|  |  |
| --- | --- |
| **An Autonomous Institute**  (Approved By AICTE, New Delhi Affiliated To VTU, Belagavi  Recognized By UGC Under 2(f) & 12(B) Accredited By NBA & NAAC) | C:\Users\Sudhakar\Desktop\NBA.jpg |

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

**VII SEMESTER**

**MVJ19CSL77**

MACHINE LEARNING LABORATORY ACADEMIC YEAR 2022 – 2023 [ODD]

***LABORATORY MANUAL***

|  |  |
| --- | --- |
| NAME OF THE STUDENT | : |
| BRANCH | : |
| UNIVERSITY SEAT NO. | : |
| SEMESTER & SECTION | : |
| BATCH | : |

# DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

**VISION:**

To create an ambiance in excellence and provide innovative emerging programs in Computer Science and Engineering and to bring out future ready engineers equipped with technical expertise and strong ethical values.

# MISSION:

1. **Concepts of Computing Discipline:** To educate students at under graduate, postgraduate and doctoral levels in the fundamental and advanced concepts of computing discipline.
2. **Quality Research:** To provide strong theoretical and practical background across the Computer Science and Engineering discipline with the emphasis on computing technologies, quality research, consultancy and training's.
3. **Continuous Teaching Learning:** To promote a teaching learning process that brings advancements in Computer Science and Engineering discipline leading to new technologies and products.
4. **Social Responsibility and Ethical Values:** To inculcate professional behavior, innovative research Capabilities , leadership abilities and strong ethical values in the young minds so as to work with the commitment for the betterment of the society

# PROGRAM EDUCATIONAL OBJECTIVES (PEOs):

**PEO1: Current Industry Practices:** Graduates will analyze real world problems and give solution using current industry practices in computing technology.

**PEO2: Research and Higher Studies:** Graduates with strong foundation in mathematics and engineering fundamentals that will enable graduates to pursue higher learning, R&D activities and consultancy.

**PEO3: Social Responsibility:** Graduates will be professionals with ethics, who will provide industry growth and social transformation as responsible citizens.

**PEO4: Entrepreneur:** Graduates will be able to become entrepreneur to address social, technical and business challenges.

# PROGRAM OUTCOMES (POs):

1. **Engineering knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **Problem analysis**: Identify, formulate, research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **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.
4. **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.
5. **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.
6. **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.
7. **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.
8. **Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and team work**: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **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.
11. **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.
12. **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.

# PROGRAM SPECIFIC OUTCOMES (PSOs):

**PSO1: Programming:** Ability to understand, analyze and develop computer programs in the areas related to algorithms, system software, multimedia, web design, DBMS, and networking for efficient design of computer-based systems of varying complexity.

**PSO2: Practical Solution:** Ability to practically provide solutions for real world problems with a broad range of programming language and open source platforms in various computing domains.

**PSO3: Research:** Ability to use innovative ideas to do research in various domains to solve societal problems.

# COURSE OBJECTIVES:

* Make use of Data sets in implementing the machine learning algorithms
* Implement the machine learning concepts and algorithms in any suitable language of choice.

# PREREQUISITES:

* Programming experience in Python.
* Knowledge of basic Machine Learning Algorithms.
* Knowledge of common statistical methods and data analysis best practices.

# COURSE OUTCOMES (CO’s):

|  |  |
| --- | --- |
| **CO No** | **CO’s** |
| **C406.1** | Understand the implementation procedures for the machine learning algorithms. |
| **C406.2** | Design Java/Python programs for various Learning algorithms. |
| **C406.3** | Apply appropriate data sets to the Machine Learning algorithms. |
| **C406.4** | Identify and apply Machine Learning algorithms to solve real world problems. |
| **C406.5** | Perform statistical analysis of machine learning techniques. |

**SOFTWARE REQUIREMENT:**

1. Python Version 3.5 and above
2. Machine Learning packages
   1. Scikit –Learn
   2. Numpy – matrices and linear algebra
   3. Scipy – many numerical routines
   4. Matplotlib – creating plots of data
   5. Pandas – facilitates structured / tabular data manipulation and visualizations
   6. Pomegranate – for fast and flexible probabilistic models
3. An Integrated Development Environment (IDE) for Python Programming.
4. For Problems 1 to 6 and 10, programs are to be developed without using the built-in classes or APIs of Java/Python.
5. Data sets can be taken from standard repositories (https://archive.ics.uci.edu/ml/datasets.html) or constructed by the students.

# Anaconda:

Anaconda Navigator is a desktop graphical user interface (GUI) included in Anaconda distribution that allows you to launch applications and easily manage conda packages, environments and channels without using command-line commands. Navigator can search for packages on Anaconda Cloud or in a local Anaconda Repository.

# Operating System:

* Windows / Linux.

# CONTENTS

|  |  |  |  |
| --- | --- | --- | --- |
| **Sl. No.** | **Program** | **RBTL** | **CO** |
| **1** | Implement and demonstrate the **FIND-Algorithm** for finding the  most specific hypothesis based on a given set of training data samples. Read the training data from a .CSV file. | L3 | CO 1,2,3,4 |
| **2** | For a given set of training data examples stored in a .CSV file, implement and demonstrate the **Candidate-Elimination algorithm** to output a description of the set of all hypotheses  consistent with the training examples. | L3 | CO 1,2,3,4 |
| **3** | Develop a program to demonstrate the prediction of values of a given dataset using linear regression. | L3 | CO 1,2,3,4 |
| **4** | Write a program to demonstrate the working of the decision tree based **ID3 algorithm**. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new  sample. | L3 | CO 1,2,3,4 |
| **5** | Build an Artificial Neural Network by implementing the  **Backpropagation algorithm** and test the same using appropriate data sets. | L3 | CO 1,2,3,4 |
| **6** | Write a program to implement the **naïve Bayesian classifier** for a sample training data set stored as a .CSV file. Compute the  accuracy of the classifier, considering few test data sets. | L3 | CO 1,2,3,4 |
| **7** | Assuming a set of documents that need to be classified, use the **naïve Bayesian Classifier** model to perform this task. Built-in Java classes/API can be used to write the program. Calculate the  accuracy, precision, and recall for your data set. | L3 | CO 1,2,3,4,5 |
| **8** | Write a program to construct a **Bayesian network** considering medical data. Use this model to demonstrate the diagnosis of heart patients using standard Heart Disease Data Set. You can  use Java/Python ML library classes/API. | L3 | CO 1,2,3,4 |
| **9** | Apply **EM algorithm** to cluster a set of data stored in a .CSV file. Use the same dataset for clustering using ***k*-Means algorithm**. Compare the results of these two algorithms and comment on the quality of clustering. You can add Java/Python  ML library classes/API in the program. | L3 | CO 1,2,3,4,5 |
| **10** | Write a program to implement ***k*-Nearest Neighbour algorithm** to classify the iris data set. Print both correct and wrong predictions. Java/Python ML library classes can be used for this  problem. | L3 | CO 1,2,3,4,5 |
| **11** | Implement the non-parametric **Locally Weighted Regression**  **algorithm** in order to fit data points. Select appropriate data set for your experiment and draw graphs. | L3 | CO 1,2,3,4,5 |

**Mapping of Course Outcomes to Program Outcomes:**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **PO1** | **PO2** | **PO3** | **PO4** | **PO5** | **PO6** | **PO7** | **PO8** | **PO9** | **PO10** | **PO11** | **PO12** | **PSO1** | **PSO2** | **PSO3** |
| **C406.1** | **3** | **3** | **2** | **2** | **2** | **1** | **1** | **-** | **3** | **3** | **2** | **1** | **3** | **3** | **2** |
| **C406.2** | **3** | **3** | **2** | **2** | **3** | **1** | **-** | **-** | **3** | **2** | **1** | **1** | **3** | **3** | **2** |
| **C406.3** | **3** | **3** | **3** | **2** | **2** | **2** | **-** | **-** | **3** | **1** | **-** | **-** | **3** | **3** | **2** |
| **C407.4** | **3** | **3** | **2** | **2** | **3** | **3** | **1** | **2** | **3** | **3** | **2** | **1** | **3** | **3** | **2** |
| **C406.5** | **3** | **3** | **3** | **2** | **2** | **2** | **1** | **1** | **1** | **2** | **1** | **1** | **3** | **3** | **2** |

# Direct Assessment

* 1. **Continuous:**
     1. 2 Internal Assessment Tests→ [Average of 2 tests] -- Max. Marks 10
     2. Record work and Weekly evaluation → Max. Marks 10
     3. Total IA Marks → 1+2 (20 Marks)

# Semester End Assessment:

Semester end lab examination evaluated for 80 Marks.

* 1. **Total Marks :** a + b = 100 Marks

# Indirect Assessment

Indirect Assessment is through course exit survey

# Conduction of Practical Examination:

* All laboratory experiments are to be included for practical examination.
* Students are allowed to pick one experiment from the lot.
* Strictly follow the instructions as printed on the cover page of answer script
* Marks distribution: Procedure + Conduction + Viva: 20 + 50 +10 (80)

**Change of experiment is allowed only once and marks allotted to the procedure part to be made zero.**

# Program No: 1 : FIND-S ALGORITHM

Implement and demonstrate the **FIND-Algorithm** for finding the most specific hypothesis based on a given set of training data samples. Read the training data from a .CSV file.

|  |
| --- |
| **Objective**  To implement and demonstrate the **FIND-Algorithm** for finding the most specific hypothesis based on a given set of training data samples. |
| **Data Set**  *Tennis data set*: This data set contains the set of examples days on which playing of tennis is possible or not, based on attributes Sky, AirTemp, Humidity, Wind, Water and Forecast. |
| **ML Algorithm**  Supervised Learning – **FIND-Algorithm** |
| **Description**   * The concept learning approach in machine learning, can be formulated as “Problem of searching through a predefined space of potential hypotheses for the hypothesis that best fits the training examples”. * Find-S algorithm for concept learning is one of the most basic algorithms of machine learning.   Find-S Algorithm   1. Initialize h to the most specific hypothesis in H 2. For each positive training instance x   For each attribute constraint a i in h :  If the constraint a i in h is satisfied by x then do nothing  Else replace a i in h by the next more general constraint that is satisfied by x   1. Output hypothesis h  * It is Guaranteed to output the most specific hypothesis within H that is consistent with the positive training examples. * Also Notice that negative examples are ignored. Limitations of the Find-S algorithm: * No way to determine if the only final hypothesis (found by Find-S) is consistent with data or there are more hypothesis that is consistent with data. * Inconsistent sets of training data can mislead the finds algorithm as it ignores negative data samples. * A good concept learning algorithm should be able to backtrack the choice of hypothesis found so that the resulting hypothesis can be improved over time. Unfortunately, Find-S   provide no such method. |

# Program:

import csv

with open('P1\_data.csv', 'r') as f: reader = csv.reader(f)

headers = next(reader) your\_list = list(reader)

h = [['0', '0', '0', '0', '0', '0']]

for i in your\_list: print(i)

if i[-1] == "TRUE":

j = 0

for x in i:

if x != "TRUE":

if x != h[0][j] and h[0][j] == '0': h[0][j] = x

elif x != h[0][j] and h[0][j] != '0': h[0][j] = '?'

else:

pass j = j + 1

print("The maximally specific hypothesis for a given training example is: ") print(h)

# Dataset: P1\_data.csv File

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Sky** | **AirTemp** | **Humidity** | **Wind** | **Water** | **Forcast** | **EnjoySpor** |
| 'Sunny' | 'Warm' | 'Normal' | 'Strong' | 'Warm' | 'Same' | TRUE |
| 'Sunny' | 'Warm' | 'High' | 'Strong' | 'Warm' | 'Same' | TRUE |
| 'Rainy' | 'Cold' | 'High' | 'Strong' | 'Warm' | 'Change' | FALSE |
| 'Sunny' | 'Warm' | 'High' | 'Strong' | 'Cool' | 'Change' | TRUE |

**Output:**

["'Sunny'", " 'Warm'", " 'Normal'", " 'Strong'", " 'Warm'", " 'Same'", 'TRUE']

["'Sunny'", " 'Warm'", " 'High'", " 'Strong'", " 'Warm'", " 'Same'", 'TRUE']

["'Rainy'", " 'Cold'", " 'High'", " 'Strong'", " 'Warm'", " 'Change'", 'FALSE']

["'Sunny'", " 'Warm'", " 'High'", " 'Strong'", " 'Cool'", " 'Change'", 'TRUE'] The maximally specific hypothesis for a given training example is: [["'Sunny'", " 'Warm'", '?', " 'Strong'", '?', '?']]

# Program No: 2 : CANDIDATE-ELIMINATION ALGORITHM

For a given set of training data examples stored in a .CSV file, implement and demonstrate the **Candidate-Elimination algorithm** to output a description of the set of all hypotheses consistent with the training examples.

|  |
| --- |
| **Objective**  To implement and demonstrate the **Candidate-Elimination algorithm** to output a description of the set of all hypotheses consistent with the training examples. |
| **Data Set**  *Tennis data set*: This data set contains the set of examples days on which playing of tennis is possible or not, based on attributes Sky, AirTemp, Humidity, Wind, Water and Forecast. |
| **ML Algorithm**  Supervised Learning – **Candidate-Elimination algorithm** |
| **Description**   * The key idea in the Candidate-Elimination algorithm is to output a description of the set of all hypotheses consistent with the training examples. * It computes the description of this set without explicitly enumerating all of its members. * This is accomplished by using the more-general-than partial ordering and maintaining a compact representation of the set of consistent hypotheses. * The algorithm represents the set of all hypotheses consistent with the observed training examples. This subset of all hypotheses is called the version space with respect to the hypothesis space H and the training examples D, because it contains all plausible versions of the target concept. * A version space can be represented with its general and specific boundary sets. * The Candidate-Elimination algorithm represents the version space by storing only its most general members G and its most specific members S. * Given only these two sets S and G, it is possible to enumerate all members of a version space by generating hypotheses that lie between these two sets in general-to-specific partial ordering over hypotheses. Every member of the version space lies between these boundaries   **Algorithm**   1. Initialize G to the set of maximally general hypotheses in H 2. Initialize S to the set of maximally specific hypotheses in H 3. For each training example d, do    1. If d is a positive example   Remove from G any hypothesis inconsistent with d , For each hypothesis s in S that is not consistent with d ,  Remove s from S  Add to S all minimal generalizations h of s such that h is consistent with d, and some member of G is more general than h  Remove from S, hypothesis that is more general than another hypothesis in S   * 1. If d is a negative example   Remove from S any hypothesis inconsistent with d For each hypothesis g in G that is not consistent with d  Remove g from G  Add to G all minimal specializations h of g such that h is consistent with d, and some member of S is more specific than h  Remove from G any hypothesis that is less general than another hypothesis in G |

# Program:

import csv

def get\_domains(examples):

d = [set() for i in examples[0]] for x in examples:

for i, xi in enumerate(x): d[i].add(xi)

return [list(sorted(x)) for x in d]

def more\_general(h1, h2):

more\_general\_parts = [] for x, y in zip(h1, h2):

mg = x == "?" or (x != "0" and (x == y or y == "0")) more\_general\_parts.append(mg)

return all(more\_general\_parts)

def fulfills(example, hypothesis):

# the implementation is the same as for hypotheses: return more\_general(hypothesis, example)

def min\_generalizations(h, x): h\_new = list(h)

for i in range(len(h)):

if not fulfills(x[i:i+1], h[i:i+1]):

h\_new[i] = '?' if h[i] != '0' else x[i] return [tuple(h\_new)]

def min\_specializations(h, domains, x): results = []

for i in range(len(h)): if h[i] == "?":

for val in domains[i]:

if x[i] != val:

h\_new = h[:i] + (val,) + h[i+1:] results.append(h\_new)

elif h[i] != "0":

h\_new = h[:i] + ('0',) + h[i+1:] results.append(h\_new)

return results

def generalize\_S(x, G, S):

S\_prev = list(S) for s in S\_prev:

if s not in S:

continue

if not fulfills(x, s):

S.remove(s)

Splus = min\_generalizations(s, x)

## keep only generalizations that have a counterpart in G S.update([h for h in Splus if any([more\_general(g,h) for g in G])]) ## remove hypotheses less specific than any other in S

S.difference\_update([h for h in S if any([more\_general(h, h1) for h1 in S if h

!= h1])])

return S

def specialize\_G(x, domains, G, S): G\_prev = list(G)

for g in G\_prev:

if g not in G:

continue if fulfills(x, g):

G.remove(g)

Gminus = min\_specializations(g, domains, x)

## keep only specializations that have a conuterpart in S G.update([h for h in Gminus if any([more\_general(h, s) for s in S])]) ## remove hypotheses less general than any other in G

G.difference\_update([h for h in G if any([more\_general(g1, h) for g1 in G if h != g1])])

return G

def candidate\_elimination(examples):

domains = get\_domains(examples)[:-1] n = len(domains)

G = set([("?",)\*n])

S = set([("0",)\*n])

print("Maximally specific hypotheses - S ") print("Maximally general hypotheses - G ")

i=0 print("\nS[0]:",str(S),"\nG[0]:",str(G))

for xcx in examples: i=i+1

x, cx = xcx[:-1], xcx[-1] # Splitting data into attributes and decisions if cx=='Y': # x is positive example

G = {g for g in G if fulfills(x, g)} S = generalize\_S(x, G, S)

else: # x is negative example

S = {s for s in S if not fulfills(x, s)} G = specialize\_G(x, domains, G, S)

print("\nS[{0}]:".format(i),S)

print("G[{0}]:".format(i),G)

return

with open('P2\_dataset1.txt') as csvFile:

examples = [tuple(line) for line in csv.reader(csvFile)] candidate\_elimination(examples)

**For data set use P2\_dataset1.txt File** sunny,warm,normal,strong,warm,same,Y sunny,warm,high,strong,warm,same,Y rainy,cold,high,strong,warm,change,N sunny,warm,high,strong,cool,change,Y **Output:**

Maximally specific hypotheses - S Maximally general hypotheses - G

S[0]: {('0', '0', '0', '0', '0', '0')}

G[0]: {('?', '?', '?', '?', '?', '?')}

S[1]: {('sunny', 'warm', 'normal', 'strong', 'warm', 'same')}

G[1]: {('?', '?', '?', '?', '?', '?')}

S[2]: {('sunny', 'warm', '?', 'strong', 'warm', 'same')}

G[2]: {('?', '?', '?', '?', '?', '?')}

S[3]: {('sunny', 'warm', '?', 'strong', 'warm', 'same')}

G[3]: {('?', 'warm', '?', '?', '?', '?'), ('sunny', '?', '?', '?', '?', '?'), ('?', '?', '?', '?', '?', 'same')}

S[4]: {('sunny', 'warm', '?', 'strong', '?', '?')}

G[4]: {('?', 'warm', '?', '?', '?', '?'), ('sunny', '?', '?', '?', '?', '?')}

# Program No 3:Linear Regression

# Simple Linear Regression With scikit-learn Let’s start with the simplest case, which is simple linear regression.

# There are five basic steps when you’re implementing linear regression:

# 1. Import the packages and classes youneed.

# 2. Provide data to work with and eventually do appropriatetransformations.

# 3. Create a regression model and fit it with existingdata.

# 4. Check the results of model fitting to know whether the model issatisfactory.

# 5. Apply the model forpredictions.

# SOURCE CODE:

# importnumpy as np

# importmatplotlib.pyplot as plt

# defestimate\_coef(x, y):

# # number of observations/points n =

# np.size(x)

# # mean of x and y vector

# m\_x, m\_y = np.mean(x), np.mean(y)

# # calculating cross-deviation and deviation aboutx SS\_xy =

# np.sum(y\*x) - n\*m\_y\*m\_x

# SS\_xx = np.sum(x\*x) -n\*m\_x\*m\_x

# # calculating regression coefficients b\_1 =

# SS\_xy / SS\_xx

# b\_0 = m\_y - b\_1\*m\_x

# return(b\_0, b\_1)

# defplot\_regression\_line(x, y, b):

# # plotting the actual points as scatter plot

# plt.scatter(x, y, color = "m", marker = "o", s = 30)

# # predicted response vector y\_pred =

# b[0] + b[1]\*x

# # plotting the regression line

# plt.plot(x, y\_pred, color = "g")

# # putting labels plt.xlabel('x')

# plt.ylabel('y')

# # function to show plot

# plt.show()

# def main():

# # observations

# 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])

# # estimating coefficients b =

# estimate\_coef(x, y)

# print("Estimated coefficients:\nb\_0 = {} \ \nb\_1 = {}".format(b[0], b[1]))

# # plotting regression line

# plot\_regression\_line(x, y, b)

# ifname == "main":

# main()

# OUTPUT:

# Estimated coefficients:

# b\_0 = -0.05862068965

# 

# Program No: 4: ID3 ALGORITHM

Write a program to demonstrate the working of the decision tree based **ID3 algorithm**. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.

|  |
| --- |
| **Objective**  To demonstrate the working of the decision tree based **ID3 algorithm** using an appropriate data set for building the decision tree and apply this knowledge to classify a new sample. |
| **Data Set**  *Tennis data set*: This data set contains the set of examples days on which playing of tennis is possible or not, based on attributes Sky, AirTemp, Humidity, Wind, Water and Forecast. |
| **ML Algorithm**  Supervised Learning – **ID3 algorithm** |
| **Description**   * ID3 algorithm is a basic algorithm that learns decision trees by constructing them topdown, beginning with the question "which attribute should be tested at the root of the tree?". * To answer this question, each instance attribute is evaluated using a statistical test to determine how well it alone classifies the training examples. The best attribute is selected and used as the test at the root node of the tree. * A descendant of the root node is then created for each possible value of this attribute, and the training examples are sorted to the appropriate descendant node (i.e., down the branch corresponding to the example's value for this attribute). * The entire process is then repeated using the training examples associated with each descendant node to select the best attribute to test at that point in the tree. * A simplified version of the algorithm, specialized to learning boolean-valued functions (i.e., concept learning), is described below.   **Algorithm:** ID3(Examples, TargetAttribute, Attributes) Input: Examples are the training examples.  Targetattribute is the attribute whose value is to be predicted by the tree.  Attributes is a list of other attributes that may be tested by the learned decision tree.  Output: Returns a decision tree that correctly classiJies the given Examples Method:   1. Create a Root node for the tree 2. If all Examples are positive, Return the single-node tree Root, with label = + 3. If all Examples are negative, Return the single-node tree Root, with label = - 4. If Attributes is empty,   Return the single-node tree Root, with label = most common value of TargetAttribute in Examples  Else  A ← the attribute from Attributes that best classifies Examples The decision attribute for Root ←A  For each possible value, vi, of A,  Add a new tree branch below Root, corresponding to the test A = |

|  |
| --- |
| vi Let Examplesvi be the subset of Examples that have value vi for A  If Examplesvi is empty Then below this new branch add a leaf node with label = most common value of TargetAttribute in Examples  Else  below this new branch add the subtree ID3(Examplesvi, TargetAttribute, Attributes–  {A})  End  5. Return Root |

# Program:

import math import csv

def load\_csv(filename):

lines = csv.reader(open(filename, "r")); dataset = list(lines)

headers = dataset.pop(0) return dataset, headers

class Node:

def init (self, attribute): self.attribute = attribute self.children = [] self.answer = ""

# NULL indicates children exists.

# Not Null indicates this is a Leaf Node def subtables(data, col, delete):

dic = {}

coldata = [ row[col] for row in data]

attr = list(set(coldata)) # All values of attribute retrived for k in attr:

dic[k] = []

for y in range(len(data)): key = data[y][col]

if delete:

del data[y][col] dic[key].append(data[y])

return attr, dic def entropy(S):

attr = list(set(S))

if len(attr) == 1: #if all are +ve/-ve then entropy = 0 return 0

counts = [0,0] # Only two values possible 'yes' or 'no'

for i in range(2):

counts[i] = sum( [1 for x in S if attr[i] == x] ) / (len(S) \* 1.0) sums = 0

for cnt in counts:

sums += -1 \* cnt \* math.log(cnt, 2) return sums

def compute\_gain(data, col):

attValues, dic = subtables(data, col, delete=False) total\_entropy = entropy([row[-1] for row in data]) for x in range(len(attValues)):

ratio = len(dic[attValues[x]]) / ( len(data) \* 1.0)

entro = entropy([row[-1] for row in dic[attValues[x]]]) total\_entropy -= ratio\*entro

return total\_entropy

def build\_tree(data, features): lastcol = [row[-1] for row in data]

if (len(set(lastcol))) == 1: # If all samples have same labels return that label node=Node("")

node.answer = lastcol[0] return node

n = len(data[0])-1

gains = [compute\_gain(data, col) for col in range(n) ]

split = gains.index(max(gains)) # Find max gains and returns index node = Node(features[split]) # 'node' stores attribute selected

#del (features[split])

fea = features[:split]+features[split+1:]

attr, dic = subtables(data, split, delete=True) # Data will be spilt in subtables for x in range(len(attr)):

child = build\_tree(dic[attr[x]], fea) node.children.append((attr[x], child))

return node

def print\_tree(node, level): if node.answer != "":

print(" "\*level, node.answer) # Displays leaf node yes/no return

print(" "\*level, node.attribute) # Displays attribute Name for value, n in node.children:

print(" "\*(level+1), value) print\_tree(n, level + 2)

def classify(node,x\_test,features): if node.answer != "":

print(node.answer) return

pos = features.index(node.attribute) for value, n in node.children:

if x\_test[pos]==value: classify(n,x\_test,features)

''' Main program '''

dataset, features = load\_csv("P3\_data3.csv") # Read Tennis data node = build\_tree(dataset, features) # Build decision tree print("The decision tree for the dataset using ID3 algorithm is ") print\_tree(node, 0)

testdata, features = load\_csv("P3\_data3\_test.csv") for xtest in testdata:

print("The test instance : ",xtest) print("The predicted label : ", end="") classify(node,xtest,features)

# Training instances: P3\_data3.csv Outlook,Temperature,Humidity,Wind,Target

sunny,hot,high,weak,no

sunny,hot,high,strong,no overcast,hot,high,weak,yes rain,mild,high,weak,yes rain,cool,normal,weak,yes rain,cool,normal,strong,no overcast,cool,normal,strong,yes sunny,mild,high,weak,no sunny,cool,normal,weak,yes rain,mild,normal,weak,yes sunny,mild,normal,strong,yes overcast,mild,high,strong,yes overcast,hot,normal,weak,yes rain,mild,high,strong,no

**Testing instances: P3\_data3\_test.csv** Outlook,Temperature,Humidity,Wind rain,cool,normal,strong sunny,mild,normal,strong

# Output:

The decision tree for the dataset using ID3 algorithm is Outlook

rain Wind strong no weak yes sunny

Humidity high

no normal yes overcast yes

The test instance : ['rain', 'cool', 'normal', 'strong'] The predicted label : no

The test instance : ['sunny', 'mild', 'normal', 'strong'] The predicted label : yes

# Program No 5: BACKPROPAGATION ALGORITHM

Build an Artificial Neural Network by implementing the **Backpropagation algorithm** and test the same using appropriate data sets.

|  |
| --- |
| **Objective**  To build an Artificial Neural Network by implementing the **Backpropagation algorithm** and test the same using appropriate data sets. |
| **Data Set**  Data stored as a list having two features – number of hours slept, number of hours studied with the test score being the class lable. |
| **ML Algorithm**  Supervised Learning – **Backpropagation algorithm** |
| **Description**   * Artificial neural networks (ANNs) provide a general, practical method for learning real- valued, discrete-valued, and vector-valued functions from examples. * Algorithms such as BACKPROPAGATION gradient descent to tune network parameters to best fit a training set of input-output pairs. * ANN learning is robust to errors in the training data and has been successfully applied to problems such as interpreting visual scenes, speech recognition, and learning robot control strategies.   Backpropagation algorithm   1. Create a feed-forward network with ni nputs, nhidden hidden units, and nout output units. 2. Initialize each wi to some small random value (e.g., between -.05 and .05). 3. Until the termination condition is met, do   For each training example <(x1,…xn),t>, do  // Propagate the input forward through the network:   * 1. Input the instance (x1, ..,xn) to the n/w & compute the n/w outputs ok for every unit   // Propagate the errors backward through the network:   * 1. For each output unit k, calculate its error term k ; k = ok(1-ok)(tk-ok)   2. For each hidden unit h, calculate its error term h; h=oh(1-oh) k wh,k k   3. For each network weight wi,j do; wi,j=wi,j+ wi,j where wi,j= j xi,j |

# Program:



import numpy as np

X = np.array(([2, 9], [1, 5], [3, 6]), dtype=float)

y = np.array(([92], [86], [89]), dtype=float)

X = X/np.amax(X,axis=0) # maximum of X array longitudinally y = y/100

#Sigmoid Function def sigmoid (x):

return 1/(1 + np.exp(-x))

#Derivative of Sigmoid Function def derivatives\_sigmoid(x):

return x \* (1 - x) #Variable initialization

epoch=5000 #Setting training iterations lr=0.1 #Setting learning rate

inputlayer\_neurons = 2 #number of features in data set hiddenlayer\_neurons = 3 #number of hidden layers neurons output\_neurons = 1 #number of neurons at output layer #weight and bias initialization

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))

#draws a random range of numbers uniformly of dim x\*y for i in range(epoch):

#Forward Propogation hinp1=np.dot(X,wh) hinp=hinp1 + bh hlayer\_act = sigmoid(hinp)

outinp1=np.dot(hlayer\_act,wout) outinp= outinp1+ bout

output = sigmoid(outinp) #Backpropagation

EO = y-output

outgrad = derivatives\_sigmoid(output) d\_output = EO\* outgrad

EH = d\_output.dot(wout.T)

hiddengrad = derivatives\_sigmoid(hlayer\_act)#how much hidden layer wts contributed to error d\_hiddenlayer = EH \* hiddengrad

wout += hlayer\_act.T.dot(d\_output) \*lr# dotproduct of nextlayererror and currentlayerop # 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("Input: \n" + str(X))

print("Actual Output: \n" + str(y)) print("Predicted Output: \n" ,output)

# Output:

Input:

[[0.66666667 1. ]

[0.33333333 0.55555556]

[1. 0.66666667]]

Actual Output: [[0.92]

[0.86]

[0.89]]

Predicted Output: [[0.89517727]

[0.88348539]

[0.89135475]]

# Program No 6: NAÏVE BAYESIAN CLASSIFIER

Write a program to implement the **naïve Bayesian classifier** for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets.

|  |
| --- |
| **Objective**  To implement the **naïve Bayesian classifier** for a sample training data set and to compute the accuracy of the classifier for the test data |
| **Data Set**  Plasma Indian Diabetes data set stored as .CSV. The attributes are Number of times pregnant, Plasma Glucose concentration, Blood Pressure, Triceps skin fold thickness, serum insulin, Body mass Index, Diabetes pedigree function , Age |
| **ML Algorithm**  Supervised Learning – **Naïve Bayes Algorithm** |
| **Description**  **Naive Bayes algorithm :** Naive Bayes algorithm is a classification technique based on Bayes’ Theorem with an assumption of independence among predictors. In simple terms, a Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature. For example, a fruit may be considered to be an apple if it is red, round, and about 3 inches in diameter. Even if these features depend on each other or upon the existence of the other features, all of these properties independently contribute to the probability that this fruit is an apple and that is why it is known as ‘Naive’.  Naive Bayes model is easy to build and particularly useful for very large data sets. Along with simplicity, Naive Bayes is known to outperform even highly sophisticated classification methods.  Bayes theorem provides a way of calculating posterior probability P(c|x) from P(c), P(x) and P(x|c). Look at the equation below:    where  P(c|x) is the posterior probability of class (c, target) given predictor (x, attributes). P(c) is the prior probability of class.  P(x|c) is the likelihood which is the probability of predictor given class. P(x) is the prior probability of predictor.  The naive Bayes classifier applies to learning tasks where each instance x is described by a conjunction of attribute values and where the target function f (x) can take on any value from  some finite set V. A set of training examples of the target function is provided, and a new |

|  |
| --- |
| instance is presented, described by the tuple of attribute values (a1, a2, ... ,an). The learner is asked to predict the target value, or classification, for this new instance.  The Bayesian approach to classifying the new instance is to assign the most probable target value,  vMAP, given the attribute values (al, a2, ..., an) that describe the instance.    We can use Bayes theorem to rewrite this expression as    Now we could attempt to estimate the two terms in Equation (19) based on the training data. It is easy to estimate each of the P(vj) simply by counting the frequency with which each target value vj occurs in the training data.  The naive Bayes classifier is based on the simplifying assumption that the attribute values are conditionally independent given the target value. In other words, the assumption is that given the target value of the instance, the probability of observing the conjunction al, a2, … , an, is just the product of the probabilities for the individual attributes: P(al, a2, … , an | vj) = Πi P(ai | vj). Substituting this, we have the approach used by the naive Bayes classifier.    where vNB denotes the target value output by the naive Bayes classifier.  When dealing with continuous data, a typical assumption is that the continuous values associated with each class are distributed according to a Gaussian distribution. For example, suppose the training data contains a continuous attribute, x. We first segment the data by the class, and then compute the mean and variance of x in each class.  Let *μ* be the mean of the values in *x* associated with class *Ck*, and let *σ2k* be the variance of the values in x associated with class Ck. Suppose we have collected some observation value v. Then, the probability distribution of v given a class Ck, p(x=v|Ck) can be computed by plugging v into the equation for a Normal distribution parameterized by *μ* and *σ2k . Thatis*    Above method is adopted in our implementation of the program. |

|  |
| --- |
| Pima Indian diabetics dataset  This dataset is originally from the National Institute of Diabetes and Digestive and Kidney  Diseases. The objective of the dataset is to diagnostically predict whether or not a patient has diabetes, based on certain diagnostic measurements included in the dataset. |

**Program** import csv import random import math

def loadCsv(filename):

lines = csv.reader(open(filename, "r")); dataset = list(lines)

for i in range(len(dataset)):

#converting strings into numbers for processing dataset[i] = [float(x) for x in dataset[i]]

return dataset

def splitDataset(dataset, splitRatio): #67% training size

trainSize = int(len(dataset) \* splitRatio); trainSet = []

copy = list(dataset);

while len(trainSet) < trainSize:

#generate indices for the dataset list randomly to pick ele for training data index = random.randrange(len(copy)); trainSet.append(copy.pop(index))

return [trainSet, copy]

def separateByClass(dataset): separated = {}

#creates a dictionary of classes 1 and 0 where the values are the instacnes belonging to each class for i in range(len(dataset)):

vector = dataset[i]

if (vector[-1] not in separated): separated[vector[-1]] = []

separated[vector[-1]].append(vector) return separated

def mean(numbers):

return sum(numbers)/float(len(numbers))

def stdev(numbers):

avg = mean(numbers)

variance = sum([pow(x-avg,2) for x in numbers])/float(len(numbers)-1) return math.sqrt(variance)

def summarize(dataset):

summaries = [(mean(attribute), stdev(attribute)) for attribute in zip(\*dataset)]; del summaries[-1]

return summaries

def summarizeByClass(dataset):

separated = separateByClass(dataset) #print(separated)

summaries = {}

for classValue, instances in separated.items(): #summaries is a dic of tuples(mean,std) for each class value

summaries[classValue] = summarize(instances) return summaries

def calculateProbability(x, mean, stdev):

exponent = math.exp(-(math.pow(x-mean,2)/(2\*math.pow(stdev,2)))) return (1 / (math.sqrt(2\*math.pi) \* stdev)) \* exponent

def calculateClassProbabilities(summaries, inputVector): probabilities = {}

for classValue, classSummaries in summaries.items():#class and attribute information as mean and sd

probabilities[classValue] = 1

for i in range(len(classSummaries)):

mean, stdev = classSummaries[i] #take mean and sd of every attribute for class 0 and 1 seperaely

x = inputVector[i] #testvector's first attribute probabilities[classValue] \*= calculateProbability(x, mean, stdev);#use

normal dist

return probabilities

def predict(summaries, inputVector):

probabilities = calculateClassProbabilities(summaries, inputVector) bestLabel, bestProb = None, -1

prob

for classValue, probability in probabilities.items():#assigns that class which has he highest

if bestLabel is None or probability > bestProb: bestProb = probability

bestLabel = classValue return bestLabel

def getPredictions(summaries, testSet): predictions = []

for i in range(len(testSet)):

result = predict(summaries, testSet[i]) predictions.append(result)

return predictions

def getAccuracy(testSet, predictions): correct = 0

for i in range(len(testSet)):

if testSet[i][-1] == predictions[i]: correct += 1

return (correct/float(len(testSet))) \* 100.0

def main():

filename = 'P5\_naivedata.csv' splitRatio = 0.67

dataset = loadCsv(filename);

print('Pima Indian Diabetes Dataset loaded...') print('Total instances available :',len(dataset)) print('Total attributes present :',len(dataset[0])-1) print("First Five instances of dataset:")

for i in range(5):

print(i+1 , ':' , dataset[i])

trainingSet, testSet = splitDataset(dataset, splitRatio) print('\nDataset is split into training and testing set.')

print('Training examples = {0} \nTesting examples = {1}'.format(len(trainingSet),len(testSet)))

# prepare model

summaries = summarizeByClass(trainingSet); #print(summaries)

# test model

predictions = getPredictions(summaries, testSet)

#print(predictions)

accuracy = getAccuracy(testSet, predictions) print('Accuracy of the classifier is : {0}%'.format(accuracy))

main()

# Output:

Pima Indian Diabetes Dataset loaded... Total instances available : 768

Total attributes present : 8 First Five instances of dataset:

1 : [6.0, 148.0, 72.0, 35.0, 0.0, 33.6, 0.627, 50.0, 1.0]

2 : [1.0, 85.0, 66.0, 29.0, 0.0, 26.6, 0.351, 31.0, 0.0]

3 : [8.0, 183.0, 64.0, 0.0, 0.0, 23.3, 0.672, 32.0, 1.0]

4 : [1.0, 89.0, 66.0, 23.0, 94.0, 28.1, 0.167, 21.0, 0.0]

5 : [0.0, 137.0, 40.0, 35.0, 168.0, 43.1, 2.288, 33.0, 1.0]

Dataset is split into training and testing set. Training examples = 514

Testing examples = 254

Accuracy of the classifier is : 72.83464566929135%

# Program No 7: DOCUMENT CLASSIFICATION USING NAÏVE BAYESIANCLASSIFIER

Assuming a set of documents that need to be classified, use the **naïve Bayesian Classifier** model to perform this task. Built-in Java classes/API can be used to write the program. Calculate the accuracy, precision, and recall for your data set.

|  |
| --- |
| **Objective**  To implement a Bayesian Classifier Model that classifies a set of documents and calculates the accuracy, precision and recall for the data set. |
| **Data Set**  Contains text as sentences labelled positive and negative. The data set contains a total of 10 instances |
| **ML Algorithm**  Supervised Learning – **Naïve Bayes Algorithm** |
| **Description**  For the theory of the naive Bayesian classifier refer Program No. 5. Theory of performance analysis analysis is elaborated here.  Analysis of Document Classification     * For classification tasks, the terms true positives, true negatives, false positives, and false negatives compare the results of the classifier under test with trusted external judgments. The terms positive and negative refer to the classifier's prediction (sometimes known as the expectation), and the terms true and false refer to whether that prediction corresponds to the external judgment (sometimes known as the observation). * Precision - Precision is the ratio of correctly predicted positive documents to the total predicted positive documents. High precision relates to the low false positive rate.   Precision = (Σ True positive ) / ( Σ True positive + Σ False positive)   * Recall (Sensitivity) - Recall is the ratio of correctly predicted positive documents to the all observations in actual class.   Recall = (Σ True positive ) / ( Σ True positive + Σ False negative)   * Accuracy - Accuracy is the most intuitive performance measure and it is simply a ratio of correctly predicted observation to the total observations. One may think that, if we have high accuracy then our model is best. Yes, accuracy is a great measure but only when you have symmetric datasets where values of false positive and false negatives are almost same. Therefore, you have to look at other parameters to evaluate the performance of your model. For our model, we have got 0.803 which means our model is approx. 80% accurate.   Accuracy = (Σ True positive + Σ True negative) / Σ Total population |

# Program:

import pandas as pd msg=pd.read\_csv('P6\_naivetext1.csv',names=['message','label']) print('Total instances in the dataset:',msg.shape[0]) msg['labelnum']=msg.label.map({'pos':1,'neg':0}) X=msg.message

Y=msg.labelnum

print('\nThe message and its label of first 5 instances are listed below') X5, Y5 = X[0:5], msg.label[0:5]

for x, y in zip(X5,Y5): print(x,',',y)

#splitting the dataset into train and test data

from sklearn.model\_selection import train\_test\_split xtrain,xtest,ytrain,ytest=train\_test\_split(X,Y) print('\nDataset is split into Training and Testing samples') print('Total training instances :', xtrain.shape[0]) print('Total testing instances :', xtest.shape[0])

#output of count vectoriser is a sparse matrix

# CountVectorizer - stands for 'feature extraction'

from sklearn.feature\_extraction.text import CountVectorizer count\_vect = CountVectorizer()

xtrain\_dtm = count\_vect.fit\_transform(xtrain) #Sparse matrix xtest\_dtm=count\_vect.transform(xtest) print(count\_vect.get\_feature\_names())

print('\nTotal features extracted using CountVectorizer:',xtrain\_dtm.shape[1])

print('\nFeatures for first 5 training instances are listed below') df=pd.DataFrame(xtrain\_dtm.toarray(),columns=count\_vect.get\_feature\_names()) print(df[0:5])#tabular representation

#print(xtrain\_dtm) #Same as above but sparse matrix representation

# Training Naive Bayes (NB) classifier on training data. from sklearn.naive\_bayes import MultinomialNB

clf = MultinomialNB().fit(xtrain\_dtm,ytrain) predicted = clf.predict(xtest\_dtm)

print('\nClassstification results of testing samples are given below') for doc, p in zip(xtest, predicted):

pred = 'pos' if p==1 else 'neg' print('%s -> %s ' % (doc, pred))

#printing accuracy metrics from sklearn import metrics print('\nAccuracy metrics')

print('\nAccuracy of the classifer is',metrics.accuracy\_score(ytest,predicted)) print('\nConfusion matrix')

print(metrics.confusion\_matrix(ytest,predicted)) print('\nRecall') print(metrics.recall\_score(ytest,predicted)) print('\nPrecison ') print(metrics.precision\_score(ytest,predicted))

# Output:

Total instances in the dataset: 18

The message and its label of first 5 instances are listed below I love this sandwich , pos

This is an amazing place , pos

I feel very good about these beers , pos This is my best work , pos

What an awesome view , pos

Dataset is split into Training and Testing samples Total training instances : 13

Total testing instances : 5

['about', 'am', 'an', 'and', 'awesome', 'bad', 'beers', 'best', 'boss', 'can', 'dance', 'deal', 'do', 'enemy',

'feel', 'good', 'he', 'horrible', 'is', 'juice', 'like', 'locality', 'love', 'my', 'not', 'of', 'place', 'restaurant',

'sandwich', 'sick', 'stay', 'sworn', 'taste', 'that', 'the', 'these', 'this', 'tired', 'to', 'very', 'view', 'what', 'with', 'work']

Total features extracted using CountVectorizer: 44 Features for first 5 training instances are listed below

about am an and awesome bad ... to very view what with work

0 0 0 1 0 1 0 ... 0 0 0 0 0 0

1 0 0 1 0 1 0 ... 0 0 1 1 0 0

2 0 0 0 0 0 0 ... 0 0 0 0 1 0

3 0 0 0 0 0 0 ... 0 0 0 0 0 0

4 1 0 0 0 0 0 ... 0 1 0 0 0 0

[5 rows x 44 columns]

Classstification results of testing samples are given below We will have good fun tomorrow -> pos

I am tired of this stuff -> neg This is an amazing place -> pos What a great holiday -> pos

I went to my enemy's house today -> neg Accuracy metrics

Accuracy of the classifer is 1.0

Confusion matrix [[2 0]

[0 3]]

Recall 1.0

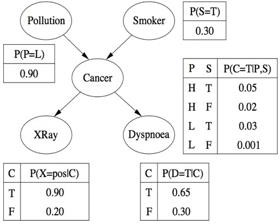
Precison 1.0

# Program No: 8: BAYESIAN NETWORK

Write a program to construct a **Bayesian network** considering medical data. Use this model to demonstrate the diagnosis of heart patients using standard Heart Disease Data Set. You can use Java/Python ML library classes/API.

|  |
| --- |
| **Objective**  To write a program to construct a **Bayesian network** considering medical data and to demonstrate the diagnosis of heart patients using standard Heart Disease Data Set. |
| **Data Set**  Diagnosis dataset of heart patients |
| **ML Algorithm**  Supervised Learning – **Bayesian Learning** |
| **Description**   * Bayesian networks are very convenient for representing similar probabilistic relationships between multiple events. * Bayesian networks as graphs - People usually represent Bayesian networks as directed graphs in which each node is a hypothesis or a random process. In other words, something that takes at least 2 possible values you can assign probabilities to. For example, there can be a node that represents the state of the dog (barking or not barking at the window), the weather (raining or not raining), etc. * The arrows between nodes represent the conditional probabilities between them — how information about the state of one node changes the probability distribution of another node it’s connected to. |

# Example:



**Program:**

import numpy as np import pandas as pd import csv

from pgmpy.estimators import MaximumLikelihoodEstimator from pgmpy.models import BayesianModel

from pgmpy.inference import VariableElimination #Read the attributes

lines = list(csv.reader(open('P7\_data7\_names.csv', 'r'))); attributes = lines[0]

#attributes = ['age', 'sex', 'cp', 'trestbps', 'chol', 'fbs', 'restecg', 'thalach', 'exang', # 'oldpeak', 'slope', 'ca', 'thal', 'heartdisease']

#Read Cleveland Heart dicease data

heartDisease = pd.read\_csv('P7\_data7\_heart.csv', names = attributes) heartDisease = heartDisease.replace('?', np.nan)

# Display the data

print('Few examples from the dataset are given below') print(heartDisease.head())

print('\nAttributes and datatypes') print(heartDisease.dtypes)

# Model Baysian Network

model = BayesianModel([('age', 'trestbps'), ('age', 'fbs'), ('sex', 'trestbps'), ('sex', 'trestbps'),

('exang', 'trestbps'),('trestbps','heartdisease'),('fbs','heartdisease'),

('heartdisease','restecg'),('heartdisease','thalach'),('heartdisease','chol')]) # Learning CPDs using Maximum Likelihood Estimators print('\nLearning CPDs using Maximum Likelihood Estimators...'); model.fit(heartDisease, estimator=MaximumLikelihoodEstimator)

# Inferencing with Bayesian Network print('\nInferencing with Bayesian Network:') HeartDisease\_infer = VariableElimination(model) # Computing the probability of bronc given smoke.

print('\n1.Probability of HeartDisease given Age=20')

q = HeartDisease\_infer.query(variables=['heartdisease'], evidence={'age': 28}) print(q['heartdisease'])

print('\n2. Probability of HeartDisease given chol (Cholestoral) =100')

q = HeartDisease\_infer.query(variables=['heartdisease'], evidence={'chol': 100}) print(q['heartdisease'])

# Output:

**Dataset:**

**P7\_data7\_names.csv (14 attributes)** age,sex,cp,trestbps,chol,fbs,restecg,thalach,exang,oldpeak, slope,ca,thal,heartdisease **P7\_data7\_heart.csv (5 instances out of 303)**

63.0,1.0,1.0,145.0,233.0,1.0,2.0,150.0,0.0,2.3,3.0,0.0,6.0,0

67.0,1.0,4.0,160.0,286.0,0.0,2.0,108.0,1.0,1.5,2.0,3.0,3.0,2

67.0,1.0,4.0,120.0,229.0,0.0,2.0,129.0,1.0,2.6,2.0,2.0,7.0,1

37.0,1.0,3.0,130.0,250.0,0.0,0.0,187.0,0.0,3.5,3.0,0.0,3.0,0

41.0,0.0,2.0,130.0,204.0,0.0,2.0,172.0,0.0,1.4,1.0,0.0,3.0,0

# Output:

Learing CPDs using Maximum Likelihood Estimators... Inferencing with Bayesian Network:

1. Probability of HeartDisease given Age=28

╒════════════════╤═════════════════════╕

│ heartdisease │ phi(heartdisease) │

╞════════════════╪═════════════════════╡

│ heartdisease\_0 │ 0.6791 │

├────────────────┼─────────────────────┤

│ heartdisease\_1 │ 0.1212 │

├────────────────┼─────────────────────┤

│ heartdisease\_2 │ 0.0810 │

├────────────────┼─────────────────────┤

│ heartdisease\_3 │ 0.0939 │

├────────────────┼─────────────────────┤

│ heartdisease\_4 │ 0.0247 │

╘════════════════╧═════════════════════╛

1. Probability of HeartDisease given chol (Cholestoral) =100

╒════════════════╤═════════════════════╕

│ heartdisease │ phi(heartdisease) │

╞════════════════╪═════════════════════╡

│ heartdisease\_0 │ 0.5400 │

├────────────────┼─────────────────────┤

│ heartdisease\_1 │ 0.1533 │

├────────────────┼─────────────────────┤

│ heartdisease\_2 │ 0.1303 │

├────────────────┼─────────────────────┤

│ heartdisease\_3 │ 0.1259 │

├────────────────┼─────────────────────┤

│ heartdisease\_4 │ 0.0506 │

╘════════════════╧═════════════════════╛

# Program No 9: CLUSTERING BASED ON EM ALGORITHM AND K-MEANS

Apply EM algorithm to cluster a set of data stored in a .CSV file. Use the same dataset for clustering using k-Means algorithm. Compare the results of these two algorithms and comment on the quality of clustering. You can add Java/Python ML library classes/API in the program.

|  |
| --- |
| **Objective**  **To a**pply **EM algorithm** to cluster a set of data stored in a .CSV file. Use the same data set for clustering using ***k*-Means algorithm** and comparing the results of these two algorithms and  comment on the quality of clustering. |
| **Data Set**  Delivery fleet driver dataset with features “Driver\_id”, “distance\_feature”, “speeding\_feature” having more than 20 instances. |
| **ML Algorithm**  EM algorithm, K means algorithm – Unsupervised clustring |
| **Description**  Expectation Maximization algorithm   * The basic approach and logic of this clustering method is as follows. * Suppose we measure a single continuous variable in a large sample of observations. Further, suppose that the sample consists of two clusters of observations with different means (and perhaps different standard deviations); within each sample, the distribution of values for the continuous variable follows the normal distribution. * The goal of EM clustering is to estimate the means and standard deviations for each cluster so as to maximize the likelihood of the observed data (distribution). * Put another way, the EM algorithm attempts to approximate the observed distributions of values based on mixtures of different distributions in different clusters. The results of EM clustering are different from those computed by k-means clustering. * The latter will assign observations to clusters to maximize the distances between clusters. The EM algorithm does not compute actual assignments of observations to clusters, but classification probabilities. * In other words, each observation belongs to each cluster with a certain probability. Of course, as a final result we can usually review an actual assignment of observations to clusters, based on the (largest) classification probability.   K means Clustering   * The algorithm will categorize the items into k groups of similarity. To calculate that similarity, we will use the euclidean distance as measurement. * The algorithm works as follows:   1. First we initialize k points, called means, randomly.   2. We categorize each item to its closest mean and we update the mean’s coordinates, which are the averages of the items categorized in that mean so far.   3. We repeat the process for a given number of iterations and at the end, we have our clusters. * The “points” mentioned above are called means, because they hold the mean values of the items categorized in it. To initialize these means, we have a lot of options. An intuitive method is to initialize the means at random items in the data set. Another   method is to initialize the means at random values between the boundaries of the data set |

|  |
| --- |
| (if for a feature x the items have values in [0,3], we will initialize the means with values for x at [0,3]).   * Pseudocode:   1. Initialize k means with random values   2. For a given number of iterations: Iterate through items: |
| Find the mean closest to the item Assign item to mean  Update mean |

# Program:

import matplotlib.pyplot as plt from sklearn import datasets

from sklearn.cluster import KMeans import pandas as pd

import numpy as np

# import some data to play with iris = datasets.load\_iris()

X = pd.DataFrame(iris.data) #print(X)

X.columns = ['Sepal\_Length','Sepal\_Width','Petal\_Length','Petal\_Width'] y = pd.DataFrame(iris.target)

y.columns = ['Targets']

# Build the K Means Model model = KMeans(n\_clusters=3)

model.fit(X) # model.labels\_ : Gives cluster no for which samples belongs to # # Visualise the clustering results

plt.figure(figsize=(14,14))

colormap = np.array(['red', 'lime', 'black'])

# Plot the Original Classifications using Petal features plt.subplot(2, 2, 1)

plt.scatter(X.Petal\_Length, X.Petal\_Width, c=colormap[y.Targets], s=40) plt.title('Real Clusters')

plt.xlabel('Petal Length') plt.ylabel('Petal Width')

# Plot the Models Classifications plt.subplot(2, 2, 2)

plt.scatter(X.Petal\_Length, X.Petal\_Width, c=colormap[model.labels\_], s=40) plt.title('K-Means Clustering')

plt.xlabel('Petal Length') plt.ylabel('Petal Width') # General EM for GMM

from sklearn import preprocessing

# transform your data such that its distribution will have a # mean value 0 and standard deviation of 1.

scaler = preprocessing.StandardScaler() scaler.fit(X)

xsa = scaler.transform(X)

xs = pd.DataFrame(xsa, columns = X.columns) from sklearn.mixture import GaussianMixture gmm = GaussianMixture(n\_components=3) gmm.fit(xs)

gmm\_y = gmm.predict(xs) plt.subplot(2, 2, 3)

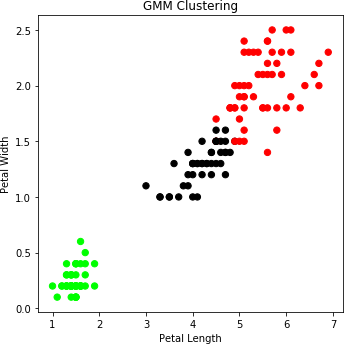
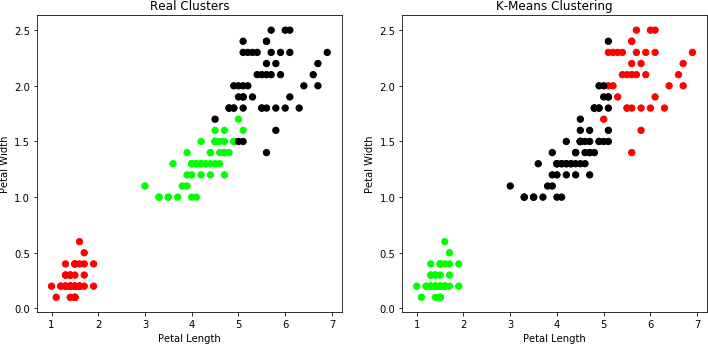
plt.scatter(X.Petal\_Length, X.Petal\_Width, c=colormap[gmm\_y], s=40) plt.title('GMM Clustering')

plt.xlabel('Petal Length') plt.ylabel('Petal Width')

print('Observation: The GMM using EM algorithm based clustering matched the true labels more closely than the Kmeans.')

# Output:

Observation: The GMM using EM algorithm based clustering matched the true labels more closely than the Kmeans.



# Program No 10: K-NEAREST NEIGHBOUR

Write a program to implement ***k*-Nearest Neighbour algorithm** to classify the iris data set. Print both correct and wrong predictions. Java/Python ML library classes can be used for this problem.

|  |
| --- |
| **Objective**  To write a program to implement ***k*-Nearest Neighbour algorithm** to classify the iris data set. |
| **Data Set**  IRIS data set with features “petal\_length”, “petal\_width”, “sepal\_lenght” , “sepal\_width” having more than 150 instances. |
| **ML Algorithm**  Supervised Learning - ***k*-Nearest Neighbour algorithm** |
| **Description:**   * K-Nearest Neighbors is one of the most basic yet essential classification algorithms in Machine Learning. It belongs to the supervised learning domain and finds intense application in pattern recognition, data mining and intrusion detection. * It is widely disposable in real-life scenarios since it is non-parametric, meaning, it does not make any underlying assumptions about the distribution of data. * Algorithm   Input: Let m be the number of training data samples. Let p be an unknown point. Method:   * 1. Store the training samples in an array of data points arr[]. This means each element of this array represents a tuple (x, y).   2. for i=0 to m   Calculate Euclidean distance d(arr[i], p).   * 1. Make set S of K smallest distances obtained. Each of these distances correspond to an already classified data point.   2. Return the majority label among S. |

# Program:

from sklearn.model\_selection import train\_test\_split from sklearn.neighbors import KNeighborsClassifier from sklearn import datasets

iris=datasets.load\_iris() print("Iris Data set loaded...") iris\_data=iris.data iris\_labels=iris.target #print(iris\_data) #print(iris\_labels)

x\_train,x\_test,y\_train,y\_test=train\_test\_split(iris\_data,iris\_labels,test\_size=0.1) print("Dataset is split into training and testing...")

print("Size of trainng data and its label",x\_train.shape,y\_train.shape) print("Size of trainng data and its label",x\_test.shape, y\_test.shape)

# Prints Label no. and their names for i in range(len(iris.target\_names)):

print("Label", i , "-",str(iris.target\_names[i]))

classifier=KNeighborsClassifier(n\_neighbors=1) classifier.fit(x\_train,y\_train) y\_pred=classifier.predict(x\_test)

# Display the results

print("Results of Classification using K-nn with K=1 ") for r in range(0,len(x\_test)):

print(" Sample:", str(x\_test[r]), " Actual-label:", str(y\_test[r]), " Predicted-label:",str(y\_pred[r])) print("Classification Accuracy :" , classifier.score(x\_test,y\_test))

# Output:

Iris Data set loaded...

Dataset is split into training and testing...

Size of trainng data and its label (135, 4) (135,) Size of trainng data and its label (15, 4) (15,) Label 0 - setosa

Label 1 - versicolor Label 2 - virginica

Results of Classification using K-nn with K=1

Sample: [4.8 3. 1.4 0.3] Actual-label: 0 Predicted-label: 0

Sample: [5.8 2.6 4. 1.2] Actual-label: 1 Predicted-label: 1

Sample: [6.7 3.1 4.7 1.5] Actual-label: 1 Predicted-label: 1

Sample: [5.4 3.4 1.5 0.4] Actual-label: 0 Predicted-label: 0

Sample: [5.7 2.5 5. 2. ] Actual-label: 2 Predicted-label: 2

Sample: [6.7 3.3 5.7 2.5] Actual-label: 2 Predicted-label: 2

Sample: [7.7 2.8 6.7 2. ] Actual-label: 2 Predicted-label: 2

Sample: [6.3 2.9 5.6 1.8] Actual-label: 2 Predicted-label: 2

Sample: [6.9 3.1 5.4 2.1] Actual-label: 2 Predicted-label: 2

Sample: [4.7 3.2 1.3 0.2] Actual-label: 0 Predicted-label: 0

Sample: [5. 3.5 1.3 0.3] Actual-label: 0 Predicted-label: 0

Sample: [5.5 4.2 1.4 0.2] Actual-label: 0 Predicted-label: 0

Sample: [5.9 3. 4.2 1.5] Actual-label: 1 Predicted-label: 1

Sample: [6.2 3.4 5.4 2.3] Actual-label: 2 Predicted-label: 2

Sample: [6.1 2.8 4. 1.3] Actual-label: 1 Predicted-label: 1 Classification Accuracy : 1.0

# Program No 11: LOCALLY WEIGHTED REGRESSION ALGORITHM

Implement the non-parametric **Locally Weighted Regression algorithm** in order to fit data points. Select appropriate data set for your experiment and draw graphs.

|  |
| --- |
| **Objective**  To implement the non-parametric **Locally Weighted Regression algorithm** in order to fit data points. |
| **Data Set**  The dataset contains billing information based on the attributes total\_bill, tip, sex, smoker, day, time, size. |
| **ML Algorithm**  Instance based learning – **Locally Weighted Regression algorithm** |
| **Description:**   * Given a dataset X, y, we attempt to find a linear model h(x) that minimizes residual sum of squared errors. The solution is given by Normal equations. * Linear model can only fit a straight line, however, it can be empowered by polynomial features to get more powerful models. Still, we have to decide and fix the number and types of features ahead. * Alternate approach is given by locally weighted regression. * Given a dataset X, y, we attempt to find a model h(x) that minimizes residual sum of weighted squared errors. * The weights are given by a kernel function which can be chosen arbitrarily and in my case I chose a Gaussian kernel. * The solution is very similar to Normal equations, we only need to insert diagonal weight matrix W.   Algorithm  def local\_regression(x0, X, Y, tau): # add bias term x0 = np.r\_[1, x0]  X = np.c\_[np.ones(len(X)), X]  # fit model: normal equations with kernel xw = X.T \* radial\_kernel(x0, X, tau)  beta = np.linalg.pinv(xw @ X) @ xw @ Y  # predict value return x0 @ beta  def radial\_kernel(x0, X, tau):  return np.exp(np.sum((X - x0) \*\* 2, axis=1) / (-2 \* tau \* tau)) |

# Program:

import matplotlib.pyplot as plt import pandas as pd

import numpy as np

def kernel(point,xmat, k): m,n = np.shape(xmat)

weights = np.mat(np.eye((m))) # eye - identity matrix for j in range(m):

diff = point - X[j]

weights[j,j] = np.exp(diff\*diff.T/(-2.0\*k\*\*2)) return weights

def localWeight(point,xmat,ymat,k): wei = kernel(point,xmat,k)

W = (X.T\*(wei\*X)).I\*(X.T\*(wei\*ymat.T)) return W

def localWeightRegression(xmat,ymat,k): m,n = np.shape(xmat)

ypred = np.zeros(m) for i in range(m):

ypred[i] = xmat[i]\*localWeight(xmat[i],xmat,ymat,k) return ypred

def graphPlot(X,ypred):

sortindex = X[:,1].argsort(0) #argsort - index of the smallest xsort = X[sortindex][:,0]

fig = plt.figure()

ax = fig.add\_subplot(1,1,1) ax.scatter(bill,tip, color='green')

ax.plot(xsort[:,1],ypred[sortindex], color = 'red', linewidth=5) plt.xlabel('Total bill')

plt.ylabel('Tip') plt.show();

# load data points

data = pd.read\_csv('P10\_data10\_tips.csv')

bill = np.array(data.total\_bill) # We use only Bill amount and Tips data tip = np.array(data.tip)

mbill = np.mat(bill) # .mat will convert nd array is converted in 2D array mtip = np.mat(tip)

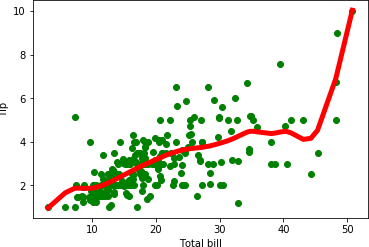
m= np.shape(mbill)[1] one = np.mat(np.ones(m))

X = np.hstack((one.T,mbill.T)) # 244 rows, 2 cols

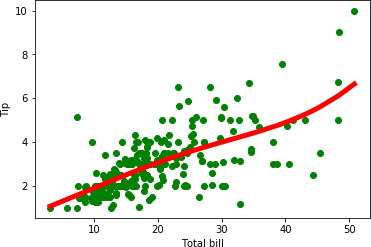
ypred = localWeightRegression(X,mtip,2) # increase k to get smooth curves graphPlot(X,ypred)

# Output:

Regression with parameter k = 2



Regression with parameter k = 10



# Viva Questions

* 1. What is the difference between supervised and unsupervised machine learning?
  2. How does Find-S algorithm and candidate elimination work in creating a hypothesis space?
  3. What is the difference between stochastic gradient descent (SGD) and gradient descent (GD)?
  4. How can you choose a classifier based on training set size?
  5. What’s the trade-off between bias and variance?
  6. How much data should you allocate for your training, validation, and test sets?
  7. Explain the IDE3 algorithm?
  8. Why we have imported numpy in program?
  9. What is the use of self in the classes?
  10. How a decision tree identifies the new data set?
  11. What do you mean by entropy?
  12. Differentiate between restriction bias and preference bias?
  13. Explain briefly classification and regression.
  14. Differentiate between supervised learning and un-supervised learning.
  15. Explain the back propagation algorithim.
  16. How we can declare arrays using numpy.
  17. What is the use of synapse?
  18. What do you mean by adaptive learning?
  19. Differentiate between feed forward and feed backward networks.
  20. What do you mean by reinforcement learning?
  21. Explain the Bayesian Networks approach.
  22. When we have to use the Bayesian Networks over the other machine learning approaches
  23. Why is Naive Bayes naive?
  24. What are the different types of Naive Bayes Classifies?
  25. What is Bayes Theorem?
  26. What are the advantages of Naive Bayes Classifies?
  27. What are the disadvantages of Naive Bayes Classifies?
  28. What is Pandas/Python Pandas?
  29. What is Python pandas used for?
  30. What is a pandas DataFrame?
  31. What is NP Python?
  32. What is Matplotlib?
  33. What are support vector machines?
  34. What is feature extraction?
  35. What does sklearn.feature\_extraction module do?
  36. What are the different types of NB Classifiers?
  37. What does CountVetcorizer module do?
  38. What do you mean by Logistic regression or Support Vector Machine?
  39. Why do Bayesian network work so well for machine learnig?
  40. What is Gaussian Processes?
  41. Differentiate between probablistic machine learning and statistical modelling.
  42. What is the use pomogranete in this program?
  43. Give some application areas of Bayesian network.
  44. How we can read the .csv file in the Jupyter notebook?
  45. Why we have used the matplotlib and what is the use of pyplot?
  46. Where we use the pandas?
  47. Explain the k-Means algorithm.
  48. Give some applications of K-mean algorithm.
  49. What is the use of scatter function?
  50. Why we have used the scikit-learn?
  51. Differentiate between Dataframe and Series.
  52. How k-Nearest Neighbor algorithm works?
  53. Why KNN is non-parametric?
  54. Can KNN be used for regression?
  55. Differentiate between KNN and K-mean?
  56. How to find best K value?
  57. How to handle categorical variables in KNN?
  58. Why we should use KNN for Regression?
  59. Explain the classification and regression in brief.
  60. Explain the Locally Weighted Regression algorithm.
  61. Why we have used the matplotlib and what is the use of pyplot?
  62. Where we use the pandas?
  63. Why we have used the scikit-learn?
  64. What is the use of xlabel and ylabel function?
  65. How we can label the X-axis and Y-axis in graph?
  66. What is the use of Scipy in this program?

# CONTENT BEYOND SYLLABUS

**TensorFlow**

Tensorflow is a computational framework for building machine learning models. TensorFlow provides a variety of different toolkits that allow you to construct models at your preferred level of abstraction. You can use lower-level APIs to build models by defining a series of mathematical operations. Alternatively, you can use higher-level APIs (like tf.estimator) to specify predefined architectures, such as linear regressors or neural networks.

**Additional Program 1:** Hello World import tensorflow as tf

# Simple hello world using TensorFlow hello = tf.constant('Hello, TensorFlow!')

# Start tf session sess = tf.Session()

# Run graph print(sess.run(hello))

# Output:

Hello, TensorFlow!

**Additional Program 2:** Basic operations from future import print\_function

import tensorflow as tf

# Basic constant operations

# The value returned by the constructor represents the output # of the Constant op.

a = tf.constant(2) b = tf.constant(3)

# Launch the default graph. with tf.Session() as sess:

print("a=2, b=3")

print("Addition with constants: %i" % sess.run(a+b)) print("Multiplication with constants: %i" % sess.run(a\*b))

# Basic Operations with variable as graph input

# The value returned by the constructor represents the output # of the Variable op. (define as input when running session) # tf Graph input

a = tf.placeholder(tf.int16) b = tf.placeholder(tf.int16)

# Define some operations add = tf.add(a, b)

mul = tf.multiply(a, b)

# Launch the default graph. with tf.Session() as sess:

# Run every operation with variable input

print("Addition with variables: %i" % sess.run(add, feed\_dict={a: 2, b: 3})) print("Multiplication with variables: %i" % sess.run(mul, feed\_dict={a: 2, b: 3}))

# More in details:

# Matrix Multiplication from TensorFlow official tutorial

# Create a Constant op that produces a 1x2 matrix. The op is

# added as a node to the default graph. #

# The value returned by the constructor represents the output # of the Constant op.

matrix1 = tf.constant([[3., 3.]])

# Create another Constant that produces a 2x1 matrix. matrix2 = tf.constant([[2.],[2.]])

# Create a Matmul op that takes 'matrix1' and 'matrix2' as inputs. # The returned value, 'product', represents the result of the matrix # multiplication.

product = tf.matmul(matrix1, matrix2)

# To run the matmul op we call the session 'run()' method, passing 'product' # which represents the output of the matmul op. This indicates to the call

# that we want to get the output of the matmul op back. #

# All inputs needed by the op are run automatically by the session. They # typically are run in parallel.

#

# The call 'run(product)' thus causes the execution of threes ops in the # graph: the two constants and matmul.

#

# The output of the op is returned in 'result' as a numpy `ndarray` object. with tf.Session() as sess:

result = sess.run(product) print(result)

# ==> [[ 12.]]

Output:

a=2, b=3

Addition with constants: 5

Multiplication with constants: 6 Addition with variables: 5 Multiplication with variables: 6 [[12.]]

Additional Program 3: Simple Neural Network from future import print\_function

# Import MNIST data

from tensorflow.examples.tutorials.mnist import input\_data mnist = input\_data.read\_data\_sets("/tmp/data/", one\_hot=True) import tensorflow as tf

# Parameters learning\_rate = 0.1

num\_steps = 500

batch\_size = 128

display\_step = 100

# Network Parameters

n\_hidden\_1 = 256 # 1st layer number of neurons n\_hidden\_2 = 256 # 2nd layer number of neurons num\_input = 784 # MNIST data input (img shape: 28\*28) num\_classes = 10 # MNIST total classes (0-9 digits)

# tf Graph input

X = tf.placeholder("float", [None, num\_input]) Y = tf.placeholder("float", [None, num\_classes]) # Store layers weight & bias

weights = {

'h1': tf.Variable(tf.random\_normal([num\_input, n\_hidden\_1])), 'h2': tf.Variable(tf.random\_normal([n\_hidden\_1, n\_hidden\_2])), 'out': tf.Variable(tf.random\_normal([n\_hidden\_2, num\_classes]))

}

biases = {

'b1': tf.Variable(tf.random\_normal([n\_hidden\_1])), 'b2': tf.Variable(tf.random\_normal([n\_hidden\_2])),

'out': tf.Variable(tf.random\_normal([num\_classes]))

}

# Create model def neural\_net(x):

# Hidden fully connected layer with 256 neurons layer\_1 = tf.add(tf.matmul(x, weights['h1']), biases['b1']) # Hidden fully connected layer with 256 neurons

layer\_2 = tf.add(tf.matmul(layer\_1, weights['h2']), biases['b2']) # Output fully connected layer with a neuron for each class out\_layer = tf.matmul(layer\_2, weights['out']) + biases['out'] return out\_layer

# Construct model logits = neural\_net(X)

prediction = tf.nn.softmax(logits) # Define loss and optimizer

loss\_op = tf.reduce\_mean(tf.nn.softmax\_cross\_entropy\_with\_logits( logits=logits, labels=Y))

optimizer = tf.train.AdamOptimizer(learning\_rate=learning\_rate) train\_op = optimizer.minimize(loss\_op)

# Evaluate model

correct\_pred = tf.equal(tf.argmax(prediction, 1), tf.argmax(Y, 1)) accuracy = tf.reduce\_mean(tf.cast(correct\_pred, tf.float32))

# Initialize the variables (i.e. assign their default value) init = tf.global\_variables\_initializer()

# Start training

with tf.Session() as sess: # Run the initializer sess.run(init)

for step in range(1, num\_steps+1):

batch\_x, batch\_y = mnist.train.next\_batch(batch\_size) # Run optimization op (backprop)

sess.run(train\_op, feed\_dict={X: batch\_x, Y: batch\_y}) if step % display\_step == 0 or step == 1:

# Calculate batch loss and accuracy

loss, acc = sess.run([loss\_op, accuracy], feed\_dict={X: batch\_x,

Y: batch\_y}) print("Step " + str(step) + ", Minibatch Loss= " + \

"{:.4f}".format(loss) + ", Training Accuracy= " + \ "{:.3f}".format(acc))

print("Optimization Finished!")

# Calculate accuracy for MNIST test images print("Testing Accuracy:", \

sess.run(accuracy, feed\_dict={X: mnist.test.images,

Y: mnist.test.labels}))

Output:

Step 1, Minibatch Loss= 15566.5215, Training Accuracy= 0.297

Step 100, Minibatch Loss= 243.3625, Training Accuracy= 0.922

Step 200, Minibatch Loss= 110.1580, Training Accuracy= 0.906

Step 300, Minibatch Loss= 89.0538, Training Accuracy= 0.883

Step 400, Minibatch Loss= 74.8770, Training Accuracy= 0.859

Step 500, Minibatch Loss= 37.8181, Training Accuracy= 0.852 Optimization Finished!

Testing Accuracy: 0.8231

# DO’S AND DON’TS

**Do’s**

Do wear ID card and follow dress code.

* Do log off the computers when you finish.
* Do ask for assistance if you need help.
* Do keep your voice low when speaking to others in the LAB.
* Do ask for assistance in downloading any software.
* Do make suggestions as to how we can improve the LAB.
* In case of any hardware related problem, ask LAB in charge for solution.
* If you are the last one leaving the LAB, make sure that the staff in charge of the LAB is informed to close the LAB.
* Be on time to LAB sessions.
* Do keep the LAB as clean as possible.

# Don’ts

* Do not use mobile phone inside the lab.
* Don’t do anything that can make the LAB dirty (like eating, throwing waste papers etc).
* Do not carry any external devices without permission.
* Don’t move the chairs of the LAB.
* Don’t interchange any part of one computer with another.
* Don’t leave the computers of the LAB turned on while leaving the LAB.
* Do not install or download any software or modify or delete any system files on any lab computers.
* Do not damage, remove, or disconnect any labels, parts, cables, or equipment.
* Don’t attempt to bypass the computer security system.
* Do not read or modify other user’s file.
* If you leave the lab, do not leave your personal belongings unattended. We are not responsible for any theft.
* Do not install or download any software or modify or delete any system files on any lab computers.
* Do not damage, remove, or disconnect any labels, parts, cables, or equipment.
* Don’t attempt to bypass the computer security system.
* Do not read or modify other user’s file.
* If you leave the lab, do not leave your personal belongings unattended. We are not responsible for any theft.