DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING (ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING)

II B.Tech II Semester

Subject Name: MACHINE LEARNING FOUNDATIONS LAB
Subject Code: C6605
Regulations: MR-22

Lab Manual

Academic Year: 2024-25

MALLA REDDY ENGINEERING COLLEGE (AUTONOMOUS)
MAIN CAMPUS
(An UGC Autonomous Institution, Approved by AICTE and Affiliated to JNTUH, Hyderabad, Accredited by NAAC with 'A++' Grade (III Cycle))
NBA Accredited Programmes – UG (CE, EEE, ME, ECE, & CSE), PG (CE-SE, EEE, EPS, ME-TE)
Maisammaguda(H), Gundlapochampally Village, Medchal Mandal,
Medchal-Malkajgiri District, Telangana State – 500100

MALLA REDDY ENGINEERING COLLEGE (AUTONOMOUS)

MR22 – ACADEMIC REGULATIONS (CBCS)

for B.Tech. (REGULAR) DEGREE PROGRAMME

Applicable for the students of B.Tech. (Regular) programme admitted from the Academic Year 2022-23 onwards

The B.Tech. Degree of Jawaharlal Nehru Technological University Hyderabad, Hyderabad shall be conferred on candidates who are admitted to the programme and who fulfill all the requirements for the award of the Degree.

VISION OF THE INSTITUTE

To be a premier center of professional education and research, offering quality programs in a socio-economic and ethical ambience.

MISSION OF THE INSTITUTE

- To impart knowledge of advanced technologies using state-of-the-art infrastructural facilities.
- To inculcate innovation and best practices in education, training and research.
- To meet changing socio-economic needs in an ethical ambience.

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING – ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

DEPARTMENT VISION

To attain global standards in Computer Science and Engineering education, training and research to meet the growing needs of the industry with socio-economic and ethical considerations.

DEPARTMENT MISSION

- To impart quality education and research to undergraduate and postgraduate students in Computer Science and Engineering.
- To encourage innovation and best practices in Computer Science and Engineering utilizing state-of-the-art facilities.
- To develop entrepreneurial spirit and knowledge of emerging technologies based on ethical values and social relevance.

PROGRAMME EDUCATIONAL OBJECTIVES (PEOs)

PEO1: Graduates will demonstrate technical skills, competency in AI & ML and exhibit team management capability with proper communication in a job environment

PEO2: Graduates will function in their profession with social awareness and responsibility

PEO3: Graduates will interact with their peers in other disciplines in industry and society and contribute to the economic growth of the country

PEO4: Graduates will be successful in pursuing higher studies in engineering or management

PROGRAMME OUTCOMES (POs)

PO1: Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

PO2: Problem analysis: Identify, formulate, review research literature and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

PO3: Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

PO4: Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

PO5: Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

PO6: The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

PO7: Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

PO8: Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

PO9: Individual and team work: Function effectively as an individual and as a member or leader in diverse teams, and in multidisciplinary settings.

PO10: Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

PO11: Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

PO12: Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

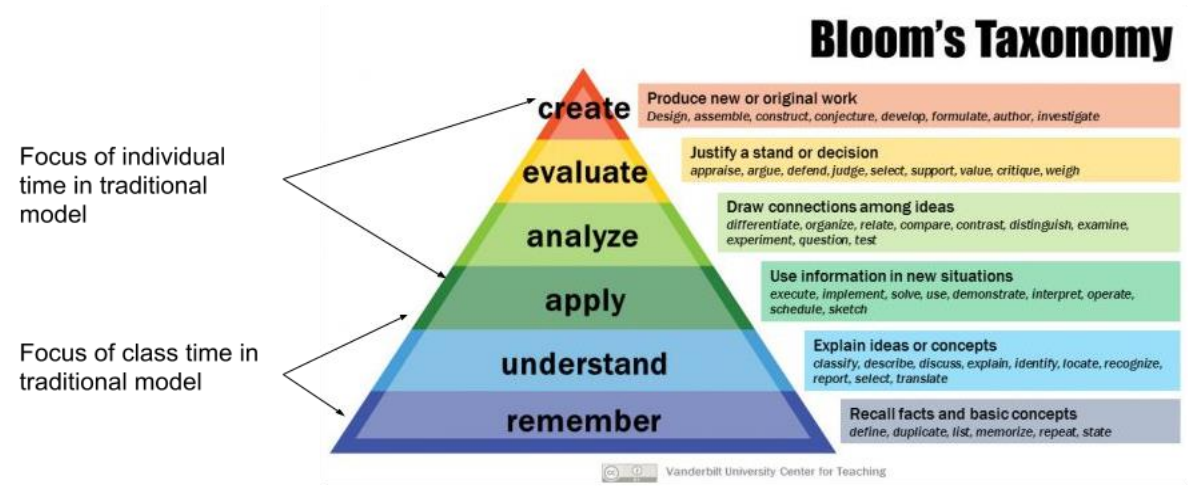
PROGRAMME SPECIFIC OUTCOMES (PSOs)

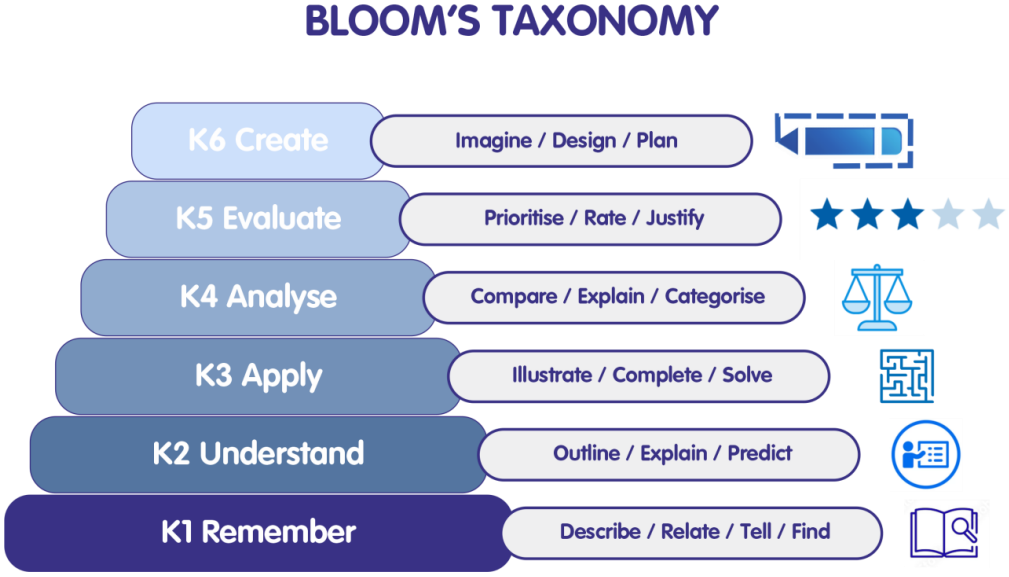
PSO1: Design and develop intelligent automated systems applying mathematical, analytical, programming and operational skills to solve real world problems

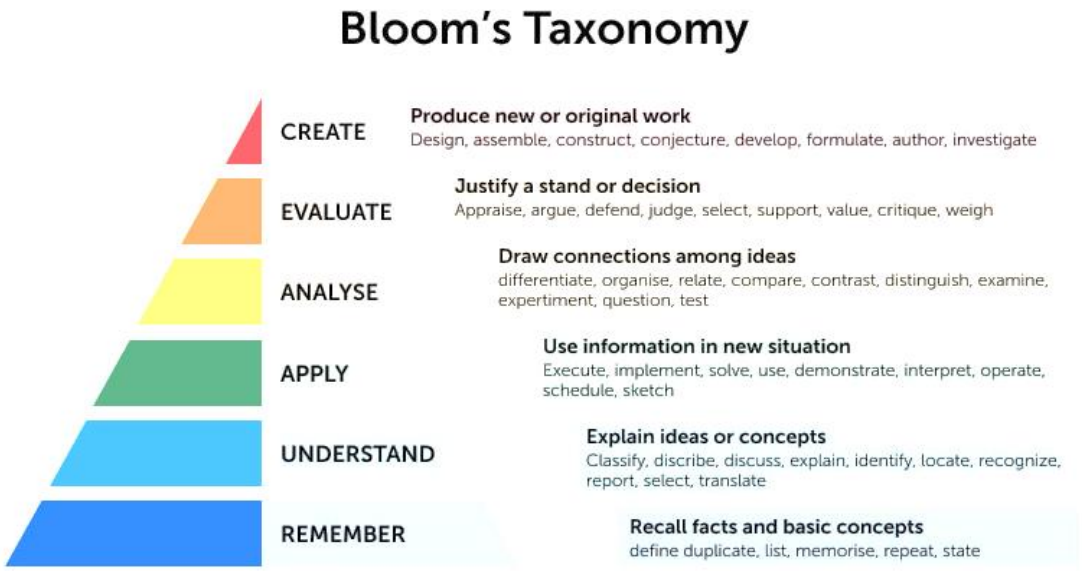
PSO2: Apply machine learning techniques, software tools to conduct experiments, interpret data and to solve complex problems

PSO3: Implement engineering solutions for the benefit of society by the use of AI and ML

BLOOM'S TAXONOMY (BT) TRIANGLE &
BLOOM'S ACTION VERBS







BLOOM'S ACTION VERBS

REVISED Bloom's Taxonomy Action Verbs

Definitions	I. Remembering	II. Understanding	III. Applying	IV. Analyzing	V. Evaluating	VI. Creating
Bloom's Definition	Exhibit memory of previously learned material by recalling facts, terms, basic concepts, and answers.	Demonstrate understanding of facts and ideas by organizing, comparing, translating, interpreting, giving descriptions, and stating main ideas.	Solve problems to new situations by applying acquired knowledge, facts, techniques and rules in a different way.	Examine and break information into parts by identifying motives or causes. Make inferences and find evidence to support generalizations.	Present and defend opinions by making judgments about information, validity of ideas, or quality of work based on a set of criteria.	Compile information together in a different way by combining elements in a new pattern or proposing alternative solutions.
Verbs	• Choose • Define • Find • How • Label • List • Match • Name • Omit • Recall • Relate • Select • Show • Spell • Tell • What • When • Where • Which • Who • Why	• Classify • Compare • Contrast • Demonstrate • Explain • Extend • Illustrate • Infer • Interpret • Outline • Relate • Rephrase • Show • Summarize • Translate	• Apply • Build • Choose • Construct • Develop • Experiment with • Identify • Interview • Make use of • Model • Organize • Plan • Select • Solve • Utilize	• Analyze • Assume • Categorize • Classify • Compare • Conclusion • Contrast • Discover • Dissect • Distinguish • Divide • Examine • Function • Inference • Inspect • List • Motive • Relationships • Simplify • Survey • Take part in • Test for • Theme	• Agree • Appraise • Assess • Award • Choose • Compare • Conclude • Criteria • Criticize • Decide • Deduct • Defend • Determine • Disprove • Estimate • Evaluate • Explain • Importance • Influence • Interpret • Judge • Justify • Mark • Measure • Opinion • Perceive • Prioritize • Prove • Rate • Recommend • Rule on • Select • Support • Value	• Adapt • Build • Change • Choose • Combine • Compile • Compose • Construct • Create • Delete • Design • Develop • Discuss • Elaborate • Estimate • Formulate • Happen • Imagine • Improve • Invent • Make up • Maximize • Minimize • Modify • Original • Originate • Plan • Predict • Propose • Solution • Solve • Suppose • Test • Theory

Anderson, L. W., & Krathwohl, D. R. (2001). A taxonomy for learning, teaching, and assessing, Abridged Edition. Boston, MA: Allyn and Bacon.

2022-23 Onwards (MR-22)	MALLA REDDY ENGINEERING COLLEGE (AUTONOMOUS)	B.Tech. IV Semester		
Code: C6605	MACHINE LEARNING FOUNDATIONS LAB	L	T	P
Credits: 1		-	-	2

Course Objectives:

- This objective of the course is to provide the students to implement the various supervised and unsupervised learning techniques along with the clustering and classification methods.

List of Experiments

1. Implement simple linear regression.
2. Implement the multivariate linear regression.
3. Implement simple logistic regression and multivariate logistics regression.
4. Implement decision trees.
5. Implement a classification algorithm.
6. Implement random forests algorithm.
7. Implement K-means with example.
8. Implement KNN algorithms with example.
9. Implement SVM on any applicable datasets.
10. Implement neural networks.
11. Implement PCA.
12. Implement anomaly detection and recommendation.

REFERENCE BOOKS:

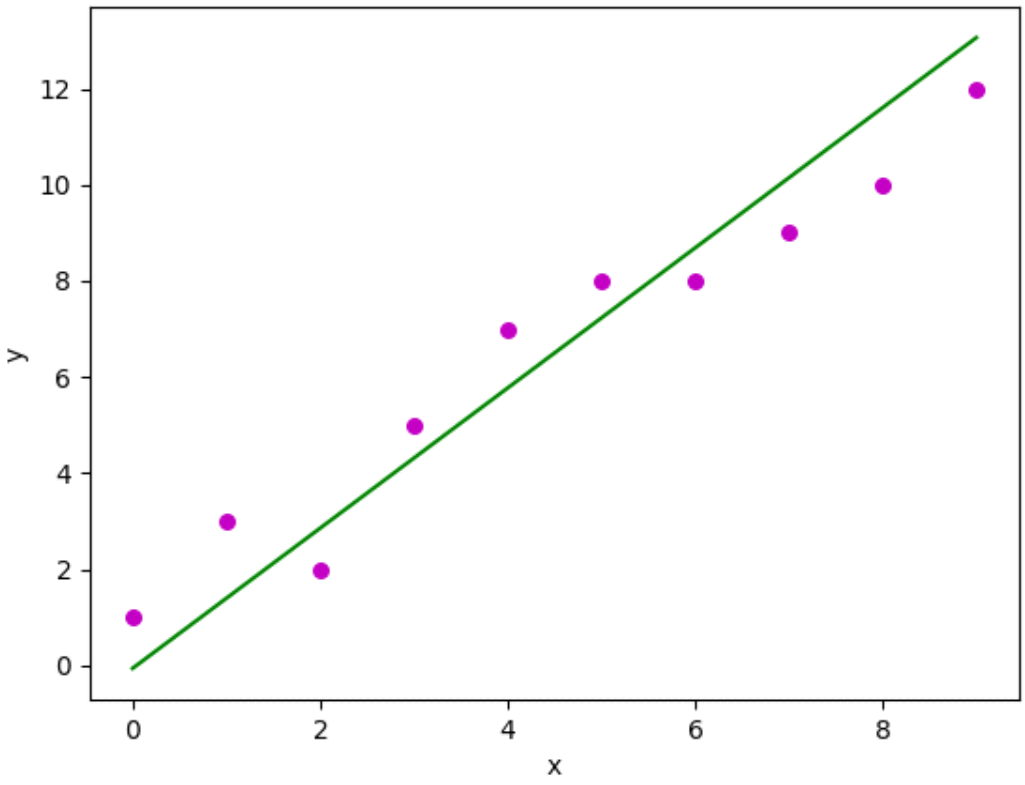
1. Willi Richert, Luis Pedro Coelho, "Building Machine Learning with Python", Packt Publishing, 2013.

1. Write a program to implement simple linear regression.

Source code:

```
import numpy as np
import matplotlib.pyplot as plt
def estimate_coef(x, y):
   n = np.size(x)
   m_x = np.mean(x)
   m_y = np.mean(y)
   SS_xy = np.sum(y*x) - n*m_y*m_x
   SS_xx = np.sum(x*x) - n*m_x*m_x
   b_1 = SS_xy / SS_xx
   b_0 = m_y - b_1*m_x
   return (b_0, b_1)
 def plot_regression_line(x, y, b):
   plt.scatter(x, y, color = "m",
     marker = "o", s = 30)
    y_pred = b[0] + b[1]*x
   plt.plot(x, y_pred, color = "g")
   plt.xlabel('x')
   plt.ylabel('y')
   plt.show()
  def main():
   x = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
   y = np.array([1, 3, 2, 5, 7, 8, 8, 9, 10, 12])
   b = estimate_coef(x, y)
   print("Estimated coefficients:\nb_0 = {}  \
   \nb_1 = {}".format(b[0], b[1]))
   plot_regression_line(x, y, b)
if __name__ == "__main__":
   main()
```

Output:

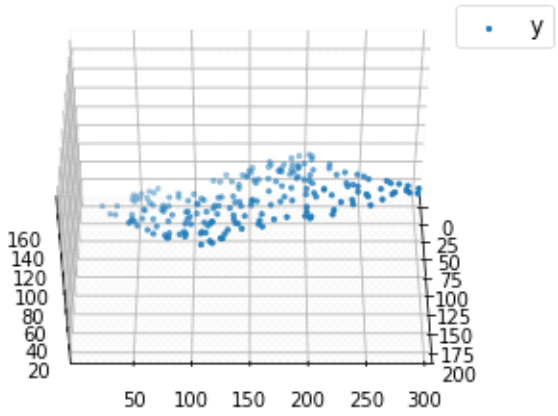


2. Write a program to implement multivariate linear regression

Sourcecode:

```
import numpy as np
import matplotlib as mpl
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt

def generate_dataset(n):
    x = []
    y = []
    random_x1 = np.random.rand()
    random_x2 = np.random.rand()
    for i in range(n):
        x1 = i
        x2 = i/2 + np.random.rand()*n
        x.append([1, x1, x2])
        y.append(random_x1 * x1 + random_x2 * x2 + 1)
    return np.array(x), np.array(y)

x, y = generate_dataset(200)

mpl.rcParams['legend.fontsize'] = 12

fig = plt.figure()
ax = fig.gca(projection ='3d')

ax.scatter(x[:, 1], x[:, 2], y, label ='y', s = 5)
ax.legend()
ax.view_init(45, 0)

plt.show()
```

output:


y
160
140
120
100
80
60
40
20
50
100
150
200
250
300
0
25
50
75
100
125
150
175
200

3(a) Write a program to implement simple logistic regression.

Sourcecode:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import os
path = os.getcwd() + '\data\ex2data1.txt'
data = pd.read_csv(path, header=None, names=['Exam 1', 'Exam 2', 'Admitted'])
data.head()

positive = data[data['Admitted'].isin([1])]
negative = data[data['Admitted'].isin([0])]

fig, ax = plt.subplots(figsize=(12,8))
ax.scatter(positive['Exam 1'], positive['Exam 2'], s=50, c='b', marker='o', label='Admitted')
ax.scatter(negative['Exam 1'], negative['Exam 2'], s=50, c='r', marker='x', label='Not Admitted')
ax.legend()
ax.set_xlabel('Exam 1 Score')
ax.set_ylabel('Exam 2 Score')

def sigmoid(z):
return 1 / (1 + np.exp(-z))
nums = np.arange(-10, 10, step=1)
fig, ax = plt.subplots(figsize=(12,8))
ax.plot(nums, sigmoid(nums), 'r')

def cost(theta, X, y):
theta = np.matrix(theta)
X = np.matrix(X)
y = np.matrix(y)
first = np.multiply(-y, np.log(sigmoid(X * theta.T)))
second = np.multiply((1 - y), np.log(1 - sigmoid(X * theta.T)))
return np.sum(first - second) / (len(X))
data.insert(0, 'Ones', 1)
cols = data.shape[1]
X = data.iloc[:,0:cols-1]
y = data.iloc[:,cols-1:cols]
X = np.array(X.values)
y = np.array(y.values)
theta = np.zeros(3)

X.shape, theta.shape, y.shape
```

```
cost(theta, X, y)

def gradient(theta, X, y):
theta = np.matrix(theta)
X = np.matrix(X)
y = np.matrix(y)
parameters = int(theta.ravel().shape[1])
grad = np.zeros(parameters)
error = sigmoid(X * theta.T) - y
for i in range(parameters):
term = np.multiply(error, X[:,i])
grad[i] = np.sum(term) / len(X)
return grad

import scipy.optimize as opt
result = opt.fmin_tnc(func=cost, x0=theta, fprime=gradient, args=(X, y))
cost(result[0], X, y)

def predict(theta, X):
probability = sigmoid(X * theta.T)
return [1 if x >= 0.5 else 0 for x in probability]
theta_min = np.matrix(result[0])
predictions = predict(theta_min, X)
correct = [1 if ((a == 1 and b == 1) or (a == 0 and b == 0)) else 0 for (a, b) in
zip(predictions, y
)]
accuracy = (sum(map(int, correct)) % len(correct))
print ('accuracy = {0}%'.format(accuracy))
```

output:

	Exam 1	Exam 2	Admitted
0	34.623660	78.024693	0
1	30.286711	43.894998	0
2	35.847409	72.902198	0
3	60.182599	86.308552	1
4	79.032736	75.344376	1

3(b) Write a program to implement multivariate logistics regression

Sourcecode:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy.io import loadmat
%matplotlib inline
data = loadmat('data/ex3data1.mat')
data

data['X'].shape, data['y'].shape

def sigmoid(z):
return 1 / (1 + np.exp(-z))
def cost(theta, X, y, learningRate):
theta = np.matrix(theta)
X = np.matrix(X)
y = np.matrix(y)
first = np.multiply(-y, np.log(sigmoid(X * theta.T)))
second = np.multiply((1 - y), np.log(1 - sigmoid(X * theta.T)))
reg = (learningRate / 2 * len(X)) * np.sum(np.power(theta[:,1:theta.shape[1]], 2))
return np.sum(first - second) / (len(X)) + reg
def gradient_with_loop(theta, X, y, learningRate):
theta = np.matrix(theta)
X = np.matrix(X)
y = np.matrix(y)
parameters = int(theta.ravel().shape[1])
grad = np.zeros(parameters)
error = sigmoid(X * theta.T) - y
for i in range(parameters):
term = np.multiply(error, X[:,i])
if (i == 0):
grad[i] = np.sum(term) / len(X)
else:
grad[i] = (np.sum(term) / len(X)) + ((learningRate / len(X)) * theta[:,i])
return grad
def gradient(theta, X, y, learningRate):
theta = np.matrix(theta)
X = np.matrix(X)
y = np.matrix(y)
parameters = int(theta.ravel().shape[1])
error = sigmoid(X * theta.T) - y
grad = ((X.T * error) / len(X)).T + ((learningRate / len(X)) * theta)
grad[0, 0] = np.sum(np.multiply(error, X[:,0])) / len(X)
```

```
return np.array(grad).ravel()
from scipy.optimize import minimize
def one_vs_all(X, y, num_labels, learning_rate):
rows = X.shape[0]
params = X.shape[1]
all_theta = np.zeros((num_labels, params + 1))
X = np.insert(X, 0, values=np.ones(rows), axis=1)
for i in range(1, num_labels + 1):
theta = np.zeros(params + 1)
y_i = np.array([1 if label == i else 0 for label in y])
y_i = np.reshape(y_i, (rows, 1))
fmin = minimize(fun=cost, x0=theta, args=(X, y_i, learning_rate), method='TNC',
jac=gradient)
all_theta[i-1,:] = fmin.x
return all_theta
rows = data['X'].shape[0]
params = data['X'].shape[1]
all_theta = np.zeros((10, params + 1))
X = np.insert(data['X'], 0, values=np.ones(rows), axis=1)
theta = np.zeros(params + 1)
y_0 = np.array([1 if label == 0 else 0 for label in data['y']])
y_0 = np.reshape(y_0, (rows, 1))
X.shape, y_0.shape, theta.shape, all_theta.shape

np.unique(data['y'])
np.unique(data['y'])
all_theta = one_vs_all(data['X'], data['y'], 10, 1)
all_theta
def predict_all(X, all_theta):
rows = X.shape[0]
params = X.shape[1]
num_labels = all_theta.shape[0]
X = np.insert(X, 0, values=np.ones(rows), axis=1)
X = np.matrix(X)
all_theta = np.matrix(all_theta)
h = sigmoid(X * all_theta.T)
h_argmax = np.argmax(h, axis=1)
h_argmax = h_argmax + 1
return h_argmax
y_pred = predict_all(data['X'], all_theta)
correct = [1 if a == b else 0 for (a, b) in zip(y_pred, data['y'])]
accuracy = (sum(map(int, correct)) / float(len(correct)))
print ('accuracy = {0}%'.format(accuracy * 100))
accuracy = 74.6%
```

output:

{'__header__': b'MATLAB 5.0 MAT-file, Platform: GLNXA64, Created on: Sun Oct 16 13:09:09 2011',

'__version__': '1.0',
'__globals__': [],
'X': array([[0., 0., 0., ..., 0., 0., 0.],
[0., 0., 0., ..., 0., 0., 0.],
[0., 0., 0., ..., 0., 0., 0.],
...,
[0., 0., 0., ..., 0., 0., 0.],
[0., 0., 0., ..., 0., 0., 0.],
[0., 0., 0., ..., 0., 0., 0.]]),
'y': array([[10],
[10],
[10],
...,
[9],
[9],
[9]], dtype=uint8)}

array([[-3.70247923e-05, 0.00000000e+00, 0.00000000e+00, ...,
-2.24803603e-10, 2.31962906e-11, 0.00000000e+00],
[-8.96250745e-05, 0.00000000e+00, 0.00000000e+00, ...,
7.26120886e-09, -6.19965350e-10, 0.00000000e+00],
[-8.39553309e-05, 0.00000000e+00, 0.00000000e+00, ...,
-7.61695539e-10, 4.64917610e-11, 0.00000000e+00],
...,
[-7.00832394e-05, 0.00000000e+00, 0.00000000e+00, ...,
-6.92008993e-10, 4.29241468e-11, 0.00000000e+00],
[-7.65187921e-05, 0.00000000e+00, 0.00000000e+00, ...,
-8.09503259e-10, 5.31058709e-11, 0.00000000e+00],
[-6.63412370e-05, 0.00000000e+00, 0.00000000e+00, ...,
-3.49765862e-09, 1.13668519e-10, 0.00000000e+00]])

4. Write a program to implement decision trees.

Sourcecode:

```
import warnings
warnings.filterwarnings("ignore")
# Importing the required packages
import numpy as np
import pandas as pd
from sklearn.metrics import confusion_matrix
from sklearn.cross_validation import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
# Function importing Dataset
def importdata():
balance_data = pd.read_csv('https://archive.ics.uci.edu/ml/machine-learning-
databases/balance-sca
le/balance-scale.data',
sep= ',', header = None)
# Printing the dataswet shape
print ("Dataset Lenght: ", len(balance_data))
print ("Dataset Shape: ", balance_data.shape)
print ("Dataset: ",balance_data.head())
return balance_data
def splitdataset(balance_data):
X = balance_data.values[:, 1:5]
Y = balance_data.values[:, 0]
X_train, X_test, y_train, y_test = train_test_split(
X, Y, test_size = 0.3, random_state = 100)
return X, Y, X_train, X_test, y_train, y_test
def train_using_gini(X_train, X_test, y_train):
clf_gini = DecisionTreeClassifier(criterion = "gini",
random_state = 100,max_depth=3, min_samples_leaf=5)
clf_gini.fit(X_train, y_train)
return clf_gini
def tarin_using_entropy(X_train, X_test, y_train):
clf_entropy = DecisionTreeClassifier(
criterion = "entropy", random_state = 100,
max_depth = 3, min_samples_leaf = 5)
clf_entropy.fit(X_train, y_train)
return clf_entropy
def prediction(X_test, clf_object):
y_pred = clf_object.predict(X_test)
print("Predicted values:")
print(y_pred)
return y_pred
```

```
return y_pred
def cal_accuracy(y_test, y_pred):
print("Confusion Matrix: ",
confusion_matrix(y_test, y_pred))
print ("Accuracy : ",
accuracy_score(y_test,y_pred)*100)
print("Report : ",
classification_report(y_test, y_pred))
def main():
data = importdata()
X, Y, X_train, X_test, y_train, y_test = splitdataset(data)
clf_gini = train_using_gini(X_train, X_test, y_train)
clf_entropy = tarin_using_entropy(X_train, X_test, y_train)
print("Results Using Gini Index:")
y_pred_gini = prediction(X_test, clf_gini)
cal_accuracy(y_test, y_pred_gini)
print("Results Using Entropy:")
y_pred_entropy = prediction(X_test, clf_entropy)
cal_accuracy(y_test, y_pred_entropy)
if __name__=="__main__":
main()
```

output:

```
Dataset Lenght: 625
Dataset Shape: (625, 5)
Dataset: 0 1 2 3 4
0 B 1 1 1 1
1 R 1 1 1 2
2 R 1 1 1 3
3 R 1 1 1 4
4 R 1 1 1 5
Results Using Gini Index:
Predicted values:
['R''L''R''R''R''L''R''L''L''L''R''L''L''L''R''L''R''L'
'L''R''L''R''L''L''R''L''L''L''R''L''L''L''R''L''L''L'
'L''R''L''L''R''L''R''L''R''R''L''L''R''L''R''R''L''R'
'R''L''R''R''L''L''R''R''L''L''L''L''L''R''R''L''L''R'
'R''L''R''L''R''R''R''L''R''L''L''L''L''R''R''L''R''L'
'R''R''L''L''L''R''R''L''L''L''R''L''R''R''R''R''R''R'
'R''L''R''L''R''R''L''R''R''R''R''R''L''R''L''L''L''L'
'L''L''L''R''R''R''R''L''R''R''R''L''L''R''L''R''L''R'
'L''L''R''L''L''R''L''R''L''R''R''R''L''R''R''R''R''R'
'L''L''R''R''R''R''L''R''R''R''L''R''L''L''L''L''R''R'
'L''R''R''L''L''R''R''R']
```

Confusion Matrix: [[0 6 7]
[0 67 18]
[0 19 71]]
Accuracy : 73.40425531914893
Report : precision recall f1-score support
B 0.00 0.00 0.00 13
L 0.73 0.79 0.76 85
R 0.74 0.79 0.76 90
avg / total 0.68 0.73 0.71 188
Results Using Entropy:
Predicted values:
['R''L''R''L''R''L''R''L''R''R''R''R''L''L''R''L''R''L'
'L''R''L''R''L''L''R''L''R''L''R''L''R''L''R''L''L''L'
'L''R''L''R''L''L''R''L''R''L''R''L''R''L''R''L''L''L'
'L''L''R''L''R''L''R''L''R''R''L''L''R''L''L''R''L''L'
'R''L''R''R''L''R''R''R''L''L''R''L''L''R''L''L''L''R'
'R''L''R''L''R''R''R''L''R''L''L''L''L''R''R''L''R''L'
'R''R''L''L''L''R''R''L''L''L''R''L''L''R''R''R''R''R'
'R''L''R''L''R''R''L''R''R''L''R''R''L''R''R''R''L''L'
'L''L''L''R''R''R''R''L''R''R''R''L''L''R''L''R''L''R'
'L''R''R''L''L''R''L''R''R''R''R''R''L''R''R''R''R''R'
'R''L''R''L''R''R''L''R''L''R''L''R''L''L''L''L''L''R'
'R''R''L''L''L''R''R''R']
Confusion Matrix: [[0 6 7]
[0 63 22]
[0 20 70]]
Accuracy : 70.74468085106383
Report : precision recall f1-score support
B 0.00 0.00 0.00 13
L 0.71 0.74 0.72 85
R 0.71 0.78 0.74 90
avg / total 0.66 0.71 0.68 188

5. Write a program to implement - Classification Algorithm

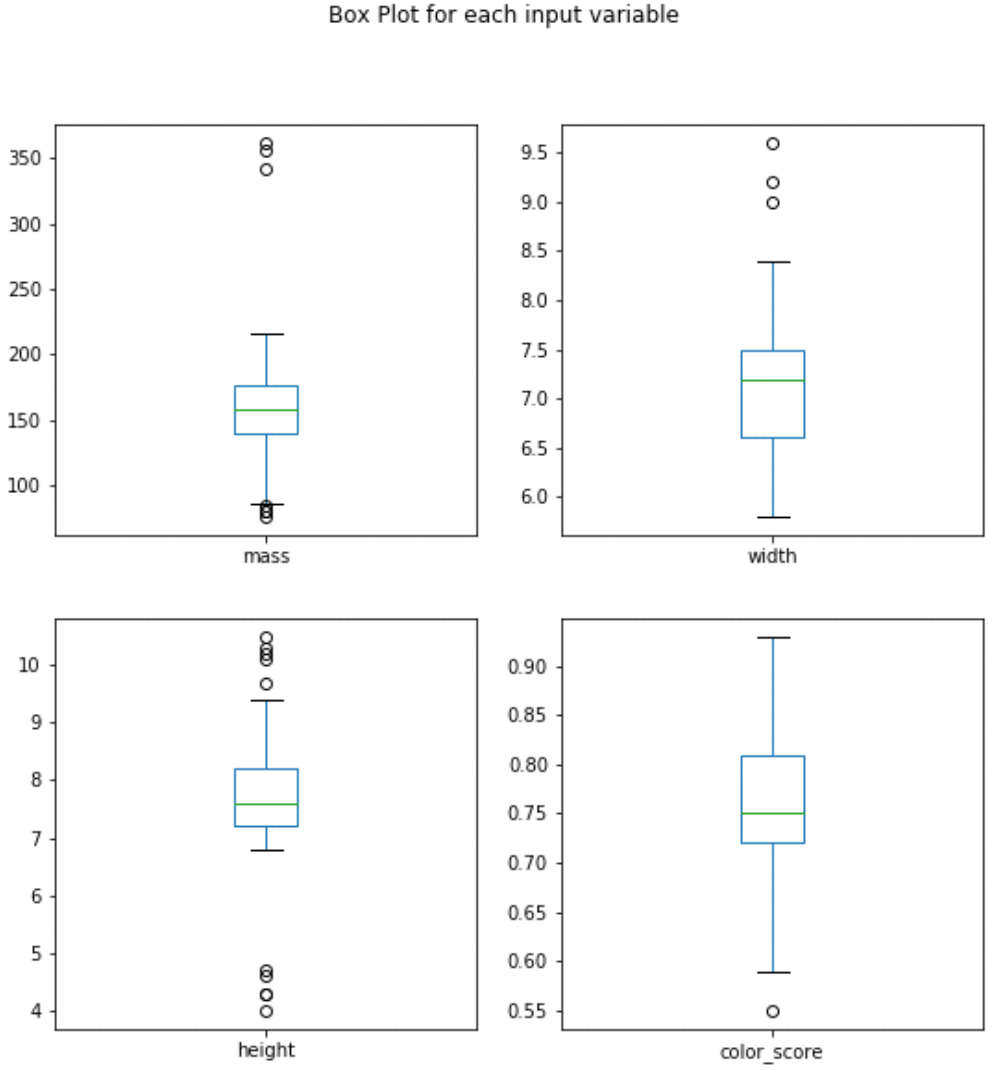
Sourcecode:

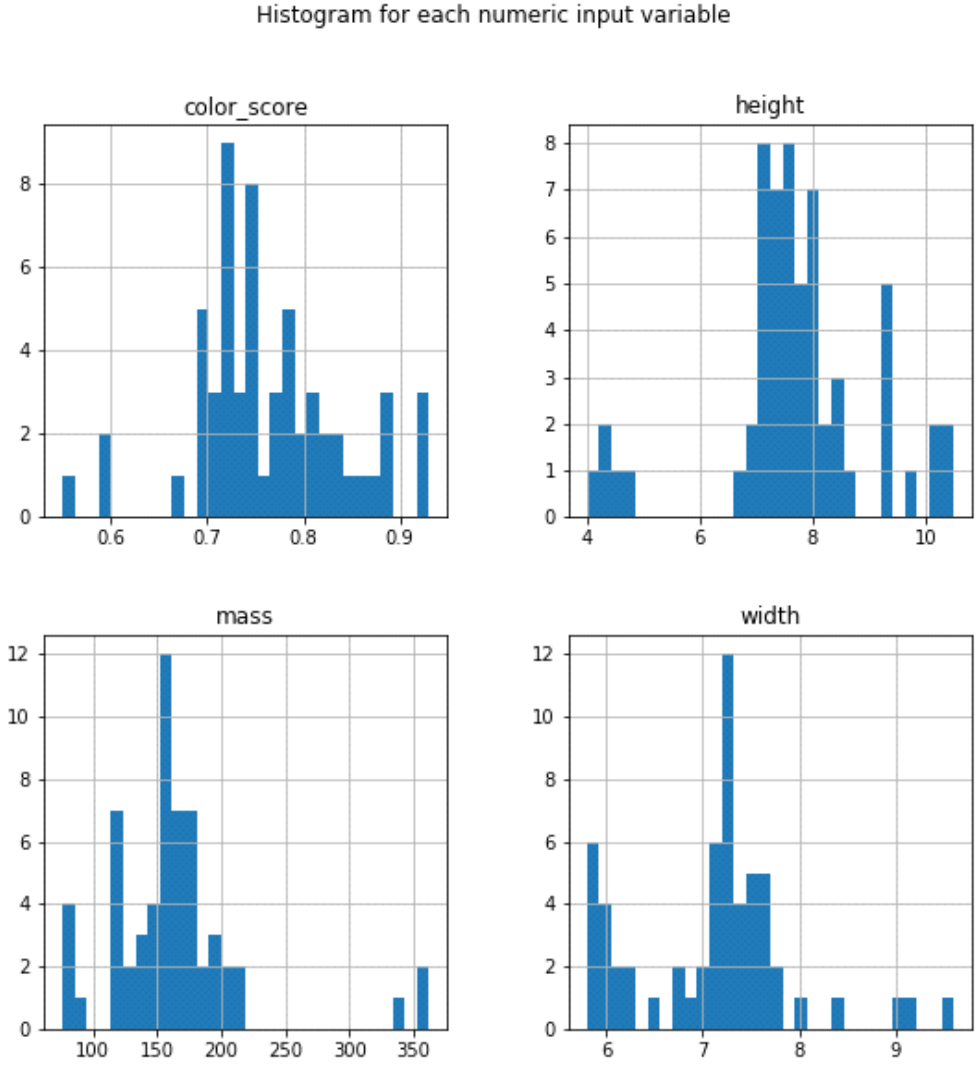
```
%matplotlib inline
import pandas as pd
import matplotlib.pyplot as plt
fruits = pd.read_table('fruit_data_with_colors.txt')
fruits.head()
print(fruits.shape)
print(fruits['fruit_name'].unique())
print(fruits.groupby('fruit_name').size())
import seaborn as sns
sns.countplot(fruits['fruit_name'],label="Count")
plt.show()
fruits.drop('fruit_label', axis=1).plot(kind='box', subplots=True, layout=(2,2),
sharex=False, shar
ey=False, figsize=(9,9),
title='Box Plot for each input variable')
plt.savefig('fruits_box')
plt.show()
import pylab as pl
fruits.drop('fruit_label',axis=1).hist(bins=30, figsize=(9,9))
pl.suptitle("Histogram for each numeric input variable")
plt.savefig('fruits_hist')
plt.show()
import warnings
warnings.filterwarnings("ignore")
from pandas.tools.plotting import scatter_matrix
from matplotlib import cm
feature_names = ['mass', 'width', 'height', 'color_score']
X = fruits[feature_names]
y = fruits['fruit_label']
cmap = cm.get_cmap('gnuplot')
scatter = pd.scatter_matrix(X, c = y, marker = 'o', s=40, hist_kwds={'bins':15},
figsize=(9,9),
cmap = cmap)
plt.suptitle('Scatter-matrix for each input variable')
plt.savefig('fruits_scatter_matrix')
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
from sklearn.linear_model import LogisticRegression
```

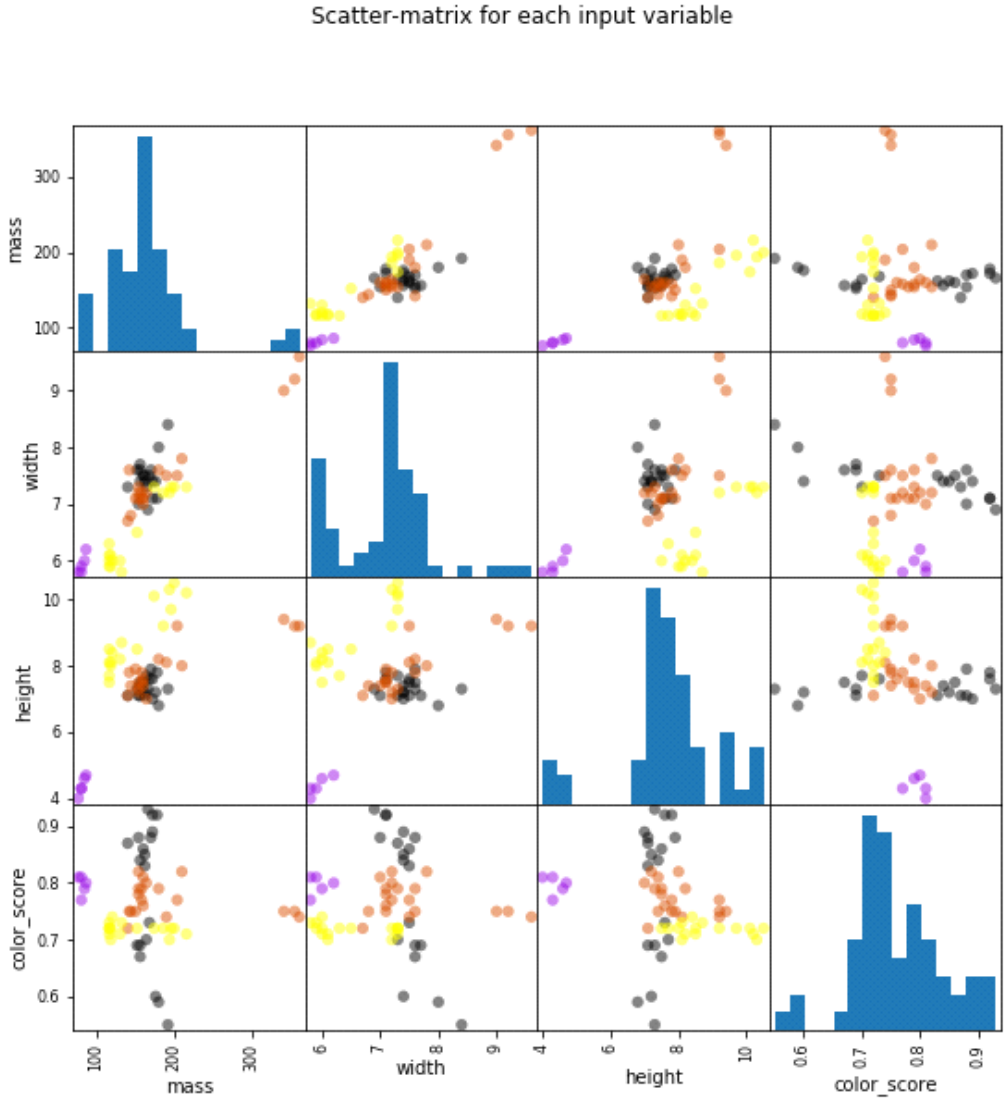
```
logreg = LogisticRegression()
logreg.fit(X_train, y_train)
print('Accuracy of Logistic regression classifier on training set: {:.2f}'
.format(logreg.score(X_train, y_train)))
print('Accuracy of Logistic regression classifier on test set: {:.2f}'
.format(logreg.score(X_test, y_test)))
from sklearn.tree import DecisionTreeClassifier
clf = DecisionTreeClassifier().fit(X_train, y_train)
print('Accuracy of Decision Tree classifier on training set: {:.2f}'
.format(clf.score(X_train, y_train)))
print('Accuracy of Decision Tree classifier on test set: {:.2f}'
.format(clf.score(X_test, y_test)))
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier()
knn.fit(X_train, y_train)
print('Accuracy of K-NN classifier on training set: {:.2f}'
.format(knn.score(X_train, y_train)))
print('Accuracy of K-NN classifier on test set: {:.2f}'
.format(knn.score(X_test, y_test)))
from sklearn.svm import SVC
svm = SVC()
svm.fit(X_train, y_train)
print('Accuracy of SVM classifier on training set: {:.2f}'
.format(svm.score(X_train, y_train)))
print('Accuracy of SVM classifier on test set: {:.2f}'
.format(svm.score(X_test, y_test)))
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
pred = knn.predict(X_test)
print(confusion_matrix(y_test, pred))
print(classification_report(y_test, pred))
```

output:

fruit_label	fruit_name	fruit_subtype	mass	width	height	color_score
1	apple	granny_smith	192	8.4	7.3	0.55
1	apple	granny_smith	180	8.0	6.8	0.59
1	apple	granny_smith	176	7.4	7.2	0.60
2	mandarin	mandarin	86	6.2	4.7	0.80
2	mandarin	mandarin	84	6.0	4.6	0.79




Histogram for each numeric input variable
color_score
height
mass
width



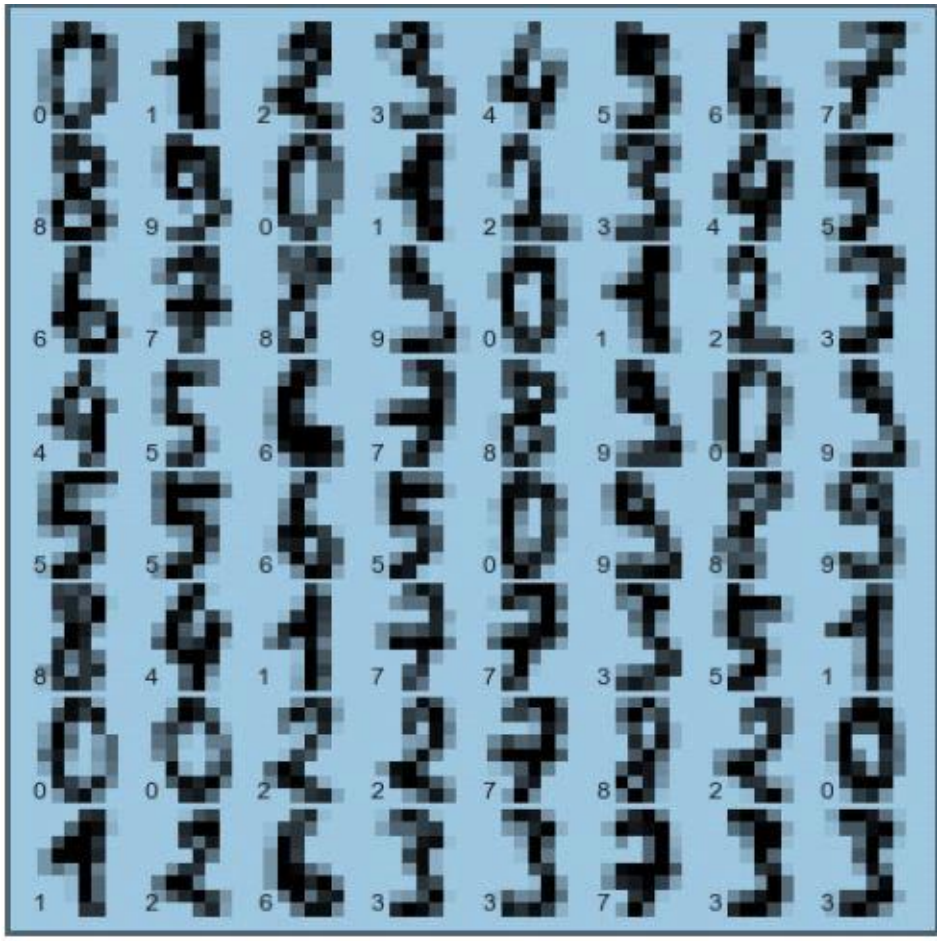
Accuracy of K-NN classifier on training set: 0.95
Accuracy of K-NN classifier on test set: 1.00
Accuracy of SVM classifier on training set: 0.61
Accuracy of SVM classifier on test set: 0.33
[[4 0 0 0]
[0 1 0 0]
[0 0 8 0]
[0 0 0 2]]
precision recall f1-score support
1 1.00 1.00 1.00 4
2 1.00 1.00 1.00 1
3 1.00 1.00 1.00 8 4 1.00 1.00 1.00 2 avg / total 1.00 1.00 1.00 15

6. Write a program to implement Random Forest for Classifying Digits

Sourcecode:

```
from sklearn.datasets import load_digits
digits = load_digits()
digits.keys()
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns; sns.set()
# set up the figure
fig = plt.figure(figsize=(6, 6)) # figure size in inches
fig.subplots_adjust(left=0, right=1, bottom=0, top=1, hspace=0.05, wspace=0.05)
for i in range(64):
ax = fig.add_subplot(8, 8, i + 1, xticks=[], yticks=[])
ax.imshow(digits.images[i], cmap=plt.cm.binary, interpolation='nearest')
ax.text(0, 7, str(digits.target[i]))
import warnings
warnings.filterwarnings("ignore")
from sklearn.ensemble import RandomForestClassifier
from sklearn.cross_validation import train_test_split
Xtrain, Xtest, ytrain, ytest = train_test_split(digits.data, digits.target, random_state=0)
model = RandomForestClassifier(n_estimators=1000)
model.fit(Xtrain, ytrain)
ypred = model.predict(Xtest)
from sklearn import metrics
print(metrics.classification_report(ypred, ytest))
from sklearn.metrics import confusion_matrix
mat = confusion_matrix(ytest, ypred)
sns.heatmap(mat.T, square=True, annot=True, fmt='d', cbar=False)
plt.xlabel('true label')
plt.ylabel('predicted label');
```

Output:

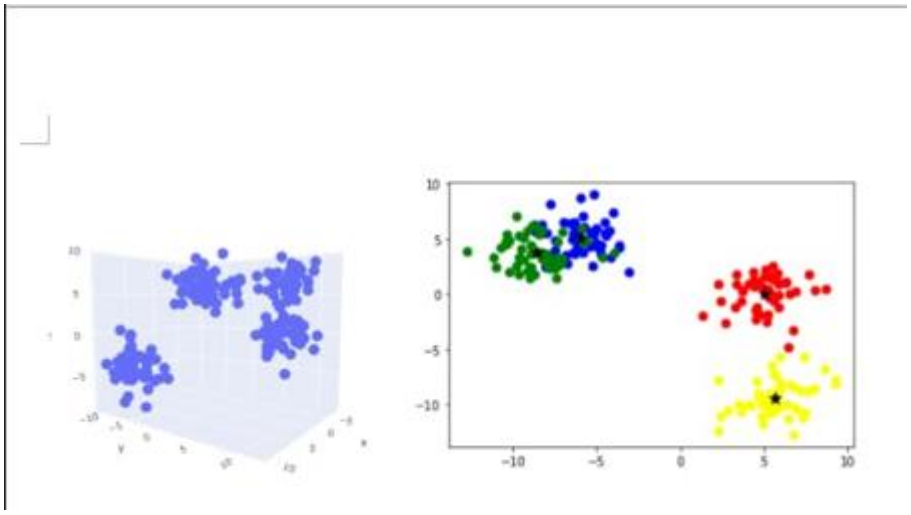


	precision	recall	f1-score	support
	1.00	0.97	0.99	38
	1.00	0.98	0.99	44
	0.95	1.00	0.98	42
	0.98	0.98	0.98	45
	0.97	1.00	0.99	37
	0.98	0.98	0.98	48
	1.00	1.00	1.00	52
	1.00	0.96	0.98	50
	0.96	0.98	0.97	47
	9 0.98	0.98	0.98	47
avg / total	0.98	0.98	0.98	450

7. Write a program to implement K-means Clustering Example

Sourcecode:

```
//Importing Packages
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
//Creating a dataset with size, dimensions, centroids and standard deviation
dataset=make_blobs(n_samples=200,        n_features=2.centers=4,cluster_std=1,6,
random_state=None)
points = dataset [0]
//Import KMeans Clustering from sklearn.cluster
Import Means kmeans=KMeans(n_clusters-4) kmeans.fit(points)
//Plotting Points(optional)
plt.scatter(dataset[0][0,0],dataset[O][:11)
 //Plotting Centroids
clusters = kmeans.cluster_centers_
print(clusters)
//Predicting Points
a=kmeans.fit_predict(points)
//Plotting Points
plt.scatter(points[a==0,01.points[a==0,1),s=50,color="red")
plt.scatter(points[a== 1,01 points[a==1,1],s=50,color="blue")
plt.scatter(points[a==2,01.points[a==2,1),s=50,color="green")
plt.scatter(points[a==3.01.points[a==3,1),s=50,color="yellow")
plt.scatter(clusters[0][0].clusters[0][1].s=100.color="black", marker=*")
plt.scatter(clusters[1][0].clusters[1][1],s=100,color="black", marker="*")
plt.scatter(clusters[2][0].clusters[2][1].s=100.color="black",marker="*")
plt.scatter(clusters[3][0].clusters[3][1],s=100.color="black",marker="*")
//For 3D Model
import plotly express as ex
b=ex.scatter_3d(x=points[0,0],y=points[:O],z=points[0])
b.show()
```


10
5
0
−5
−10
−10
−5
0
5
10

8 . Write a program to implement KNN algorithms with example

Source code:

//Imports and datasets

```
import pandas as pd
from sklearn.datasets import load_iris
iris=load_iris()
```

//To check for column constraints of the dataset

```
iris.feature_names
```

//Checking for the target(names of the clusters for KNN)

```
iris.target_names
```

//Dataframe

```
df=pd.DataFrame(iris.data,columns=iris.feature_names)
df.head()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

//Identifying our clusters

```
df['target']=iris.target
df['flower_name'] =df.target.apply(lambda x: iris.target_names[x])
df.head()
df[df.target==0].head()
df[df.target==1].head()
df[df.target==2].head()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target	flower_name
100	6.3	3.3	6.0	2.5	2	virginica
101	5.8	2.7	5.1	1.9	2	virginica
102	7.1	3.0	5.9	2.1	2	virginica
103	6.3	2.9	5.6	1.8	2	virginica
104	6.5	3.0	5.8	2.2	2	virginica

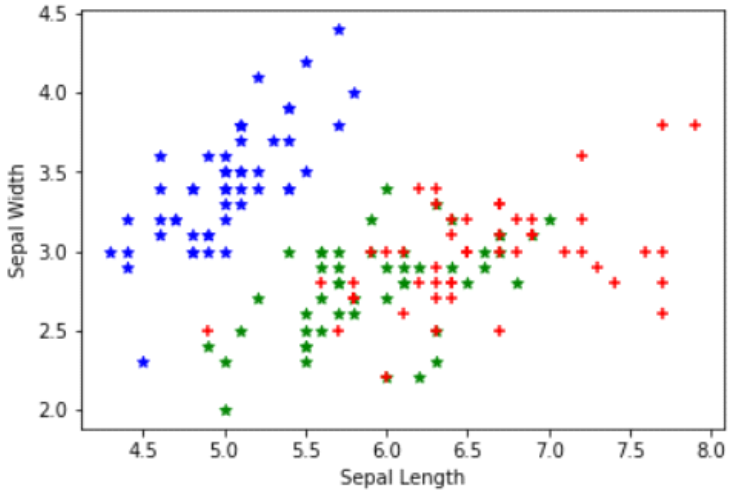
//Grouping our clusters into different dataframes

```
df0=df[:50]
```

```
df1=df[50:100]
df2=df[100:]
```

//Plotting

```
import matplotlib.pyplot as plt
%matplotlib inline
X=df.drop(['target','flower_name'],axis='columns')
y=df.target

plt.xlabel('Sepal Length')
plt.ylabel('Sepal Width')
plt.scatter(df0['sepal length (cm)'],df0['sepal width (cm)'],color="blue",marker='*')
plt.scatter(df1['sepal length (cm)'],df1['sepal width (cm)'],color="green",marker='*')
plt.scatter(df2['sepal length (cm)'],df2['sepal width (cm)'],color="red",marker='+')

plt.show()
```



//Predicting values

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.9,random_state=1)
from sklearn.neighbors import KNeighborsClassifier
knn=KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train,y_train)
knn.score(X_test,y_test)
from sklearn.metrics import confusion_matrix
y_pred=knn.predict(X_test)
cm=confusion_matrix(y_test,y_pred)
print(cm)
```

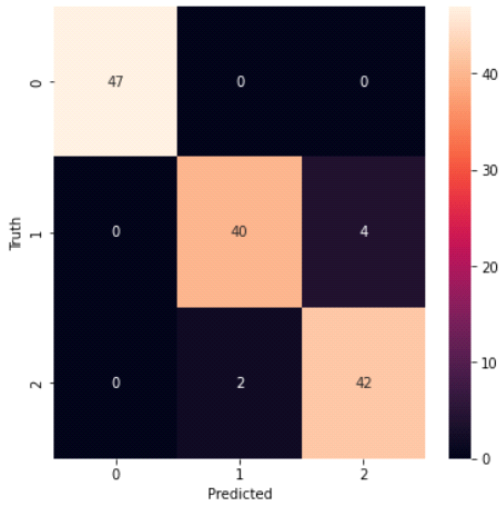
//Confusion matrix

```
[[47  0  0]
 [ 0 40  4]
 [ 0  2 42]]
```

//Heatmap

```
import seaborn as sn
plt.figure(figsize=(6,6))
sn.heatmap(cm, annot=True)
plt.xlabel('Predicted')
plt.ylabel('Truth')
```

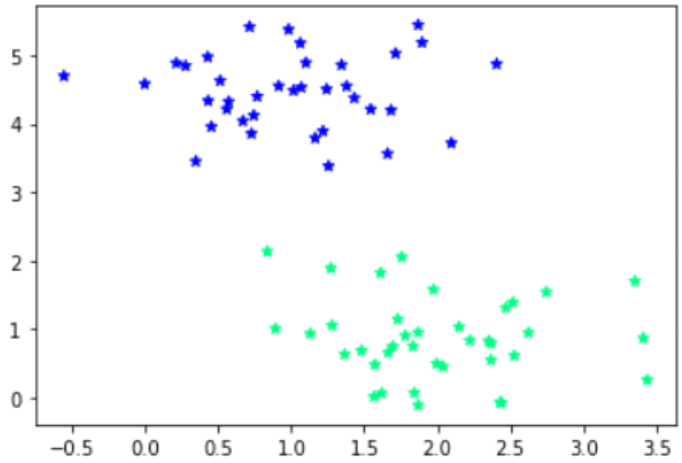
```
Text(33.0, 0.5, 'Truth')
```

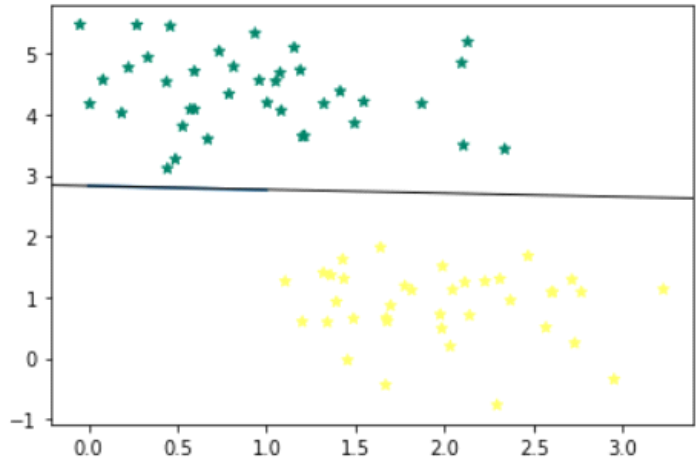


9. Write a program to implement SVM Example
Source code:

```
//Imports
from sklearn.datasets import make_blobs
import matplotlib.pyplot as plt
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.svm import SVC
//Initializing the dataset
X,y=make_blobs(n_samples=150,random_state=0,cluster_std=0.60,centers=2)
//Training and testing data
X_train , X_test , y_train , y_test=train_test_split(X, y ,
test_size=75,random_state=0)
y = y.reshape(y.shape[0])
plt.scatter(X_train[:,0], X_train[:,1],c=y_train,cmap='winter',marker='*')
plt.show()
//calling  SVM model
svc=SVC(kernel='linear')
svc.fit(X_train,y_train)
//plotting  axes
ax=plt.gca()
xlim=ax.get_xlim()
ax.scatter(X_test[:,0], X_test[:,1],c=y_test,cmap='summer',marker='*')
//Linear hyperplane and line equation
w=svc.coef_[0]
a=-w[0]/w[1]
xx=np.linspace(xlim[0],xlim[1])
yy=a * xx -(svc.intercept_[0]/w[1])
plt.plot(xx,yy)
plt.axline((0, 2.83),slope=-0.06, linewidth=1, color='black')
plt.show()
```

OUTPUT:





10. Write a program to Implementing Neural Network

Source code:

```
import numpy as np
X=np.array([[1,0,1,0],[1,0,1,1],[0,1,0,1]])
y=np.array([[1],[1],[0]])
def sigmoid (x):
return 1/(1 + np.exp(-x))
def derivatives_sigmoid(x):
return x * (1 - x)
epoch=5000 #Setting training iterations
lr=0.1 #Setting learning rate
inputlayer_neurons = X.shape[1] #number of features in data set
hiddenlayer_neurons = 3 #number of hidden layers neurons
output_neurons = 1 #number of neurons at output layer
wh=np.random.uniform(size=(inputlayer_neurons,hiddenlayer_neurons))
bh=np.random.uniform(size=(1,hiddenlayer_neurons))
wout=np.random.uniform(size=(hiddenlayer_neurons,output_neurons))
bout=np.random.uniform(size=(1,output_neurons))
for i in range(epoch):
hidden_layer_input1=np.dot(X,wh)
hidden_layer_input=hidden_layer_input1 + bh
hiddenlayer_activations = sigmoid(hidden_layer_input)
output_layer_input1=np.dot(hiddenlayer_activations,wout)
output_layer_input= output_layer_input1+ bout
output = sigmoid(output_layer_input)
E = y-output
slope_output_layer = derivatives_sigmoid(output)
slope_hidden_layer = derivatives_sigmoid(hiddenlayer_activations)
d_output = E * slope_output_layer
Error_at_hidden_layer = d_output.dot(wout.T)
d_hiddenlayer = Error_at_hidden_layer * slope_hidden_layer
wout += hiddenlayer_activations.T.dot(d_output) *lr
bout += np.sum(d_output, axis=0,keepdims=True) *lr
wh += X.T.dot(d_hiddenlayer) *lr
bh += np.sum(d_hiddenlayer, axis=0,keepdims=True) *lr
print (output)
```

output:

y=np.array([[1],[1],[0]])

[[0.96962029] [0.96488635] [0.0601801]]

11. Write a program to implement PCA Example

Source code:

//Imports

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import load_iris
iris=load_iris()
```

//Dataset

```
X=pd.DataFrame(iris.data,columns=iris.feature_names)
y=pd.Categorical.from_codes(iris.target,iris.target_names)
X.head()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

//Scaling data

```
scaler=StandardScaler()
X=scaler.fit_transform(X)
```

//Applying PCA

```
pca=PCA(n_components=2)
principal_components=pca.fit_transform(X)
new_X=pd.DataFrame(principal_components,columns=['PC1','PC2'])
new_X
```

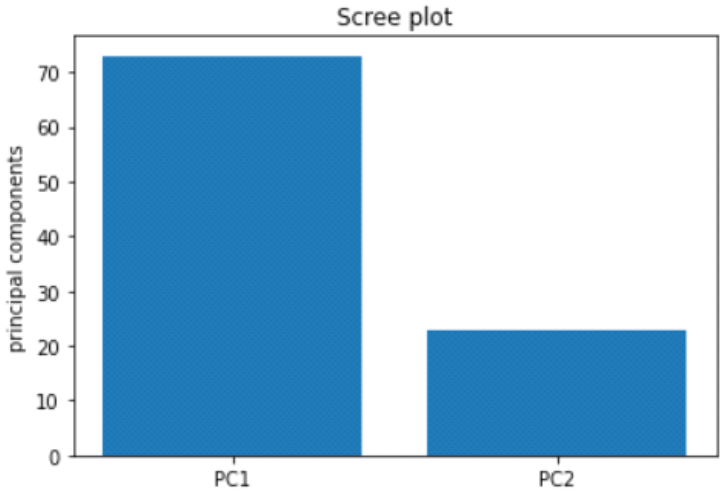
//Variance

```
per_var=np.round(pca.explained_variance_ratio_*100,decimals=1)
labels=['PC'+str(x) for x in range(1,len(per_var)+1)]
```

//Bar Graph

```
plt.bar(x=range(1,len(per_var)+1),height=per_var,tick_label=labels)
plt.ylabel('Percentage of Variance')
plt.ylabel('principal components')
plt.title('Scree plot')
plt.show()
```

OUTPUT:



12. Write a program to implement anomaly detection and recommendation

Source code:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sb
from scipy.io import loadmat
%matplotlib inline
data = loadmat('data/ex8data1.mat')
X = data['X']
X.shape
fig, ax = plt.subplots(figsize=(12,8))
ax.scatter(X[:,0], X[:,1])
def estimate_gaussian(X):
mu = X.mean(axis=0)
sigma = X.var(axis=0)
return mu, sigma
mu, sigma = estimate_gaussian(X)
mu, sigma
Xval = data['Xval']
yval = data['yval']
Xval.shape, yval.shape
from scipy import stats
dist = stats.norm(mu[0], sigma[0])
dist.pdf(X[:,0])[0:50]
p = np.zeros((X.shape[0], X.shape[1]))
p[:,0] = stats.norm(mu[0], sigma[0]).pdf(X[:,0])
p[:,1] = stats.norm(mu[1], sigma[1]).pdf(X[:,1])
p.shape
pval = np.zeros((Xval.shape[0], Xval.shape[1]))
pval[:,0] = stats.norm(mu[0], sigma[0]).pdf(Xval[:,0])
pval[:,1] = stats.norm(mu[1], sigma[1]).pdf(Xval[:,1])
def select_threshold(pval, yval):
best_epsilon = 0
best_f1 = 0
f1 = 0
step = (pval.max() - pval.min()) / 1000
for epsilon in np.arange(pval.min(), pval.max(), step):
preds = pval <epsilon
tp = np.sum(np.logical_and(preds == 1, yval == 1)).astype(float)
fp = np.sum(np.logical_and(preds == 1, yval == 0)).astype(float)
fn = np.sum(np.logical_and(preds == 0, yval == 1)).astype(float)
precision = tp / (tp + fp)
```

```
recall = tp / (tp + fn)
f1 = (2 * precision * recall) / (precision + recall)
if f1 >best_f1:
if f1 >best_f1:
best_f1 = f1
best_epsilon = epsilon
return best_epsilon, best_f1
epsilon, f1 = select_threshold(pval, yval)
epsilon, f1
# indexes of the values considered to be outliers
outliers = np.where(p <epsilon)
fig, ax = plt.subplots(figsize=(12,8))
ax.scatter(X[:,0], X[:,1])
ax.scatter(X[outliers[0],0], X[outliers[0],1], s=50, color='r', marker='o')
# Collaborative Filtering
data = loadmat('data/ex8_movies.mat')
data
Y = data['Y']
R = data['R']
Y.shape, R.shape
Y[1,R[1,:]].mean()
fig, ax = plt.subplots(figsize=(12,12))
ax.imshow(Y)
ax.set_xlabel('Users')
ax.set_ylabel('Movies')
fig.tight_layout()
def cost(params, Y, R, num_features):
Y = np.matrix(Y) # (1682, 943)
R = np.matrix(R) # (1682, 943)
num_movies = Y.shape[0]
num_users = Y.shape[1]
# reshape the parameter array into parameter matrices
X = np.matrix(np.reshape(params[:num_movies * num_features], (num_movies,
num_features))) # (1
682, 10)
Theta = np.matrix(np.reshape(params[num_movies * num_features:],
(num_users, num_features))) #
(943, 10)
# initializations
J = 0
# compute the cost
error = np.multiply((X * Theta.T) - Y, R) # (1682, 943)
squared_error = np.power(error, 2) # (1682, 943)
J = (1. / 2) * np.sum(squared_error)
return J
users = 4
```

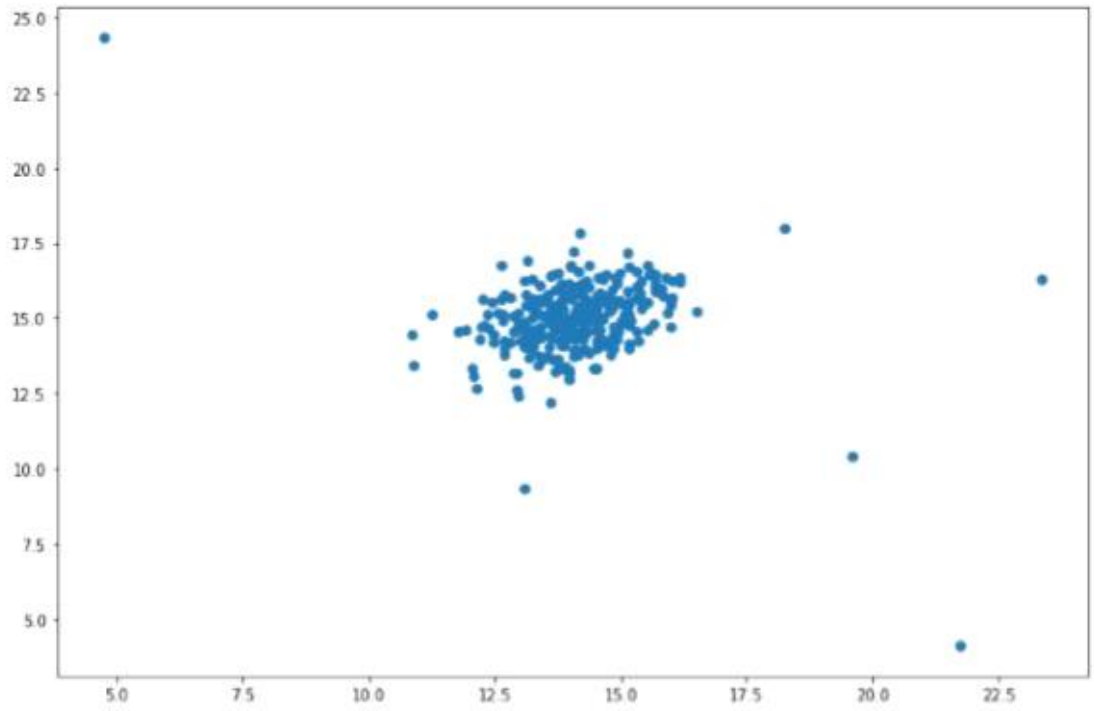
```
movies = 5
features = 3
params_data = loadmat('data/ex8_movieParams.mat')
X = params_data['X']
Theta = params_data['Theta']
X_sub = X[:movies, :features]
Theta_sub = Theta[:users, :features]
Y_sub = Y[:movies, :users]
R_sub = R[:movies, :users]
params = np.concatenate((np.ravel(X_sub), np.ravel(Theta_sub)))
cost(params, Y_sub, R_sub, features)
def cost(params, Y, R, num_features):
Y = np.matrix(Y) # (1682, 943)
R = np.matrix(R) # (1682, 943)
num_movies = Y.shape[0]
num_users = Y.shape[1]
X = np.matrix(np.reshape(params[:num_movies * num_features], (num_movies,
num_features))) # (1
682, 10)
Theta = np.matrix(np.reshape(params[num_movies * num_features:],
(num_users, num_features))) #
(943, 10)
# initializations
J = 0
X_grad = np.zeros(X.shape) # (1682, 10)
Theta_grad = np.zeros(Theta.shape) # (943, 10)
# compute the cost
error = np.multiply((X * Theta.T) - Y, R) # (1682, 943)
squared_error = np.power(error, 2) # (1682, 943)
J = (1. / 2) * np.sum(squared_error)
# calculate the gradients
X_grad = error * Theta
Theta_grad = error.T * X
# unravel the gradient matrices into a single array
grad = np.concatenate((np.ravel(X_grad), np.ravel(Theta_grad)))
return J, grad
J, grad = cost(params, Y_sub, R_sub, features)
J, grad
def cost(params, Y, R, num_features, learning_rate):
Y = np.matrix(Y) # (1682, 943)
R = np.matrix(R) # (1682, 943)
num_movies = Y.shape[0]
num_users = Y.shape[1]
# reshape the parameter array into parameter matrices
X = np.matrix(np.reshape(params[:num_movies * num_features], (num_movies,
num_features))) # (1
```

```
682, 10)
Theta = np.matrix(np.reshape(params[num_movies * num_features:],
(num_users, num_features))) #
(943, 10)
# initializations
J = 0
X_grad = np.zeros(X.shape) # (1682, 10)
Theta_grad = np.zeros(Theta.shape) # (943, 10)
# compute the cost
error = np.multiply((X * Theta.T) - Y, R) # (1682, 943)
squared_error = np.power(error, 2) # (1682, 943)
J = (1. / 2) * np.sum(squared_error)
# add the cost regularization
J = J + ((learning_rate / 2) * np.sum(np.power(Theta, 2)))
J = J + ((learning_rate / 2) * np.sum(np.power(X, 2)))
# calculate the gradients with regularization
X_grad = (error * Theta) + (learning_rate * X)
Theta_grad = (error.T * X) + (learning_rate * Theta)
# unravel the gradient matrices into a single array
grad = np.concatenate((np.ravel(X_grad), np.ravel(Theta_grad)))
return J, grad
movie_idx = {}
f = open('data/movie_ids.txt')
for line in f:
tokens = line.split('')
tokens[-1] = tokens[-1][:-1]
movie_idx[int(tokens[0]) - 1] = ''.join(tokens[1:])
ratings = np.zeros((1682, 1))
ratings[0] = 4
ratings[6] = 3
ratings[11] = 5
ratings[53] = 4
ratings[63] = 5
ratings[65] = 3
ratings[68] = 5
ratings[97] = 2
ratings[182] = 4
ratings[225] = 5
ratings[354] = 5
print('Rated {0} with {1} stars.'.format(movie_idx[0], str(int(ratings[0]))))
print('Rated {0} with {1} stars.'.format(movie_idx[6], str(int(ratings[6]))))
print('Rated {0} with {1} stars.'.format(movie_idx[11], str(int(ratings[11]))))
print('Rated {0} with {1} stars.'.format(movie_idx[53], str(int(ratings[53]))))
print('Rated {0} with {1} stars.'.format(movie_idx[63], str(int(ratings[63]))))
print('Rated {0} with {1} stars.'.format(movie_idx[65], str(int(ratings[65]))))
print('Rated {0} with {1} stars.'.format(movie_idx[68], str(int(ratings[68]))))
```

```
print('Rated {0} with {1} stars.'.format(movie_idx[97], str(int(ratings[97]))))
print('Rated {0} with {1} stars.'.format(movie_idx[182], str(int(ratings[182]))))
print('Rated {0} with {1} stars.'.format(movie_idx[225], str(int(ratings[225]))))
print('Rated {0} with {1} stars.'.format(movie_idx[354], str(int(ratings[354]))))
R = data['R']
Y = data['Y']
Y = np.append(Y, ratings, axis=1)
R = np.append(R, ratings != 0, axis=1)
from scipy.optimize import minimize
movies = Y.shape[0]
users = Y.shape[1]
features = 10
learning_rate = 10.
X = np.random.random(size=(movies, features))
Theta = np.random.random(size=(users, features))
params = np.concatenate((np.ravel(X), np.ravel(Theta)))
Ymean = np.zeros((movies, 1))
Ynorm = np.zeros((movies, users))
for i in range(movies):
idx = np.where(R[i,:] == 1)[0]
Ymean[i] = Y[i,idx].mean()
Ynorm[i,idx] = Y[i,idx] - Ymean[i]
fmin = minimize(fun=cost, x0=params, args=(Ynorm, R, features, learning_rate),
method='CG', jac=True, options={'maxiter': 100})
fmin
X = np.matrix(np.reshape(fmin.x[:movies * features], (movies, features)))
Theta = np.matrix(np.reshape(fmin.x[movies * features:], (users, features)))
X.shape, Theta.shape
predictions = X * Theta.T
my_preds = predictions[:, -1] + Ymean
sorted_preds = np.sort(my_preds, axis=0)[::-1]
sorted_preds[:10]
idx = np.argsort(my_preds, axis=0)[::-1]
print("Top 10 movie predictions:")
for i in range(10):
j = int(idx[i])
print('Predicted rating of {0} for movie {1}.'
.format(str(float(my_preds[j])), movie_idx[j]))
```

Output:

(307, 2)



(array ([14.11222578, 14.99771051]), array([1.83263141, 1.70974533]))

((307, 2), (307, 1))

array([0.183842 , 0.20221694, 0.21746136, 0.19778763, 0.20858956, 0.21652359, 0.16991291, 0.15123542, 0.1163989 , 0.1594734 , 0.21716057, 0.21760472, 0.20141857, 0.20157497, 0.21711385, 0.21758775, 0.21695576, 0.2138258 , 0.21057069, 0.1173018 , 0.20765108, 0.21717452, 0.19510663, 0.21702152, 0.17429399, 0.15413455, 0.21000109, 0.20223586, 0.21031898, 0.21313426, 0.16158946, 0.2170794 , 0.17825767, 0.17414633, 0.1264951 , 0.19723662, 0.14538809, 0.21766361, 0.21191386, 0.21729442, 0.21238912, 0.18799417, 0.21259798, 0.21752767, 0.20616968, 0.21520366, 0.1280081 , 0.21768113, 0.21539967, 0.16913173])

(307, 2)

(0.009566706005956842, 0.7142857142857143)

{'__header__': b'MATLAB 5.0 MAT-file, Platform: GLNXA64, Created on: Thu Dec 1 17:19:26 2011', '__version__': '1.0', '__globals__': [], 'Y': array([[5, 4, 0, ..., 5, 0, 0],

In [14]: Y = data['Y'] R = data['R'] Y.shape, R.shape

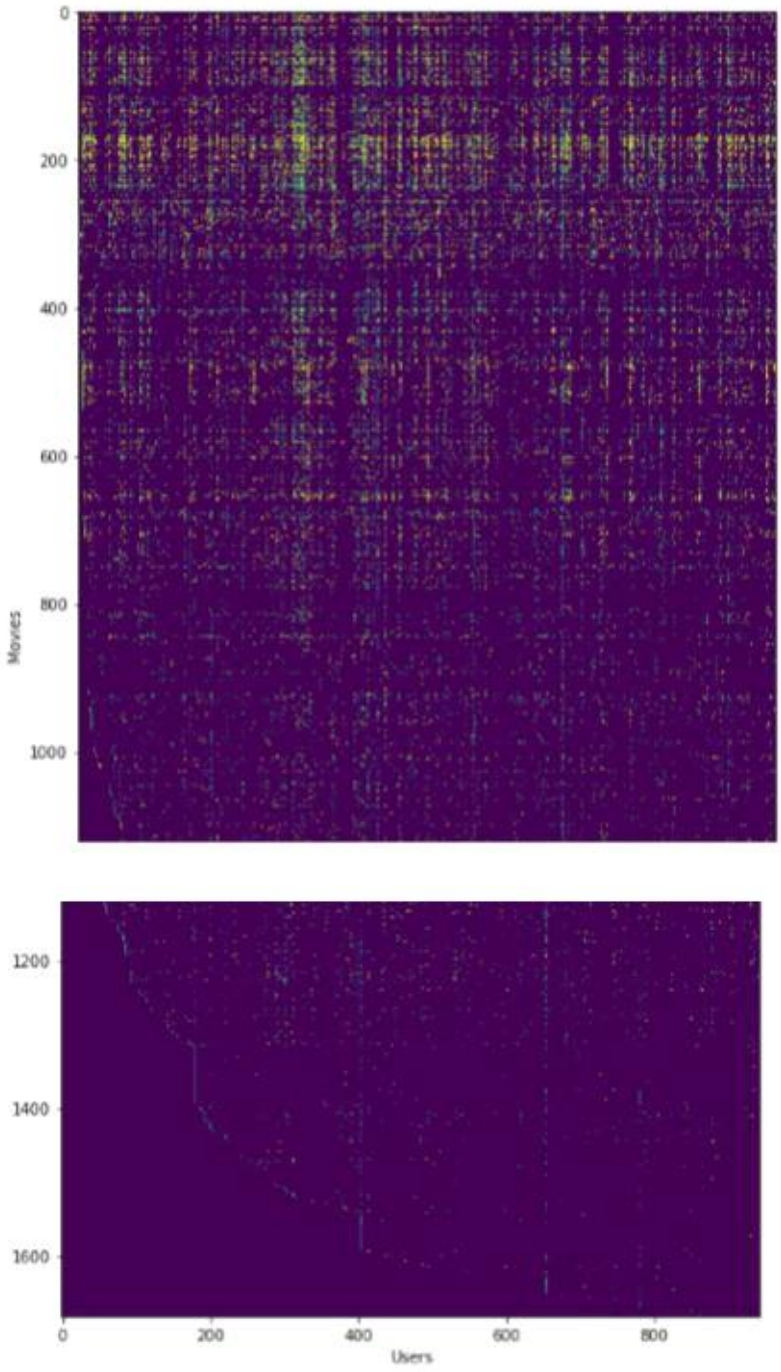
In [15]: Y[1,R[1,:]].mean()

In [16]: fig, ax = plt.subplots(figsize=(12,12)) ax.imshow(Y) ax.set_xlabel('Users') ax.set_ylabel('Movies') fig.tight_layout()

'Y': array([[5, 4, 0, ..., 5, 0, 0], [3, 0, 0, ..., 0, 0, 5], [4, 0, 0, ..., 0, 0, 0], ..., [0, 0, 0, ..., 0, 0, 0], [0, 0, 0, ..., 0, 0, 0], [0, 0, 0, ..., 0, 0, 0]], dtype=uint8), 'R': array([[1, 1, 0, ..., 1, 0, 0], [1, 0, 0, ..., 0, 0, 1], [1, 0, 0, ..., 0, 0, 0], ..., [0, 0, 0, ..., 0, 0, 0], [0, 0, 0, ..., 0, 0, 0], [0, 0, 0, ..., 0, 0, 0]], dtype=uint8)}

((1682, 943), (1682, 943))

2.5832449628844114



22.224603725685675

(22.224603725685675, array([-2.52899165, 7.57570308, -1.89979026, -0.56819597, 3.35265031, -0.52339845, -0.83240713, 4.91163297, -0.76677878, -0.38358278, 2.26333698, -0.35334048, -0.80378006, 4.74271842, -0.74040871, -10.5680202 , 4.62776019, -7.16004443, -3.05099006, 1.16441367, -3.47410789, 0. , 0. , 0. , 0. , 0. , 0.]))

(31.34405624427422, array([-0.95596339, 6.97535514, -0.10861109, 0.60308088, 2.77421145, 0.25839822, 0.12985616, 4.0898522 , -0.89247334, 0.29684395, 1.06300933, 0.66738144, 0.60252677, 4.90185327, -0.19747928, -10.13985478, 2.10136256, -6.76563628, -2.29347024, 0.48244098, -2.99791422, -0.64787484, -0.71820673, 1.27006666, 1.09289758, -0.40784086, 0.49026541]))

fun: 38978.314384647194 jac: array([-0.04421097, -0.00308654, -0.00752965, ..., -0.00237255, 0.00605155, 0.0068428]) message: 'Maximum number of iterations has been exceeded.' nfev: 155 nit: 100 njev: 155 status: 1 success: False x: array([0.22064958, 0.96957432, -0.47342215, ..., -0.08203747, 0.08609211, -0.04624869])

((1682, 10), (944, 10))

Matrix ([[5.00003544], [5.0000234], [5.00002157], [5.00001632], [5.00001607], [5.00000945], [5.0000089], [5.00000706], [5.0000061], [4.99998294]])

Top 10 movie predictions: Predicted rating of 5.000035439531592 for movie Marlene Dietrich: Shadow and Light (1996) . Predicted rating of 5.000023400858388 for movie Great Day in Harlem, A (1994). Predicted rating of 5.000021573580529 for movie They Made Me a Criminal (1939). Predicted rating of 5.0000163160578275 for movie Star Kid (1997). Predicted rating of 5.000016070020329 for movie Someone Else's America (1995). Predicted rating of 5.000009445486224 for movie Saint of Fort Washington, The (1993). Predicted rating of 5.000008895567655 for movie Entertaining Angels: The Dorothy Day Story (1996). Predicted rating of 5.000007062141572 for movie Aiqing wansui (1994). Predicted rating of 5.000006098541294 for movie Santa with Muscles (1996). Predicted rating of 4.99998294111069 for movie Prefontaine (1997).