

Classification Algorithms and Clustering

CSCI 442-542 Lecture Notes, Fall 2018

Charleston Southern University, Charleston, SC, USA

By

Oluleye (Hezekiah) Babatunde, Ph.D

Email: obabatunde@csuniv.edu

October 23, 2018



Classification Algorithms

A **Classification Algorithm** is a well-defined procedure that

- takes **data** as input and
- produces output in the form of models or patterns.

$$y = \underbrace{\text{function}}_{\text{Classifier}} \left(\underbrace{\begin{bmatrix} 1 & 3 & 6 & 3 \\ 5 & 7 & 9 & 3 \\ 6 & 9 & 6 & 3 \end{bmatrix}}_{\text{DataSet}} \right) \rightarrow \text{Class } \underbrace{\{c_1, c_2, \dots, c_n\}}_{\text{ClassiInformation}}$$

Overview of Classification Algorithms

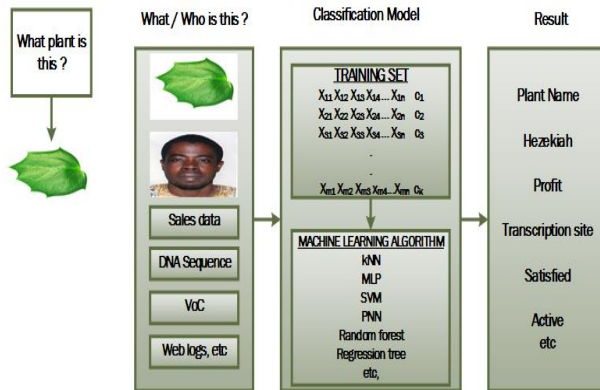


Figure 1: Training data for both supervised and unsupervised ML

A list of classification algorithms

A list of classification algorithms

- Logistic Regression
- Naive Bayes Classifier
- Support Vector Machines
- Decision Trees
- Boosted Trees
- Random Forest
- Neural Networks
- Nearest Neighbor
- etc.,.

Regression

Regression (see page 168 of our text)

The goal of regression is to predict the value of one or more continuous target variables t given the value of a n -dimensional vector x of input variables. The simplest linear model for regression is one that involves a linear combination of the input variables

$$y(\mathbf{x}, \mathbf{w}) = w_0 + w_1x_1 + \dots + w_nx_n \quad (1)$$

where $\mathbf{x} = \{x_1, \dots, x_n\}^T$. More generally Eq 1 can be written as

$$y(\mathbf{x}, \mathbf{w}) = w_0 + \sum_{i=1}^{n-1} w_i \phi(x) \quad (2)$$

where $\phi(x)$ are known as **basis functions**.

Regression (contd.)

The function $\phi(x)$ may be of the following form:

Examples of Basis Functions

$$\phi(x) = \exp -\frac{(x - \mu_i)^2}{2\sigma^2}$$

$$\phi(x) = \psi\left(\frac{x - \mu_i}{\sigma}\right)$$

where $\psi(a)$ is the logistic sigmoid function defined by

$$\psi(a) = \frac{1}{1 + \exp(-a)}$$

Regression (contd.)

Linear Regression in Python (Part1)

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

# Generate dataset called xDATA
xDATA = np.random.random((20, 1))
#print(xDATA)

# yDATA = a*xDATA + b with noise
yDATA = 0.5 * xDATA + 1.0 +
np.random.normal(size=xDATA.shape)
#print(yDATA)

# create a linear regression model
model = LinearRegression()
model.fit(yDATA, yDATA)
```

Regression (contd.)

Linear Regression in Python (Part2)

```
# predict yDATA from the data
xDATA_new = np.linspace(0, 10, 10)
print(xDATA)
yDATA_new = model.predict(xDATA_new[:, np.newaxis])
print(yDATA_new)

# plot the results
plt.figure(figsize=(8, 6))
ax = plt.axes()
ax.scatter(xDATA, yDATA)
ax.plot(xDATA_new, yDATA_new)

ax.set_xlabel('xLABEL')
ax.set_ylabel('yLABEL')

ax.axis('tight')
plt.show()
```


Regression (contd.)

Logistic Regression (Predictive Learning Model)

This is statistical method for analysing a data set with the following assumptions.

- Target variable is binary.
- Predictive features are interval (continuous) or categorical.
- Features are independent of one another.
- Sample size is adequate – Rule of thumb: 50 records per predictor

The goal of logistic regression is to find the best fitting model to describe the relationship between the dichotomous characteristic of interest (dependent variable = response or outcome variable) and a set of independent (predictor or explanatory) variables.

Regression (contd.)

Logistic Regression is given as follows

$$\ln \left(\frac{m}{1-m} \right) = w_0 + b_1 x \quad (3)$$

where

$$m = \frac{1}{1 + \exp^{-y}} \quad (4)$$

This logistic regression function is useful for predicting the class of a binomial target feature.

Logistic regression is similar to linear regression, with the only difference being that the class variable which should contain integer values indicating the class relative to the observation.

Regression (contd.)

```
from sklearn.datasets import load_iris
iris = load_iris()
#print(iris)
Features, ClassInfo = iris.data[:-1,:], iris.target[:-1]
#print(Features), print(ClassInfo)
from sklearn.linear_model import LogisticRegression
logistic = LogisticRegression()
logistic.fit(Features,ClassInfo)
```

Regression (contd.)

Application of Regression

- **Trend lines:** Linear regression can be used to model variation in some quantitative data with passage of time (like GDP, oil prices, etc.)
- **Economics:** Linear regression can be used to predict consumption spending, fixed investment spending, inventory investment, purchases of a country's exports, spending on imports, the demand to hold liquid assets, labor demand, and labor supply.
Finance: Linear regression can be used to analyze capital price asset model and quantify the systematic risks of an investment.
Biology: Linear regression is used to model causal relationships between parameters in biological systems.

Random Forest

Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks, that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random decision forests correct for decision trees' habit of over fitting to their training set.


k-Nearest Neighbor (kNN)

The kNN is a supervised classification algorithm that takes a set of observation and uses them to learn how to label other observations. To predict a new point, the kNN looks at the labelled points closest to that new point (those are its nearest neighbors), and has those neighbors vote, so whichever label the most of the neighbors have is the label for the new point (the “**k**” is the number of neighbors it checks)

Why Classification Algorithms (see [1])

- 1 **Computational finance**, for credit scoring and algorithmic trading.
- 2 **Image processing and computer vision**, for face recognition, motion detection, and object detection.
- 3 **Computational biology**, for tumor detection, drug discovery, and DNA sequencing.
- 4 **Energy production**, for price and load forecasting.
- 5 **Automotive, aerospace, and manufacturing**, for predictive maintenance.
- 6 **Natural language processing**, for voice recognition applications.

DataSet For Classification Algorithms

Training set 

	F1	F2	F3	F4	F5	Class
1	0.0123	0.1440	0.0216	0.0347	0.2331	1
2	0.0056	0.1885	0.0194	0.0177	0.1448	1
3	0.0092	0.2030	0.0191	0.0152	0.1295	1
4	0.0391	0.3195	0.0618	0.1499	0.4714	2
5	0.0610	0.2652	0.0530	0.1331	0.4606	2
6	0.0419	0.2980	0.0539	0.1459	0.4695	2
7	0.0306	0.4230	0.0840	0.1599	0.4729	3
8	0.0294	0.3721	0.0595	0.1419	0.4660	3
9	0.0662	0.3531	0.0589	0.1417	0.4659	3
10	0.0163	0.0764	0.0323	0.0719	0.3557	4
11	0.0493	0.0522	0.0336	0.0730	0.3583	4
12	0.0443	0.0883	0.0362	0.0803	0.3764	4
13	0.0517	0.0536	0.0323	0.0815	0.3870	5
14	0.0453	0.0294	0.0320	0.0746	0.3707	5
15	0.0466	0.0018	0.0295	0.0688	0.3562	5


Test sample  0.0511 0.0083 0.0326 0.0714 0.3628 ?

Figure 2: Training and test data for supervised ML

DataSet For Classification Algorithms (contd.)

<u>Training data</u> <u>(Supervised Learning)</u>	<u>Training data</u> <u>(Unsupervised Learning)</u>
$X_{11} \ X_{12} \ X_{13} \ \dots \ X_{1n} \ C_1$	$X_{11} \ X_{12} \ X_{13} \ \dots \ X_{1n}$
$X_{21} \ X_{22} \ X_{23} \ \dots \ X_{2n} \ C_2$	$X_{21} \ X_{22} \ X_{23} \ \dots \ X_{2n}$
$X_{31} \ X_{32} \ X_{33} \ \dots \ X_{3n} \ C_3$	$X_{31} \ X_{32} \ X_{33} \ \dots \ X_{3n}$
\vdots	\vdots
$X_{m1} \ X_{m2} \ X_{m3} \ \dots \ X_{mn} \ C_k$	$X_{m1} \ X_{m2} \ X_{m3} \ \dots \ X_{mn}$

Figure 3: Training data for both supervised and unsupervised ML

Sample Dataset

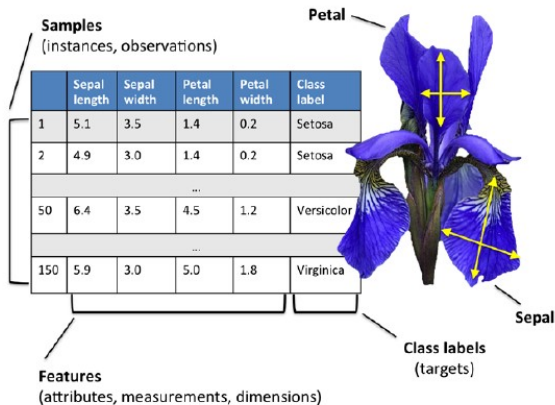


Figure 4: The Iris dataset contains the measurements of 150 iris flowers from three different species: Setosa, Versicolor, and Virginica

Representation of Irish Dataset

Sample Dataset

SN	Feature1	Feature2	Feature3	Feature4	Class
1	$X_{1,1}$	$X_{1,2}$	$X_{1,3}$	$X_{1,4}$	1
2	$X_{2,1}$	$X_{2,2}$	$X_{2,3}$	$X_{2,4}$	1
3	$X_{3,1}$	$X_{3,2}$	$X_{3,3}$	$X_{3,4}$	1
\vdots	\vdots	\vdots	\vdots	\vdots	\vdots
\vdots	\vdots	\vdots	\vdots	\vdots	\vdots
150	$X_{150,1}$	$X_{150,2}$	$X_{150,3}$	$X_{150,4}$	3

- **SN** is the number observation in the dataset.
- **Feature1**, **Feature2**, **Feature3**, **Feature4** are respectively sepal length, sepal width, petal length and petal width respectively.
- **Class** is the class information

Representation of Irish Dataset (contd.)

WHAT'S IN A NAME?

Attributes and labels go by a variety of names, and new machine learners can get tripped up by the name switching from one author to another or even one paragraph to another from a single author.

Attributes (the variables being used to make predictions) are also known as the following:

- Predictors
 - Features
-
- Independent variables
 - Inputs

Labels are also known as the following:

- Outcomes
- Targets
- Dependent variables
- Responses

Figure 5: Information about fields in a dataset

Gender Classification

Gender Classification

Problem: classify whether a given person is a male or a female based on the measured features. The features include height, weight, and foot size. The training data is given as follows.

Person	height (feet)	weight (lbs)	foot size(inches)
male	6	180	12
male	5.92 (5'11")	190	11
male	5.58 (5'7")	170	12
male	5.92 (5'11")	165	10
female	5	100	6
female	5.5 (5'6")	150	8
female	5.42 (5'5")	130	7
female	5.75 (5'9")	150	9

Figure 6: Gender classification

PROBLEM STATEMENT ?

Test data

Testing

Below is a sample to be classified as male or female.

Person	height (feet)	weight (lbs)	foot size(inches)
sample	6	130	8

Figure 7: Test data set for gender classification

Mathematical Representation of Dataset

The **training data** D (for supervised learning) can be presented as

$$\{D | D \in \mathbb{R}^{m \times (n+1)} = \{X_{ij}, y_i\}, i = 1(2)m, j = 1(2)n, y \in \mathbb{R}^{m \times 1}\} \quad (5)$$

where m is the number of observation in the **historical dataset** and n is the number of predictor variables.

The prediction (or **query about test set**) takes the form of the function:

$$f(X_{ij}) = \hat{y} \in \{y_i\}; 1 \leq i \leq m; 1 \leq j \leq n. \quad (6)$$

A general classifier now be written as:

$$Classifier : \{\mathbb{R}^{p \times n}\} \times \{\mathbb{R}^{m \times n}\} \mapsto y_i, i = 1(1)m. \quad (7)$$

where $\mathbb{R}^{p \times n}$ = **test set**, $\mathbb{R}^{m \times n}$ = **training set**, and p is the number of observation in the test set.



Classification Algorithms in BioInformatics

Classification Algorithms in BioInformatics

DNA in our genome is the “blueprint of life” and is a sequence of bases, namely, A, G, C, and T. RNA is transcribed from DNA, and proteins are translated from the RNA. Proteins are what the living body is and does. Just as a DNA is a sequence of bases, a protein is a sequence of amino acids (as defined by bases). One application area of computer science in molecular biology is alignment, which is matching one sequence to another. This is a difficult string matching problem because strings may be quite long, there are many template strings to match against, and there may be deletions, insertions, and substitutions.

Classification Algorithms in BioInformatics (contd.)

Clustering is used in learning motifs, which are sequences of amino acids that occur repeatedly in proteins. Motifs are of interest because they may correspond to structural or functional elements within the sequences they characterize. The analogy is that if the amino acids are letters and proteins are sentences, motifs are like words, namely, a string of letters with a particular meaning occurring frequently in different sentences.

How to Build Classification Algorithms

How to Build Classification Algorithms

- 1. Extract and assemble features to be used for prediction.
- 2. Determine the size and shape and pre-process the dataset. The data sets are most commonly arranged with columns corresponding to a single attribute and rows corresponding to a single observation, but not always. For example, some text mining literature arranges the matrix the other way around—with columns corresponding to an observation and rows corresponding to an attribute.
- 3. Develop targets for the training.
- 4. Train a model.
- 5. Assess performance on test data.

Dividing dataset into training and test set

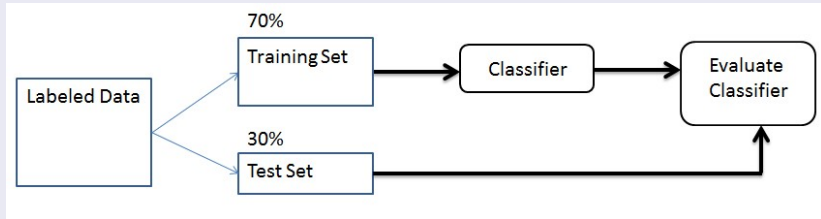


Figure 8: Dividing dataset into training and test set

Machine Learning Process

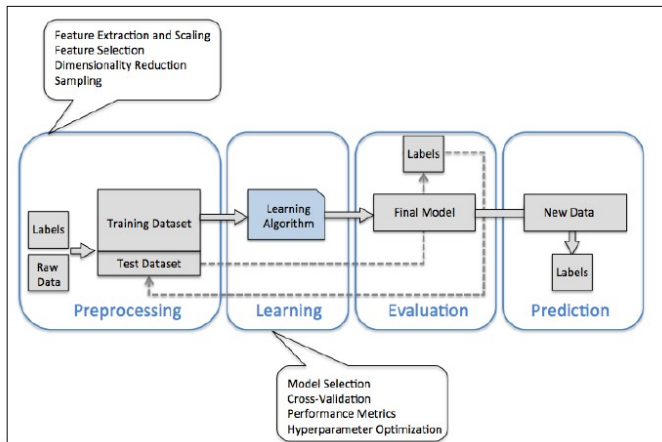


Figure 9: Machine Learning Process

Performance Metric

predicted label	Ariel Sharon	11	2	1	2	0	1	0	0
	Colin Powell	1	59	2	11	0	0	0	0
	Donald Rumsfeld	2	2	27	3	1	0	0	1
	George W Bush	1	3	0	105	1	2	0	1
	Gerhard Schroeder	0	0	0	1	18	2	0	0
	Hugo Chavez	0	0	0	1	0	14	0	0
	Junichiro Koizumi	0	0	0	1	1	0	12	1
	Tony Blair	0	2	1	2	2	1	0	39
		Ariel Sharon	Colin Powell	Donald Rumsfeld	George W Bush	Gerhard Schroeder	Hugo Chavez	Junichiro Koizumi	Tony Blair
		true label							

Figure 10: An example of confusion matrix

Performance Metric

Evaluation metrics

		Actual Label	
		Positive	Negative
Predicted Label	Positive	True Positive (TP)	False Positive (FP)
	Negative	False Negative (FN)	True Negative (TN)
Accuracy	$(TP + TN) / (TP + TN + FP + FN)$	The percentage of predictions that are correct	
Precision	$TP / (TP + FP)$	The percentage of positive predictions that are correct	
Sensitivity (Recall)	$TP / (TP + FN)$	The percentage of positive cases that were predicted as positive	
Specificity	$TN / (TN + FP)$	The percentage of negative cases that were predicted as negative	

Figure 11: Evaluation Metric in Machine Learning

Data Acquisition

```
import csv

with open('C:/Users/obabatunde/Desktop/scores.csv') as csvfile:
    readCSV = csv.reader(csvfile, delimiter=',')
    for row in readCSV:
        print(row)

#
=====
# ['57' 63 78 67 66 37 98 34 32 47']
# ['26' 21 73 8 46 38 56 36 40 49']
# ['29' 19 24 99 50 56 76 31 56 32']
# ['28' 83 86 77 80 62 25 99 11 41']
# ['34' 76 36 31 71 67 49 15 12 16']
# ['20' 30 56 25 63 39 6 98 7 2']
# ['31' 9 26 93 87 90 14 12 17 71']
# ['20' 8 27 57 63 58 84 21 18 51']
# ['38' 65 43 8 29 58 85 65 62 26']
# ['60' 45 85 91 17 99 98 27 2 41']
#
=====
```

Figure 12: Importing Data From CSV Files

Data Acquisition (contd.)

Importing data from WWW

```
url =
"http://esapubs.org/archive/ecol/E084/093/Mammal_lifehistori
es_v2.txt"
data = pd.read_csv(url, delimiter="\t")

print(data.head())

data_by_order = data.groupby('order')

for order, order_data in data_by_order:
    avg_mass = np.mean(order_data['mass(g)'])
    print ("The average mass of {} is {}
grams".format(order, avg_mass))
```

Figure 13: Reading data from WWW

Data Preprocessing

Example: Data Scaling

```
from __future__ import print_function
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler,
RobustScaler

# For reproducibility
np.random.seed(1000)

if __name__ == '__main__':
    # Create a dummy dataset
    data = np.ndarray(shape=(100, 2))

    for i in range(100):
        data[i, 0] = 2.0 + np.random.normal(1.5, 3.0)
        data[i, 1] = 0.5 + np.random.normal(1.5, 3.0)

    # Show the original and the scaled dataset
    fig, ax = plt.subplots(1, 2, figsize=(14, 5))

    ax[0].scatter(data[:, 0], data[:, 1])
```

Figure 14: Part1 of Data Scaling code

Data Preprocessing (contd.)

Example: Data Scaling

```
ax[0].set_xlim([-10, 10])
ax[0].set_ylim([-10, 10])
ax[0].grid()
ax[0].set_xlabel('X')
ax[0].set_ylabel('Y')
ax[0].set_title('Raw data')

# Scale data
ss = StandardScaler()
scaled_data = ss.fit_transform(data)

ax[1].scatter(scaled_data[:, 0], scaled_data[:, 1])
ax[1].set_xlim([-10, 10])
ax[1].set_ylim([-10, 10])
ax[1].grid()
ax[1].set_xlabel('X')
ax[1].set_ylabel('Y')
ax[1].set_title('Scaled data')

plt.show()
```

Figure 15: Part2 of Data Scaling code

Data Preprocessing (contd.)

Example: Data Scaling

```
# Scale data using a Robust Scaler
fig, ax = plt.subplots(2, 2, figsize=(8, 8))

ax[0, 0].scatter(data[:, 0], data[:, 1])
ax[0, 0].set_xlim([-10, 10])
ax[0, 0].set_ylim([-10, 10])
ax[0, 0].grid()
ax[0, 0].set_xlabel('X')
ax[0, 0].set_ylabel('Y')
ax[0, 0].set_title('Raw data')

rs = RobustScaler(quantile_range=(15, 85))
scaled_data = rs.fit_transform(data)

ax[0, 1].scatter(scaled_data[:, 0], scaled_data[:, 1])
ax[0, 1].set_xlim([-10, 10])
ax[0, 1].set_ylim([-10, 10])
ax[0, 1].grid()
ax[0, 1].set_xlabel('X')
ax[0, 1].set_ylabel('Y')
ax[0, 1].set_title('Scaled data (15% - 85%)')
```

Figure 16: Part3 of Data Scaling code

Data Preprocessing (contd.)

Example: Data Scaling

```
rs1 = RobustScaler(quantile_range=(25, 75))
scaled_data1 = rs1.fit_transform(data)

ax[1, 0].scatter(scaled_data1[:, 0], scaled_data1[:, 1])
ax[1, 0].set_xlim([-10, 10])
ax[1, 0].set_ylim([-10, 10])
ax[1, 0].grid()
ax[1, 0].set_xlabel('X')
ax[1, 0].set_ylabel('Y')
ax[1, 0].set_title('Scaled data (25% - 75%)')

rs2 = RobustScaler(quantile_range=(30, 65))
scaled_data2 = rs2.fit_transform(data)

ax[1, 1].scatter(scaled_data2[:, 0], scaled_data2[:, 1])
ax[1, 1].set_xlim([-10, 10])
ax[1, 1].set_ylim([-10, 10])
ax[1, 1].grid()
ax[1, 1].set_xlabel('X')
ax[1, 1].set_ylabel('Y')
ax[1, 1].set_title('Scaled data (30% - 60%)')
plt.show()
```

Figure 17: Part4 of Data Scaling code

Data Preprocessing (contd.)

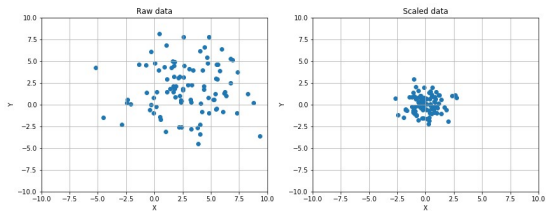


Figure 18: Raw data versus Scaled data

Data Preprocessing (contd.)

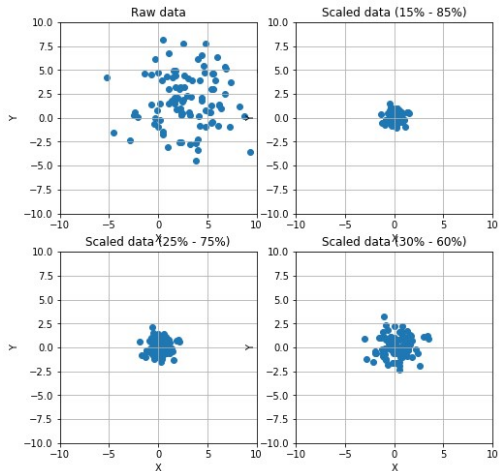


Figure 19: Raw data versus different scaled data

Data Normalization

Data Normalization

$$X = \begin{bmatrix} 35 \\ 36 \\ 46 \\ 68 \\ 70 \end{bmatrix}$$

The norm of the vector X is

$$||Data|| = \sqrt{35^2 + 36^2 + 46^2 + 68^2 + 70^2} = \sqrt{14161} = 119$$

Data Normalization (contd.)

$$\tilde{X} = \begin{bmatrix} \frac{35}{119} = 0.2941 \\ \frac{36}{119} = 0.3025 \\ \frac{46}{119} = 0.3866 \\ \frac{68}{119} = 0.5714 \\ \frac{70}{119} = 0.5882 \end{bmatrix}$$

The norm of vector X is now equal to one:

$$\|X\| = \sqrt{0.2941^2 + 0.3025^2 + 0.3866^2 + 0.5714^2 + 0.5882^2} = 1$$

NOTE:

Data can also be normalized using centering and standard deviation: Z-scores and some other methods.

Data Normalization (contd.)

Python-Based Normalization

```
import numpy as np
from sklearn import preprocessing

X = np.asarray([[35, 36, 38],
               [46, 67, 78],
               [54, 67, 98]],
               dtype=np.float) # Float is needed.

# Before-normalization.
print (X)

# l2-normalize the samples (rows).
X_normalized = preprocessing.normalize(X, norm='l2')

# After normalization.
print (X_normalized)

# Before-normalization.
# [[35. 36. 38.]
#  [46. 67. 78.]
#  [54. 67. 98.]]
#
# [[0.55583571 0.57171674 0.60347878]
#  [0.40836088 0.5947865 0.69243802]
#  [0.41405151 0.51373057 0.75142681]]
```

Figure 20: Python-Based Normalization

Data Normalization (contd.)

Root Mean Squares (RMS)

The RMS value of a dataset is the square root of the arithmetic mean of the squares of the values.

For $X = \{x_1, x_2, \dots, x_n\}$, the RMS

$$X_{rms} = \sqrt{\frac{1}{n} (x_1^2 + x_2^2 + \dots + x_n^2)}$$

For a continuous function (or waveform) $f(t)$ defined over the interval $T_1 \leq t \leq T_2$ is

$$f_{rms} = \sqrt{\frac{1}{T_2 - T_1} \int_{T_1}^{T_2} [f(t)]^2 dt}$$

Data Normalization (contd.)

Matrix Norms

With $p = 1, 2, \infty$, the induced matrix norms can be computed or

estimated by $\|A\|_1 = \max_{1 \leq j \leq n} \sum_{i=1}^m |a_{ij}|$,

which is simply the maximum absolute column sum of the matrix;

$\|A\|_\infty = \max_{1 \leq i \leq m} \sum_{j=1}^n |a_{ij}|$,

which is simply the maximum absolute row sum of the matrix;

$\|A\|_2 = \sigma_{\max}(A) \leq \left(\sum_{i=1}^m \sum_{j=1}^n |a_{ij}|^2 \right)^{1/2} = \|A\|_F$,

Data Normalization (contd.)

p-norm

Let $p \geq 1$ be a real number. The p-norm (also called ℓ_p - norm) of vectors $\mathbf{x} = \mathbf{x} = (x_1, \dots, x_n)$ is

$$\mathbf{x}_p := \left(\sum_{i=1}^n |x_i|^p \right)^{1/p}.$$

Data Normalization (contd.)

Examples of Matrix Norm

Let

$$A = \begin{bmatrix} -3 & 5 & 7 \\ 2 & 6 & 4 \\ 0 & 2 & 8 \end{bmatrix},$$

Then we have

$$\|A\|_1 = \max(|-3| + 2 + 0; 5 + 6 + 2; 7 + 4 + 8) = \max(5, 13, 19) = 19$$

$$\|A\|_\infty = \max(|-3| + 5 + 7; 2 + 6 + 4; 0 + 2 + 8) = \max(15, 12, 10) = 15.$$

Data Normalization (contd.)

Matrix Norms

The norm of a square matrix A is a non-negative real number denoted $\|A\|$. There are several different ways of defining a matrix norm, but they all share the following properties:

1. $\|A\| \geq 0$ for any square matrix A .
2. $\|A\| = 0$ if and only if the matrix $A = 0$.
3. $\|kA\| = |k| \|A\|$, for any scalar k .
4. $\|A + B\| \leq \|A\| + \|B\|$.
5. $\|AB\| \leq \|A\| \|B\|$.

Data Normalization (contd.)

Computation of Matrix Norms

```
from numpy import linalg as LA
a = np.arange(9) + 2
b = a.reshape((3, 3))
LA.norm(a)
print(a)
print(b)
print(LA.norm(b, axis=(1)))

#[ 2  3  4  5  6  7  8  9 10]

# [[ 2  3  4]
#  [ 5  6  7]
#  [ 8  9 10]]

# [ 5.38516481 10.48808848 15.65247584]
```

Figure 21: Python computation of Matrix Norms

Entropy in Classification Algorithms

Entropy in Classification Algorithms

The entropy of a discrete random variable $X = x_i, i = 1(2)n$ with probability mass function $P(X)$ is given as:

$$H(X) = - \sum_{i=1}^n P(x_i) \log_b P(x_i), \text{ b = the base of the logarithm used} \quad (8)$$

The conditional entropy of two events $X = x_i$ and $Y = y_j$ is

$$H(X|Y) = - \sum_{i,j} p(x_i, y_j) \log \frac{p(x_i, y_j)}{p(y_j)}$$

where $p(x_i, y_j)$ is the probability that $X = x_i$ and $Y = y_j$.

Entropy in Classification Algorithms (contd.)

Question: Compute the entropy of the **Product** Column in the table below:

Product	Type
Product1	Type1
Product1	Type2
Product2	Type1
Product2	Type1
Product2	Type2

Solution

$$Pr(Product1) = \frac{2}{5} = 0.4 \text{ and } Pr(Product2) = \frac{3}{5} = 0.6$$

Entropy (of Product Column) =

$$-(\{0.4 * \log_2 0.4\} + \{0.6 * \log_2 0.6\}) = -0.9710$$

Entropy and Information Gain (IG)

Entropy

Let $X, c \in \mathbb{Z}^+$ be the features and class information respectively or pair of discrete random variables. Then the entropy of X is given as

$$H(X) = - \sum_{i=0}^{n-1} p(X = x) \log_2 p(X = x)$$

and

$$H(X|c) = \sum_{i=0}^n p(x = x) H(X|c = c^*)$$

Then $IG(X, Y) = H(X) - H(X|Y)$

Entropy and Information Gain (IG) (contd.)

```
import math
import collections
from sklearn.datasets import load_iris
iris = load_iris()

# store the Features and response vector
Features = iris.data
classInfo = iris.target

def H(X):
    probabilities = [n_x/len(X) for x, n_x in
collections.Counter(X).items()]
    ans = [-p_x*math.log(p_x,2) for p_x in probabilities]
    return sum(ans)

print(H(Features[:,0]), H(Features[:,1]), H(Features[:,2]),
H(Features[:,3]))
print(H(classInfo))

# 4.822018088381165 4.0117097612189285 5.033829378702226
4.065662933799395
# 1.584962500721156
```

Figure 22: Python code for computing entropy

Algorithm for k Nearest Neighbor

Algorithm for k Nearest Neighbor

- 1: **procedure** COMPUTEKNN()
- 2: **Input** $TRAININGSET = \{x_i, c_i\}$, x = feature set, $i = 1(1)M$,
 M = number of observations in the training set, c = class
 information, $j = 1(2)N_c$, N_c = number of classess available and
 x_{test} = test image
- 3: **Assign** $p_i \leftarrow \{x_i, c_i\}$, $i = 1(1)M$, $c_i \in N_c$ where p_i = posteriores
 of x_i .
- 4: **Compute** $D(x_{test}, x_i) = \sqrt{\sum_{m=1}^M (x_{test} - x_{im})^2}$
- 5: **sort** p_i based on D.
- 6: **Select** the first k points from the sorted list
- 7: **Assign** ClassLabel $\leftarrow p^*$ if $c^* = argmax(count(x_i))$
- 8: **end procedure**

Classification Using KNeighborsClassifier (kNN)

Classification Using KNeighborsClassifier (kNN)

```
# Load the dataset.
# Split the dataset into train and testing purpose.
# Euclidean distance calculation function.
# Prediction of classes for records with unknown labels.
# Accuracy calculation of our prediction model.

X = [[0], [1], [2], [3]] # training set
y = [0, 0, 1, 1]         # class label as 0 or 1
from sklearn.neighbors import KNeighborsClassifier
myModel = KNeighborsClassifier(n_neighbors=3)
myModel.fit(X, y)
print(myModel.predict([[1.1]]))
```

Figure 23: Machine Learning Using KNeighborsClassifier (kNN)

The result of the prediction is [0]. which is class 0.

Classification Algorithms in face recognition



Figure 24: Classification Algorithms in face recognition

Complete Code Listing for kNN-based Classification

Complete Code Listing for kNN-based Classification

```
import matplotlib
matplotlib.use('GTKAgg')

import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
from sklearn import neighbors, datasets

n_neighbors = 2
# import some data to play with
iris = datasets.load_iris()
print(iris)

# prepare data
X = iris.data[:, :2]
y = iris.target
h = .02
print(X)
```

Figure 25: Part1 of code listing

Complete Code Listing for kNN-based Classification (contd.)

Complete Code Listing for kNN-based Classification

```
# create an instance of Neighbours Classifier and fit the data.
clf = neighbors.KNeighborsClassifier(n_neighbors,
weights='distance')
clf.fit(X, y)

# calculate min, max and limits
delta = .02
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, delta),
                    np.arange(y_min, y_max, delta))

# predict class using data and kNN classifier
Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])

# Put the result into a color plot
Z = Z.reshape(xx.shape)
plt.figure()
plt.pcolormesh(xx, yy, Z, cmap=cmap_light)
```

Figure 26: Part2 of code listing

Complete Code Listing for kNN-based Classification (contd.)

Complete Code Listing for kNN-based Classification

```
# Plot also the training points
plt.scatter(X[:, 0], X[:, 1], c=y, cmap=cmap_bold)
plt.xlim(xx.min(), xx.max())
plt.ylim(yy.min(), yy.max())
plt.title("3-Class classification (k = %i)" % (n_neighbors))
plt.show()

# create an instance of Neighbours Classifier and fit the
data.
clf = neighbors.KNeighborsClassifier(n_neighbors,
weights='distance')
clf.fit(X, y)

# make prediction
sl = input('Enter sepal length (cm): ') # raw_input
sw = input('Enter sepal width (cm): ')
dataClass = clf.predict([[sl,sw]])
print('Prediction: '),

if dataClass == 0:
    print('Iris Setosa')
elif dataClass == 1:
    print('Iris Versicolour')
else:
    print('Iris Virginica')
```

Figure 27: Part3 of code listing

Working with Unlabeled Data(Clustering Analysis)

Working with Unlabeled Data(Clustering Analysis)

Here we are looking at grouping objects by similarity using k-means: Clustering (or cluster analysis) is a technique that allows us to find groups of similar objects, objects that are more related to each other than to objects in other groups. Examples of business-oriented applications of clustering include the grouping of documents, music, and movies by different topics, or finding customers that share similar interests based on common purchase behaviors as a basis for recommendation engines.

Working with Unlabeled Data(Clustering Analysis) (contd.)

DataSet for Unsupervised Classification

<u>Training data</u> <u>(Supervised Learning)</u>	<u>Training data</u> <u>(Unsupervised Learning)</u>
$X_{11} \ X_{12} \ X_{13} \ \dots \ X_{1n} \ C_1$	$X_{11} \ X_{12} \ X_{13} \ \dots \ X_{1n}$
$X_{21} \ X_{22} \ X_{23} \ \dots \ X_{2n} \ C_2$	$X_{21} \ X_{22} \ X_{23} \ \dots \ X_{2n}$
$X_{31} \ X_{32} \ X_{33} \ \dots \ X_{3n} \ C_3$	$X_{31} \ X_{32} \ X_{33} \ \dots \ X_{3n}$
\vdots	\vdots
\vdots	\vdots
\vdots	\vdots
$X_{m1} \ X_{m2} \ X_{m3} \ \dots \ X_{mn} \ C_k$	$X_{m1} \ X_{m2} \ X_{m3} \ \dots \ X_{mn}$

Figure 28: Training data for unsupervised Classification

Working with Unlabeled Data(Clustering Analysis) (contd.)

kMeans

The k-means algorithm divides the given dataset into disjoint clusters , each described by the mean of the samples in the cluster. The means are commonly called the cluster “**centroids**”. The K-means algorithm aims to choose centroids that minimise the inertia, or within-cluster sum of squared criterion. This inertia is also called **scoring function**.

$$\sum_{i=0}^n \min_{\mu_j \in C} (||x_i - \mu_j||^2)$$

Working with Unlabeled Data(Clustering Analysis) (contd.)

Principal Component Analysis (PCA Page 79 of our text)

PCA is statistical technique for identifying patterns in data, and expressing the data in such a way as to highlight their similarities and differences. Since patterns in data can be hard to find in data of high dimension, where the luxury of graphical representation is not available, PCA is a powerful tool for analysing data.

PCA uses orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called principal components. If there are m observations with n variables, then the number of distinct principal components is $\min(m - 1, n)$.

Working with Unlabeled Data(Clustering Analysis) (contd.)

How to Compute PCA

- **1:**Get some data
- **2:**Subtract the **mean of the column data** from from each data in the column. This produces a data set whose mean is zero.
- **3:** Calculate the **covariance** matrix.
- **4:** Calculate the **eigen vectors and eigen values** of the covariance matrix.
- **5:** Choose the components and form a feature vector

The first principal component has the largest possible variance

Working with Unlabeled Data(Clustering Analysis)

(contd.)

Worked Example

x	y	$x - \bar{x}$	$y - \bar{y}$
3	4	-2.4	0.6
9	2	3.6	-1.4
3	1	-2.4	-2.4
8	3	2.6	-0.4
4	7	-1.4	3.6

Note: The **eigenvector** with the highest **eigenvalue** is the principal component of the data set. It is the most significant relationship between the data dimensions.

Working with Unlabeled Data(Clustering Analysis) (contd.)

Compute the Covariance of $\{x, y\}$

$$\begin{bmatrix} 8.3 & -1.7 \\ -1.7 & 5.3 \end{bmatrix}$$

Compute the Eignevalues of the Covariance Matrix

$$\begin{bmatrix} 4.5328 \\ 9.0672 \end{bmatrix}$$

Compute the Eignevectors of the Covariance Matrix

$$\begin{bmatrix} -0.4113 & -0.9115 \\ -0.9115 & 0.4113 \end{bmatrix}$$

Working with Unlabeled Data(Clustering Analysis) (contd.)

Final Step in PCA

This is constructed by taking the eigen vectors that you want to keep from the list of eigen vectors, and forming a matrix with these eigen vectors in the columns.

Computing PCA with Python

Irish DataSet

$$\mathbf{x}^T = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} \text{sepal length} \\ \text{sepal width} \\ \text{petal length} \\ \text{petal width} \end{pmatrix}$$

Computing PCA with Python (contd.)

Covariance

$$\sigma_{jk} = \frac{1}{n-1} \sum_{i=1}^N (x_{ij} - \bar{x}_j) (x_{ik} - \bar{x}_k).$$

where

$$\bar{\mathbf{x}} = \sum_{k=1}^n x_i.$$

Computing PCA with Python (contd.)

Python code example

```
from sklearn.decomposition import PCA as sklearnPCA
sklearn_pca = sklearnPCA(n_components=2)
import pandas as pd

df = pd.read_csv(
    filepath_or_buffer='https://archive.ics.uci.edu/ml/machine-
learning-databases/iris/iris.data',
    header=None,
    sep=',')
Features = df.ix[:,0:3].values
Class = df.ix[:,4].values
print(Features)
#print(Class)
from sklearn.preprocessing import StandardScaler
Features_std = StandardScaler().fit_transform(Features)
Class_sklearn = sklearn_pca.fit_transform(Features_std)
print(Class_sklearn)
```

References



[https:](https://www.mathworks.com/discovery/machine-learning.html)

[//www.mathworks.com/discovery/machine-learning.html](https://www.mathworks.com/discovery/machine-learning.html)



Babatunde, O. H. (2015). *A neuro-genetic hybrid approach to automatic identification of plant leaves*. Retrieved from <http://ro.ecu.edu.au/theses/1733>.



[https:](https://www.mathworks.com/discovery/machine-learning.html)

[//www.mathworks.com/discovery/machine-learning.html](https://www.mathworks.com/discovery/machine-learning.html)