymbol{m})^T$$

where

$\boldsymbol{m}$ is the overall mean, and $\boldsymbol{m}_i$ and N_i are the sample mean and sizes of the respective classes.

A quick check that the eigenvector-eigenvalue calculation is correct and satisfy the equation:

$$\boldsymbol{A}\boldsymbol{v} = \lambda \boldsymbol{v}$$

where

$\boldsymbol{A} = S_W^{-1} S_B$

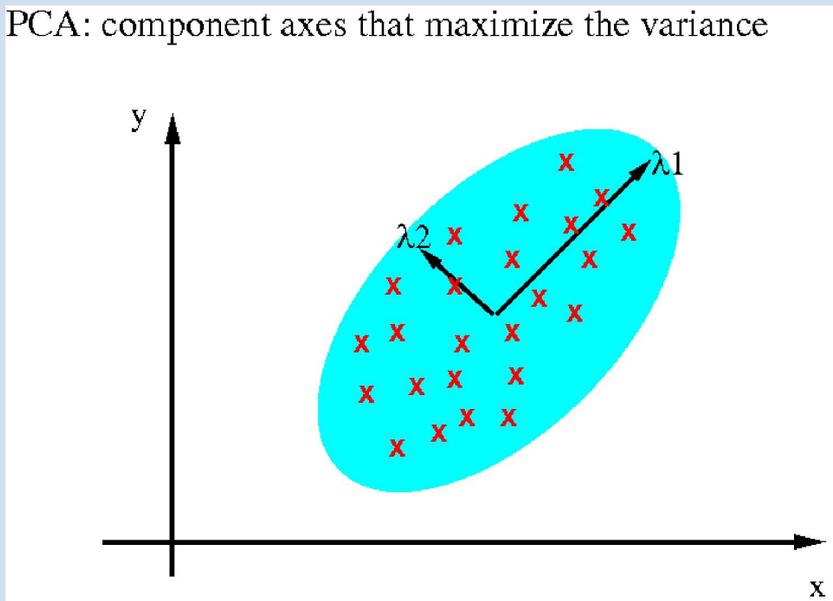
$\boldsymbol{v} =$ Eigenvector

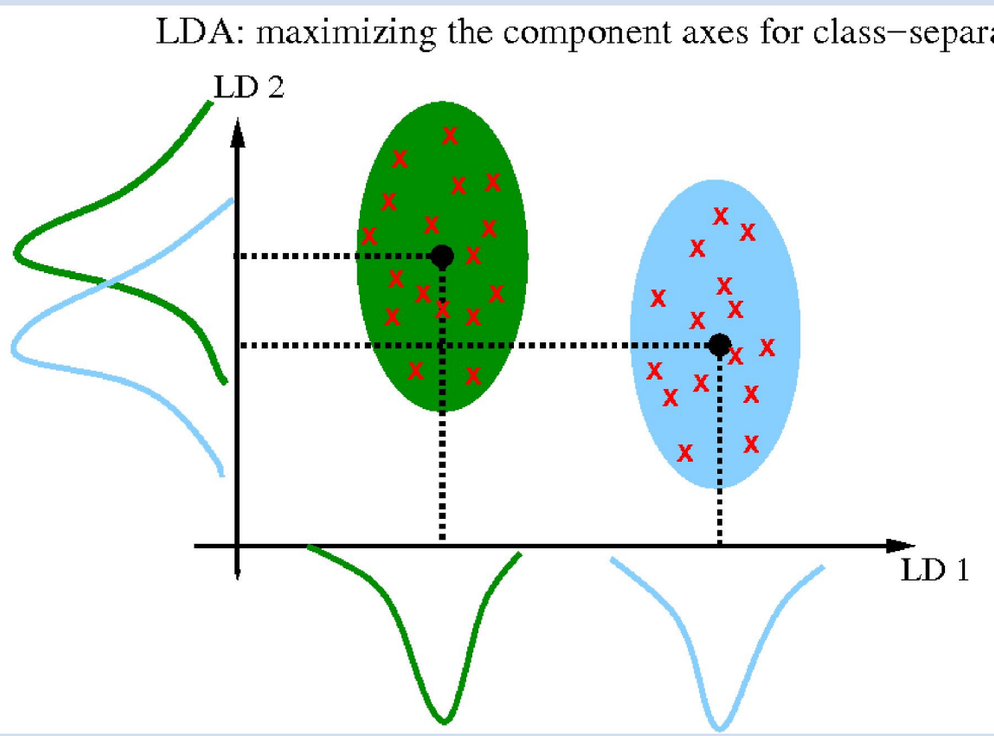
$\lambda =$ Eigenvalue

➔ solve the generalized eigenvalue problem for the matrix $S^{-1}{}_W S_B$ to obtain the linear discriminants.

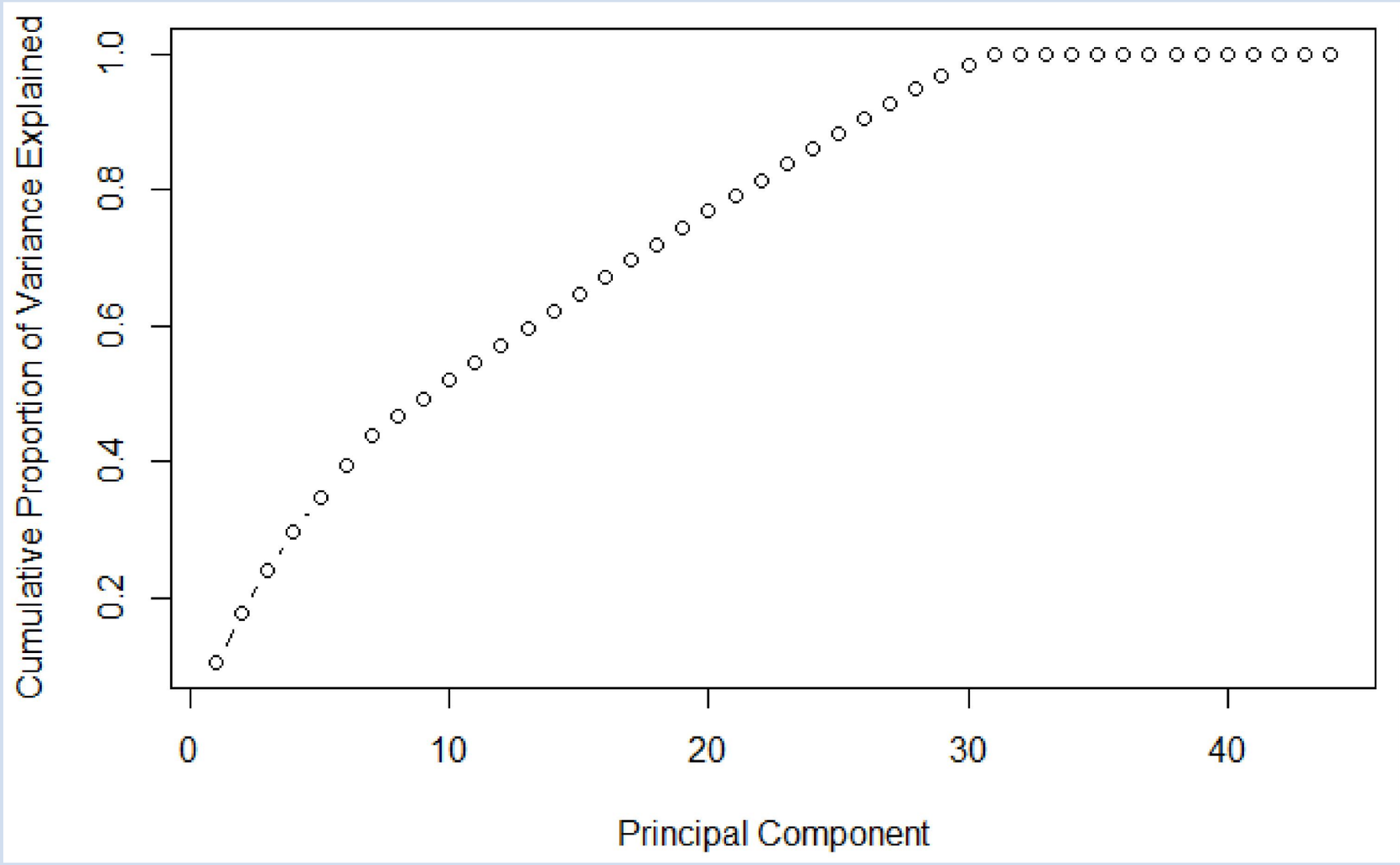
INTRODUCTION: Dimensionality Reduction

PCA Vs. LDA
Both Linear Discriminant Analysis (LDA) and PCA are linear transformation methods. PCA yields the directions (principal components) that maximize the variance of the data, whereas LDA also aims to find the directions that maximize the separation (or discrimination) between different classes, which can be useful in pattern classification problem (PCA “ignores” class labels).





INTRODUCTION: Dimensionality Reduction



INTRODUCTION: Dimensionality Reduction

- Compare those techniques on the smaller data set of the 2009 KDD challenge in terms of reduction ratio, degrading accuracy, and speed. The final accuracy and its degradation depend, of course, on the model selected for the analysis. Thus, the compromise between reduction ratio and final accuracy is optimized against a bag of three specific models: decision tree, neural networks, and Naive Bayes.

Dimensionality Reduction	Reduction Rate	Accuracy on validation set	Best Threshold	AuC	Notes
Baseline	0%	73%	–	81%	Baseline models are using all input features
Missing Values Ratio	71%	76%	0.4	82%	–
Low Variance Filter	73%	82%	0.03	82%	Only for numerical columns
High Correlation Filter	74%	79%	0.2	82%	No correlation available between numerical and nominal columns
PCA	62%	74%	–	72%	Only for numerical columns
Random Forrest / Ensemble Trees	**86%**	**76%**	–	**82%**	–
Backward Feature Elimination + missing values ratio	99%	94%	–	78%	Backward Feature Elimination and Forward Feature Construction are **prohibitively slow** on high dimensional data sets. It becomes practical to use them, only if following other dimensionality reduction techniques, like here the one based on the number of missing values.
Forward Feature Construction + missing values ratio	91%	83%	–	63%	

Practice Sections

```
# import required libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# read the data
train=pd.read_csv("../InputData/Train_UWu5bXk.csv")

# checking the percentage of missing values in each variable
print(train.isnull().sum()/len(train)*100)

# saving missing values in a variable
a = train.isnull().sum()/len(train)*100
# saving column names in a variable
variables = train.columns
variable = [ ]
for i in range(0,12):
    if a[i]<=20:   #setting the threshold as 20%
        variable.append(variables[i])

print("\n".join(variable))
```

```
Item_Identifier              0.000000
Item_Weight                 17.165317
Item_Fat_Content             0.000000
Item_Visibility              0.000000
Item_Type                    0.000000
Item_MRP                     0.000000
Outlet_Identifier            0.000000
Outlet_Establishment_Year    0.000000
Outlet_Size                 28.276428
Outlet_Location_Type         0.000000
Outlet_Type                  0.000000
Item_Outlet_Sales            0.000000
dtype: float64
```

```
Item_Identifier
Item_Weight
Item_Fat_Content
Item_Visibility
Item_Type
Item_MRP
Outlet_Identifier
Outlet_Establishment_Year
Outlet_Location_Type
Outlet_Type
Item_Outlet_Sales
```

Practice Sections

```
print(np.nanmedian(train['Item_Weight']))
print(train['Outlet_Size'].mode()[0])
train['Item_Weight'] = train['Item_Weight'].fillna(np.nanmedian(train['Item_Weight']))
train['Outlet_Size'] = train['Outlet_Size'].fillna(train['Outlet_Size'].mode()[0])
print("train after fill nan: ")
print(train.isnull().sum()/len(train)*100)
print(train.var())

numeric = train[['Item_Weight', 'Item_Visibility', 'Item_MRP', 'Outlet_Establishment_Year']]
var = numeric.var()
print("var:")
print(var)

numeric = numeric.columns
variable = [ ]
for i in range(0,len(var)):
    if var[i]>=10:   #setting the threshold as 10%
        variable.append(numeric[i])

print(variable)
```

```
12.6
Medium
train after fill nan:
Item_Identifier              0.0
Item_Weight                  0.0
Item_Fat_Content             0.0
Item_Visibility              0.0
Item_Type                    0.0
Item_MRP                     0.0
Outlet_Identifier            0.0
Outlet_Establishment_Year    0.0
Outlet_Size                  0.0
Outlet_Location_Type         0.0
Outlet_Type                  0.0
Item_Outlet_Sales            0.0
dtype: float64
```

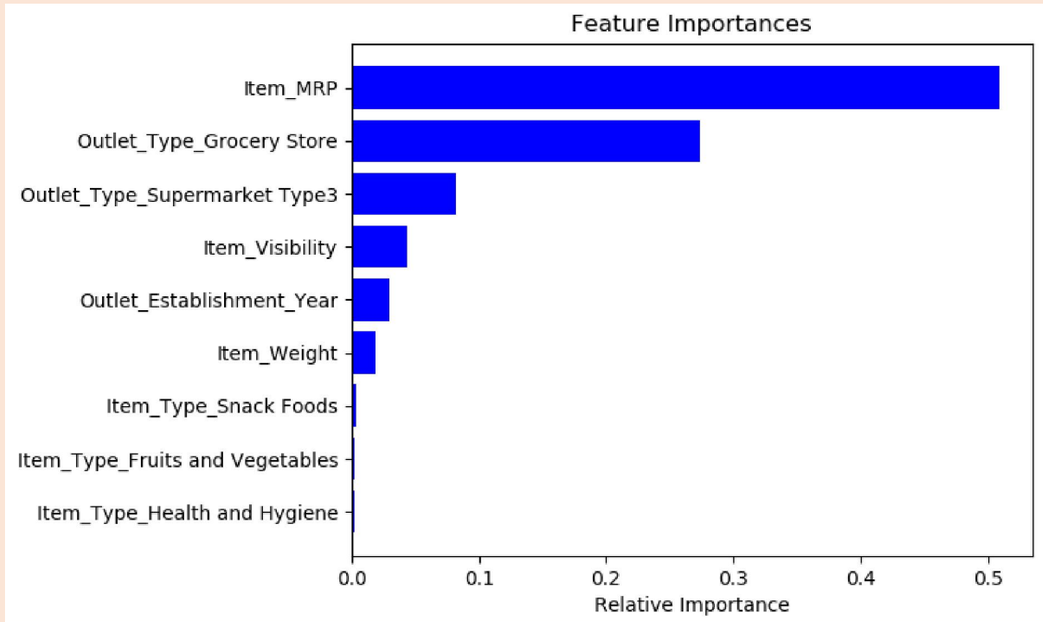
```
Item_Weight                  1.786956e+01
Item_Visibility              2.662335e-03
Item_MRP                     3.878184e+03
Outlet_Establishment_Year    7.008637e+01
Item_Outlet_Sales            2.912141e+06
dtype: float64
```

```
var:
Item_Weight                  17.869561
Item_Visibility               0.002662
Item_MRP                   3878.183909
Outlet_Establishment_Year    70.086372
dtype: float64
['Item_Weight', 'Item_MRP', 'Outlet_Establishment_Year']
```

Practice Sections

```
df=train.drop('Item_Outlet_Sales', 1)
corrMatrix = df.corr()
print(corrMatrix.to_string())
```

```
from sklearn.ensemble import RandomForestRegressor
df=df.drop(['Item_Identifier', 'Outlet_Identifier'], axis=1)
model = RandomForestRegressor(random_state=1, max_depth=10)
df=pd.get_dummies(df)
model.fit(df,train.Item_Outlet_Sales)
```




```
features = df.columns
importances = model.feature_importances_
indices = np.argsort(importances)[-9:]  # top 10 features
plt.title('Feature Importances')
plt.barh(range(len(indices)), importances[indices], color='b', align='center')
plt.yticks(range(len(indices)), [features[i] for i in indices])
plt.xlabel('Relative Importance')
plt.show()
```

Practice Sections

```
from sklearn.feature_selection import SelectFromModel
feature = SelectFromModel(model)
Fit = feature.fit_transform(df, train.Item_Outlet_Sales)
from sklearn.linear_model import LinearRegression
from sklearn.feature_selection import RFE
from sklearn import datasets
lreg = LinearRegression()
rfe = RFE(lreg, 10)
rfe = rfe.fit_transform(df, train.Item_Outlet_Sales)
from sklearn.feature_selection import f_regression
ffs = f_regression(df,train.Item_Outlet_Sales )
variable = [ ]
fvalue = [ ]
for i in range(0,len(df.columns)-1):
    if ffs[0][i] >=10:
        variable.append(df.columns[i])
        fvalue.append(ffs[0][i])

print("\n".join(variable))
for val in fvalue:
    print(val)
```

```
Item_Visibility
Item_MRP
Outlet_Establishment_Year
Item_Type_Baking Goods
Outlet_Size_Medium
Outlet_Size_Small
Outlet_Location_Type_Tier 1
Outlet_Location_Type_Tier 2
Outlet_Location_Type_Tier 3
Outlet_Type_Grocery Store
Outlet_Type_Supermarket Type1
Outlet_Type_Supermarket Type2
```

```
143.34545133590848
4049.456228781782
20.621569901998658
12.571092140062579
48.40131137865592
83.31637997390477
106.8544614080816
29.022072393471863
18.36605105593487
1739.3213456791361
102.00917067673923
12.360166691489365
```

Practice Sections

The Boston housing data

As a reminder, we are using three features from the Boston housing dataset: `'RM'`, `'LSTAT'`, and `'PTRATIO'`. For each data point (neighborhood):

- `'RM'` is the average number of rooms among homes in the neighborhood.
- `'LSTAT'` is the percentage of homeowners in the neighborhood considered "lower class" (working poor).
- `'PTRATIO'` is the ratio of students to teachers in primary and secondary schools in the neighborhood.

	RM	LSTAT	PTRATIO	MEDV
2	6.575	4.98	15.3	504000.0
3	6.421	9.14	17.8	453600.0
4	7.185	4.03	17.8	728700.0
5	6.998	2.94	18.7	701400.0
6	7.147	5.33	18.7	760200.0
7	6.43	5.21	18.7	602700.0
8	6.012	12.43	15.2	480900.0
9	6.172	19.15	15.2	569100.0

Practice Sections

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans

OrigData = pd.read_csv('../InputData/housing.csv')
data = OrigData
features = data.drop('MEDV', axis = 1)
prices = data['MEDV']

kmeans = KMeans(n_clusters=3)
model = kmeans.fit(features)
classes = model.predict(features)
OrigData['class'] = classes
print(OrigData)
```

	RM	LSTAT	PTRATIO	MEDV	class
0	6.575	4.98	15.3	504000.0	0
1	6.421	9.14	17.8	453600.0	0
2	7.185	4.03	17.8	728700.0	0
3	6.998	2.94	18.7	701400.0	0
4	7.147	5.33	18.7	760200.0	0
5	6.430	5.21	18.7	602700.0	0
6	6.012	12.43	15.2	480900.0	1
7	6.172	19.15	15.2	569100.0	1
8	5.631	29.93	15.2	346500.0	2
9	6.004	17.10	15.2	396900.0	1
10	6.377	20.45	15.2	315000.0	1
11	6.009	13.27	15.2	396900.0	1
12	5.889	15.71	15.2	455700.0	1
13	5.949	8.26	21.0	428400.0	0
14	6.096	10.26	21.0	382200.0	0
15	5.834	8.47	21.0	417900.0	0
16	5.935	6.58	21.0	485100.0	0
17	5.990	14.67	21.0	367500.0	1
18	5.456	11.69	21.0	424200.0	1
19	5.727	11.28	21.0	382200.0	1

Practice Sections

```
# Feature Selections: Dimensionality Reduction with PCA.
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

x = features
y = classes
# Standardizing the features
x = StandardScaler().fit_transform(x)
nSelectedFeature = 3
SelectedAttList = []
for i in range(1, nSelectedFeature + 1):
    SelectedAttList.append("PC" + str(i))

pca = PCA(n_components=nSelectedFeature)
principalComponents = pca.fit_transform(x)
principalDf = pd.DataFrame(data=principalComponents, columns=SelectedAttList)
PCA_Data = principalDf
PCA_Data['class'] = classes
PCA_Data = PCA_Data.dropna()
print(PCA_Data.loc[1:10,:].to_string())
```

	PC1	PC2	PC3	class
1	-0.672757	0.039033	0.169514	0
2	-1.855728	-0.635528	-0.111658	0
3	-1.571191	-0.932394	0.236651	0
4	-1.500134	-0.935298	-0.166790	0
5	-0.827955	-0.488976	0.603086	0
6	-0.578136	1.498918	0.151635	1
7	-0.131723	1.653590	-0.709144	1
8	1.344098	2.403317	-1.247229	2
9	-0.154388	1.681358	-0.320629	1
10	-0.211127	1.574063	-1.059549	1

Practice Sections

```
targets = 'class'
data = PCA_Data.copy()
training = data.sample(frac=0.7, random_state=1)
testing = data.loc[~data.index.isin(training.index)]
TrainData = training.drop(targets, 1)
TargetTrainData = training[targets]
TestData = testing.drop(targets, 1)
TargetTestData = testing[targets]
# Apply the Logistic Regression
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn import metrics
Model = LogisticRegression()
# training by Logistic Regression
Model.fit(TrainData, TargetTrainData.values.ravel())
# predict the test data
PredictTestData = Model.predict(TestData)
# compare and calculate accuracy
Accuracy = accuracy_score(TargetTestData, PredictTestData)
print(Model)
print('PCA Logistic regression accuracy: {:.3f}'.format(Accuracy))
```

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)
PCA Logistic regression accuracy: 0.932
```

Practice Sections

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
model = LDA(n_components=3)
LDAModel = model.fit(TrainData, TargetTrainData)
# predict the test data
PredictTestData = LDAModel.predict(TestData)
# compare and calculate accuracy
Accuracy = accuracy_score(TargetTestData, PredictTestData)
print(LDAModel)
print('LDA accuracy: {:.3f}'.format(Accuracy))
```

```
LinearDiscriminantAnalysis(n_components=3, priors=None, shrinkage=None,
              solver='svd', store_covariance=False, tol=0.0001)
LDA accuracy: 0.959
```

THANK YOU