**List of Experiments**

(The following tasks can be implemented in a language of your choice or any tools available)

1. Implement the CANDIDATE – ELIMINATION algorithm. Show how it is used to learn from training examples.
2. **import** **random**
3. **import** **csv**
4. **import** **warnings**
5. warnings.filterwarnings('ignore')
6. **from** **matplotlib.patches** **import** Rectangle
7. **from** **IPython.html.widgets** **import** interact, IntSlider
8. **def** g\_0(n):
9. **return** ("?",)\*n
10. **def** s\_0(n):
11. **return** ('0',)\*n
12. **def** more\_general(h1, h2):
13. more\_general\_parts = []
14. **for** x, y **in** zip(h1, h2):
15. mg = x == "?" **or** (x != "0" **and** (x == y **or** y == "0"))
16. more\_general\_parts.append(mg)
17. **return** all(more\_general\_parts)
18. l1 = [1, 2, 3]
19. l2 = [3, 4, 5]
20. list(zip(l1, l2))
21. *# min\_generalizations*
22. **def** fulfills(example, hypothesis):
23. *### the implementation is the same as for hypotheses:*
24. **return** more\_general(hypothesis, example)
25. **def** min\_generalizations(h, x):
26. h\_new = list(h)
27. **for** i **in** range(len(h)):
28. **if** **not** fulfills(x[i:i+1], h[i:i+1]):
29. h\_new[i] = '?' **if** h[i] != '0' **else** x[i]
30. **return** [tuple(h\_new)]
31. min\_generalizations(h=('0', '0' , 'sunny'),
32. x=('rainy', 'windy', 'cloudy'))
33. **def** min\_specializations(h, domains, x):
34. results = []
35. **for** i **in** range(len(h)):
36. **if** h[i] == "?":
37. **for** val **in** domains[i]:
38. **if** x[i] != val:
39. h\_new = h[:i] + (val,) + h[i+1:]
40. results.append(h\_new)
41. **elif** h[i] != "0":
42. h\_new = h[:i] + ('0',) + h[i+1:]
43. results.append(h\_new)
44. **return** results
45. min\_specializations(h=('?', 'x',),
46. domains=[['a', 'b', 'c'], ['x', 'y']],
47. x=('b', 'x'))
48. examples = [tuple(line) **for** line **in** csv.reader(csvFile)]
49. *#examples = [('sunny', 'warm', 'normal', 'strong', 'warm', 'same',True),*
50. *# ('sunny', 'warm', 'high', 'strong', 'warm', 'same',True),*
51. *# ('rainy', 'cold', 'high', 'strong', 'warm', 'change',False),*
52. *# ('sunny', 'warm', 'high', 'strong', 'cool', 'change',True)]*
53. examples
54. **def** get\_domains(examples):
55. d = [set() **for** i **in** examples[0]]
56. **for** x **in** examples:
57. **for** i, xi **in** enumerate(x):
58. d[i].add(xi)
59. **return** [list(sorted(x)) **for** x **in** d]
60. get\_domains(examples)
61. **def** candidate\_elimination(examples):
62. domains = get\_domains(examples)[:-1]
64. G = set([g\_0(len(domains))])
65. S = set([s\_0(len(domains))])
66. i=0
67. *#print("\n G[{0}]:".format(i),G)*
68. *#print("\n S[{0}]:".format(i),S)*
69. **for** xcx **in** examples:
70. i=i+1
71. x, cx = xcx[:-1], xcx[-1] *# Splitting data into attributes and decisions*
72. **if** cx=='Y': *# x is positive example*
73. G = {g **for** g **in** G **if** fulfills(x, g)}
74. S = generalize\_S(x, G, S)
75. **else**: *# x is negative example*
76. S = {s **for** s **in** S **if** **not** fulfills(x, s)}
77. G = specialize\_G(x, domains, G, S)
78. *#print("\n G[{0}]:".format(i),G)*
79. *#print("\n S[{0}]:".format(i),S)*
80. **return** G,S
81. **def** generalize\_S(x, G, S):
82. S\_prev = list(S)
83. **for** s **in** S\_prev:
84. **if** s **not** **in** S:
85. **continue**
86. **if** **not** fulfills(x, s):
87. S.remove(s)
88. Splus = min\_generalizations(s, x)
89. *## keep only generalizations that have a counterpart in G*
90. S.update([h **for** h **in** Splus **if** any([more\_general(g,h)
91. **for** g **in** G])])
92. *## remove hypotheses less specific than any other in S*
93. S.difference\_update([h **for** h **in** S **if**
94. any([more\_general(h, h1)
95. **for** h1 **in** S **if** h != h1])])
96. **return** S
97. **def** specialize\_G(x, domains, G, S):
98. G\_prev = list(G)
99. **for** g **in** G\_prev:
100. **if** g **not** **in** G:
101. **continue**
102. **if** fulfills(x, g):
103. G.remove(g)
104. Gminus = min\_specializations(g, domains, x)
105. *## keep only specializations that have a conuterpart in S*
106. G.update([h **for** h **in** Gminus **if** any([more\_general(h, s)
107. **for** s **in** S])])
108. *## remove hypotheses less general than any other in G*
109. G.difference\_update([h **for** h **in** G **if**
110. any([more\_general(g1, h)
111. **for** g1 **in** G **if** h != g1])])
112. **return** G
113. G, S = candidate\_elimination(examples)
114. print("G[4] =", G)
115. print("S[4] =", S)
116. **class** **HypothesisNode**(object):
117. **def** \_\_init\_\_(self, h, level=0, parents=**None**):
118. self.h = h
119. self.level = level
120. **if** parents **is** **None**:
121. parents = []
122. self.parents = set(parents)
123. **def** \_\_repr\_\_(self):
124. **return** "HypothesisNode(**{}**, **{}**, **{}**)".format(self.h, self.level, self.parents)
126. **def** build\_hypothesis\_space(G, S):
127. levels = [[HypothesisNode(x, 0) **for** x **in** G]]
128. curlevel = 1
129. **def** next\_level(h, S):
130. **for** s **in** S:
131. **for** i **in** range(len(h)):
132. **if** h[i] == '?' **and** s[i] != '?':
133. **yield** h[:i] + (s[i],) + h[i+1:]
134. nextLvl = {}
135. **while** **True**:
136. **for** n **in** levels[-1]:
137. **for** hyp **in** next\_level(n.h, S):
138. **if** hyp **in** nextLvl:
139. nextLvl[hyp].parents.add(n)
140. **else**:
141. nextLvl[hyp] = HypothesisNode(hyp, curlevel, [n])
142. **if** **not** nextLvl:
143. **break**
144. levels.append(list(nextLvl.values()))
145. curlevel += 1
146. nextLvl = {}
147. **return** levels
148. **import** **networkx** **as** **nx**
150. levels = build\_hypothesis\_space(G, S)
151. g = nx.Graph()
152. **for** nodes **in** levels:
153. **for** n **in** nodes:
154. **for** p **in** n.parents:
155. g.add\_edge(n.h, p.h)
156. pos = {}
157. ymin = 0.1
158. ymax = 0.9
159. **for** nodes, y **in** [(levels[0], ymin), (levels[-1], ymax)]:
160. xvals = np.linspace(0, 1, len(nodes))
161. **for** x, n **in** zip(xvals, nodes):
162. pos[n.h] = [x, y]
163. pos = nx.layout.fruchterman\_reingold\_layout(g, pos=pos, fixed=pos.keys())
164. nx.draw\_networkx\_edges(g, pos=pos, alpha=0.25)
165. nx.draw\_networkx\_labels(g, pos=pos)
166. plt.box(**True**)
167. plt.xticks([])
168. plt.yticks([])
169. plt.xlim(-1, 2)
170. gcf().set\_size\_inches((10, 10))
171. plt.show()
172. print()
173. draw\_hypothesis\_space(G, S)

2)Write a program to implement Linear Regression and Logistic Regression

Data Set provided: Income\_Data.csv for Linear regression

**LINEAR REGRESSION**

import matplotlib.pyplot as plt

import pandas as pd

#importing dataset

dataset= pd.read\_csv('Income\_Data.csv')

#dataset=pd.read\_csv('testing.csv')

dataset.isnull().any(axis=0)

features= dataset.iloc[:,:-1].values

labels= dataset.iloc[:,-1].values

#splitting

from sklearn.cross\_validation import train\_test\_split

features\_train, features\_test, labels\_train, labels\_test= train\_test\_split(features, labels, test\_size=0.1, random\_state=0)

# fitting simple linear regression to training set

from sklearn.linear\_model import LinearRegression

regressor= LinearRegression()

regressor.fit(features\_train, labels\_train)

#prdicting

labels\_pred= regressor.predict(features\_test)

#model score

Score=regressor.score(features\_test, labels\_test)

Score\_train=regressor.score(features\_train, labels\_train)

#visualization

plt.scatter(features\_train, labels\_train, color='red')

plt.plot(features\_train, regressor.predict(features\_train),color='blue')

plt.title("Income vs Experience")

plt.xlabel("ML-Experience")

plt.ylabel("Income")

plt.show()

#Visualize test result

plt.scatter(features\_test, labels\_test,color="red")

plt.plot(features\_train, regressor.predict(features\_train),color='blue')

plt.title("Income vs Experience")

plt.xlabel("ML-Experience")

plt.ylabel("Income")

plt.show()

**LOGISTIC REGRESSION**

Data Set provided: Social\_Network\_Ads.csv for Linear regression

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

df= pd.read\_csv("Social\_Network\_Ads .csv")

df.isnull().any(axis=0)

features=df.iloc[:,2:4].values

labels=df.iloc[:,4].values

from sklearn.cross\_validation import train\_test\_split

f\_train, f\_test, l\_train, l\_test=train\_test\_split(features, labels, test\_size=0.25, random\_state=0 )

# feature scaling

from sklearn.preprocessing import StandardScaler

sc=StandardScaler()

f\_train=sc.fit\_transform(f\_train)

f\_test=sc.fit\_transform(f\_test)

#fitting logistic regression to the training set

from sklearn.linear\_model import LogisticRegression

lr=LogisticRegression(random\_state=0)

lr.fit(f\_train, l\_train)

# predict

labels\_pred= lr.predict(f\_test)

# making confusion matrix

from sklearn.metrics import confusion\_matrix

cm=confusion\_matrix(l\_test,labels\_pred)

score=lr.score(f\_test,l\_test)

3) Implement the ID3 algorithm for learning Boolean–valued functions for classifying the training examples by searching through the space of a Decision Tree.

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

df=pd.read\_csv("Social\_Network\_Ads .csv")

df.isnull().any(axis=0)

features=df.iloc[:,2:4]

label=df.iloc[:,-1]

# feature scaling

from sklearn.preprocessing import StandardScaler

sc=StandardScaler()

features=sc.fit\_transform(features)

# splitting

from sklearn.cross\_validation import train\_test\_split

f\_train, f\_test, l\_train, l\_test=train\_test\_split(features, label, test\_size=0.25, random\_state=0)

# fititng decision tree classification

from sklearn.tree import DecisionTreeClassifier

classifier=DecisionTreeClassifier(criterion='entropy', random\_state=0)

classifier.fit(f\_train, l\_train)

# predict

labels\_pred=classifier.predict(f\_test)

# confusion matrix

from sklearn.metrics import confusion\_matrix

cm= confusion\_matrix(l\_test, labels\_pred)

#score

score=classifier.score(f\_test, l\_test)

# visualizing

from matplotlib.colors import ListedColormap

features\_set, labels\_set = f\_train, l\_train

X1, X2 = np.meshgrid(np.arange(start = features\_set[:, 0].min() - 1, stop = features\_set[:, 0].max() + 1, step = 0.01),

np.arange(start = features\_set[:, 1].min() - 1, stop = features\_set[:, 1].max() + 1, step = 0.01))

plt.contourf(X1,X2,classifier.predict(np.array([X1.ravel(),X2.ravel()]).T).reshape(X1.shape),alpha = 0.75, cmap = ListedColormap(('red', 'green')))

plt.xlim(X1.min(), X1.max())

plt.ylim(X2.min(), X2.max())

for i, j in enumerate(np.unique(labels\_set)):

plt.scatter(features\_set[labels\_set == j, 0], features\_set[labels\_set == j, 1],

c = ListedColormap(('red', 'green'))(i), label = j)

plt.title('SVM (Training set)')

plt.xlabel('Age')

plt.ylabel('Estimated Salary')

plt.legend()

plt.show()

# Visualising the Test set results

from matplotlib.colors import ListedColormap

features\_set, labels\_set = f\_test, l\_test

X1, X2 = np.meshgrid(np.arange(start = features\_set[:, 0].min() - 1, stop = features\_set[:, 0].max() + 1, step = 0.01),

np.arange(start = features\_set[:, 1].min() - 1, stop = features\_set[:, 1].max() + 1, step = 0.01))

plt.contourf(X1,X2,classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),

alpha = 0.75, cmap = ListedColormap(('red', 'green')))

plt.xlim(X1.min(), X1.max())

plt.ylim(X2.min(), X2.max())

for i, j in enumerate(np.unique(labels\_set)):

plt.scatter(features\_set[labels\_set == j, 0], features\_set[labels\_set == j, 1],

c = ListedColormap(('red', 'green'))(i), label = j)

plt.title('SVM (Test set)')

plt.xlabel('Age')

plt.ylabel('Estimated Salary')

plt.legend()

plt.show()

4) Design and implement Naïve Bayes Algorithm for learning and classifying TEXT DOCUMENTS.

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.cross\_validation import train\_test\_split

from sklearn.naive\_bayes import GaussianNB, BernoulliNB, MultinomialNB

#importing data set

data=pd.read\_csv("training\_titanic.csv")

#covert categorical values to numerical

data["sex\_cleaned"]=np.where(data["Sex"]=="male",0,1)

data["Embarked\_cleaned"]=np.where(data["Embarked"]=="S",0,

np.where(data["Embarked"]=="C",1,

np.where(data["Embarked"]=="Q",2,3)

)

)

#cleaning dataset og NaN

data=data[["Fare", "Survived"]].dropna(axis=0, how='any')

data=data[["Survived","Pclass","sex\_cleaned","Age","SibSp","Parch","Fare","Embarked\_cleaned"]].dropna(axis=0,how='any')

#split dataset in training and test datasets

f\_train, f\_test=train\_test\_split(data, test\_size=0.5, random\_state=0)

#gaussian

gnb=GaussianNB()

used\_features=["Fare",]

#train classifier

gnb.fit(f\_train[used\_features].values, f\_train["Survived"])

labels\_pred=gnb.predict(f\_test[used\_features])

# making confusion matrix

from sklearn.metrics import confusion\_matrix

cm\_gnb= confusion\_matrix(f\_test["Survived"],labels\_pred)

#multinomial

mnb=MultinomialNB()

#train classifier

mnb.fit(f\_train[used\_features].values,f\_train["Survived"])

labels\_pred= mnb.predict(f\_test[used\_features])

cm\_mnb=confusion\_matrix(f\_test["Survived"],labels\_pred)

#bernoulli

bnb=BernoulliNB()

#train classifer

bnb.fit(f\_train[used\_features].values, f\_train["Survived"])

labels\_pred=bnb.predict(f\_test[used\_features])

cm\_bnb=confusion\_matrix(f\_test["Survived"],labels\_pred)

5) Implement K-Nearest Neighbor algorithm to classify the iris data set. Calculate the score also.

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

df=pd.read\_csv("Social\_Network\_Ads .csv")

features=df.iloc[:,2:-1].values

labels=df.iloc[:,-1].values

#splitting dataset

from sklearn.cross\_validation import train\_test\_split

f\_train, f\_test, l\_train, l\_test= train\_test\_split(features, labels, test\_size=0.25, random\_state=0)

# feature scaling

from sklearn.preprocessing import StandardScaler

sc=StandardScaler()

f\_train=sc.fit\_transform(f\_train)

f\_test=sc.fit\_transform(f\_test)

# fitting k-nn to training set

from sklearn.neighbors import KNeighborsClassifier

classifier=KNeighborsClassifier(n\_neighbors=5,p=1)

classifier.fit(f\_train,l\_train)

pred=classifier.predict(f\_test)

from sklearn.metrics import confusion\_matrix

cm=confusion\_matrix(l\_test,pred)

score=classifier.score(f\_test,l\_test)

# Visualising the Training set results

from matplotlib.colors import ListedColormap

features\_set, labels\_set = f\_train, l\_train

features1, features2 = np.meshgrid(np.arange(start = features\_set[:, 0].min() - 1, stop = features\_set[:, 0].max() + 1, step = 0.01),

np.arange(start = features\_set[:, 1].min() - 1, stop = features\_set[:, 1].max() + 1, step = 0.01))

plt.contourf(features1, features2, classifier.predict(np.array([features1.ravel(), features2.ravel()]).T).reshape(features1.shape),

alpha = 0.75, cmap = ListedColormap(('red', 'green')))

plt.xlim(features1.min(), features1.max())

plt.ylim(features2.min(), features2.max())

for i, j in enumerate(np.unique(labels\_set)):

plt.scatter(features\_set[labels\_set == j, 0], features\_set[labels\_set == j, 1],

c = ListedColormap(('red', 'green'))(i), label = j)

plt.title('K-NN (Training set)')

plt.xlabel('Age')

plt.ylabel('Estimated Salary')

plt.legend()

plt.show()

# Visualising the Test set results

# Visualising the Training set results

from matplotlib.colors import ListedColormap

features\_set, labels\_set = f\_test, l\_test

features1, features2 = np.meshgrid(np.arange(start = features\_set[:, 0].min() - 1, stop = features\_set[:, 0].max() + 1, step = 0.01),

np.arange(start = features\_set[:, 1].min() - 1, stop = features\_set[:, 1].max() + 1, step = 0.01))

plt.contourf(features1, features2, classifier.predict(np.array([features1.ravel(), features2.ravel()]).T).reshape(features1.shape),

alpha = 0.75, cmap = ListedColormap(('red', 'green')))

plt.xlim(features1.min(), features1.max())

plt.ylim(features2.min(), features2.max())

for i, j in enumerate(np.unique(labels\_set)):

plt.scatter(features\_set[labels\_set == j, 0], features\_set[labels\_set == j, 1],

c = ListedColormap(('red', 'green'))(i), label = j)

plt.title('K-NN (Test set)')

plt.xlabel('Age')

plt.ylabel('Estimated Salary')

plt.legend()

plt.show()

6) Write a program to implement Support Vector Machine. Also discuss the confusion matrix and score of model.

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

df=pd.read\_csv("Social\_Network\_Ads .csv")

features=df.iloc[:,2:4].values

label=df.iloc[:,-1].values

# spliting dataset

from sklearn.cross\_validation import train\_test\_split

f\_train, f\_test, l\_train, l\_test= train\_test\_split(features, label, test\_size=0.25, random\_state=0)

#scaing

from sklearn.preprocessing import StandardScaler

sc=StandardScaler()

f\_train=sc.fit\_transform(f\_train)

f\_test=sc.transform(f\_test)

#fitting SVM to training set

from sklearn.svm import SVC

svc=SVC(kernel="linear", random\_state=0)

svc.fit(f\_train,l\_train)

# prediction of test features

labels\_pred=svc.predict(f\_test)

# confusion matrix

from sklearn.metrics import confusion\_matrix

cm= confusion\_matrix(l\_test, labels\_pred)

#calculate score

scores= svc.score(f\_test, l\_test)

# visualizing

from matplotlib.colors import ListedColormap

features\_set, labels\_set = f\_train, l\_train

X1, X2 = np.meshgrid(np.arange(start = features\_set[:, 0].min() - 1, stop = features\_set[:, 0].max() + 1, step = 0.01),

np.arange(start = features\_set[:, 1].min() - 1, stop = features\_set[:, 1].max() + 1, step = 0.01))

plt.contourf(X1, X2, svc.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),

alpha = 0.75, cmap = ListedColormap(('red', 'green')))

plt.xlim(X1.min(), X1.max())

plt.ylim(X2.min(), X2.max())

for i, j in enumerate(np.unique(labels\_set)):

plt.scatter(features\_set[labels\_set == j, 0], features\_set[labels\_set == j, 1],

c = ListedColormap(('red', 'green'))(i), label = j)

plt.title('SVM (Training set)')

plt.xlabel('Age')

plt.ylabel('Estimated Salary')

plt.legend()

plt.show()

# Visualising the Test set results

from matplotlib.colors import ListedColormap

features\_set, labels\_set = f\_test, l\_test

X1, X2 = np.meshgrid(np.arange(start = features\_set[:, 0].min() - 1, stop = features\_set[:, 0].max() + 1, step = 0.01),

np.arange(start = features\_set[:, 1].min() - 1, stop = features\_set[:, 1].max() + 1, step = 0.01))

plt.contourf(X1, X2, svc.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),

alpha = 0.75, cmap = ListedColormap(('red', 'green')))

plt.xlim(X1.min(), X1.max())

plt.ylim(X2.min(), X2.max())

for i, j in enumerate(np.unique(labels\_set)):

plt.scatter(features\_set[labels\_set == j, 0], features\_set[labels\_set == j, 1],

c = ListedColormap(('red', 'green'))(i), label = j)

plt.title('SVM (Test set)')

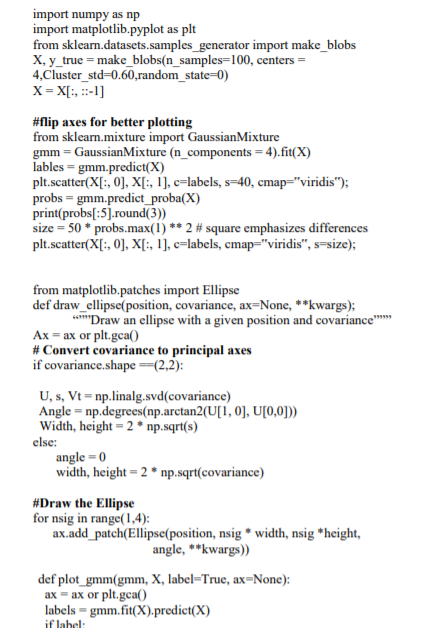
plt.xlabel('Age')

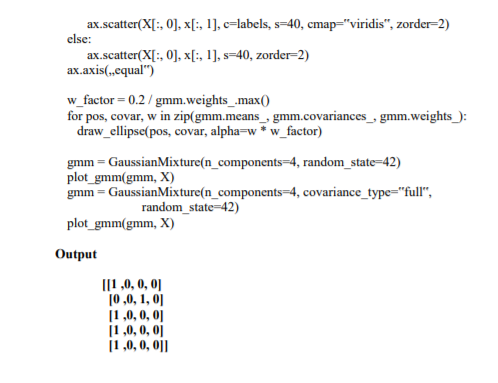
plt.ylabel('Estimated Salary')

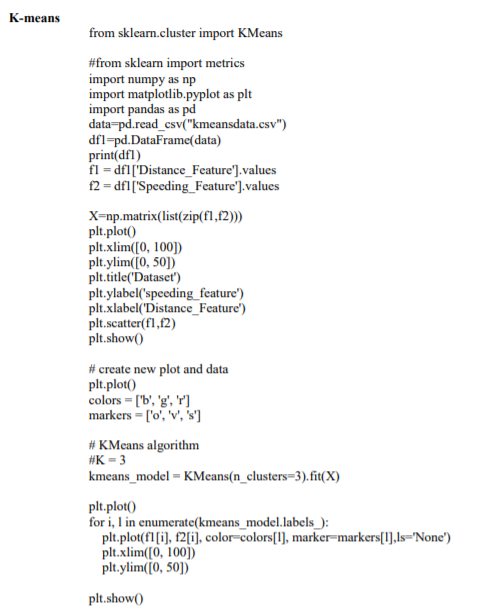
plt.legend()

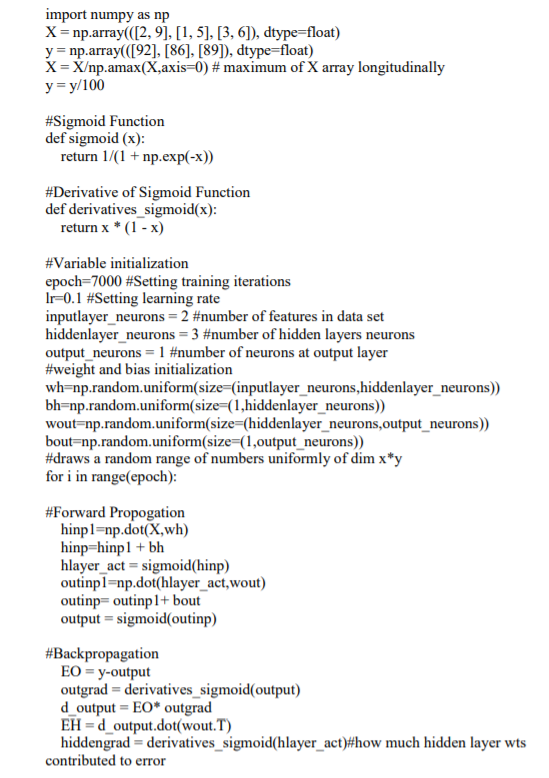
plt.show()

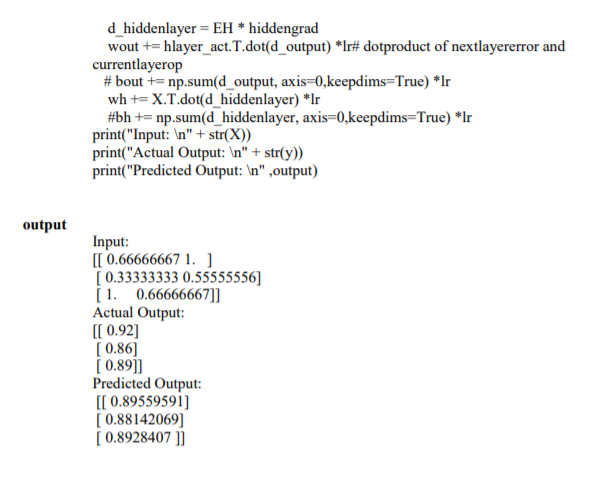
7) Apply EM algorithm to cluster a set of data and also apply K-Means algorithm on the same data set to compare two algorithms.



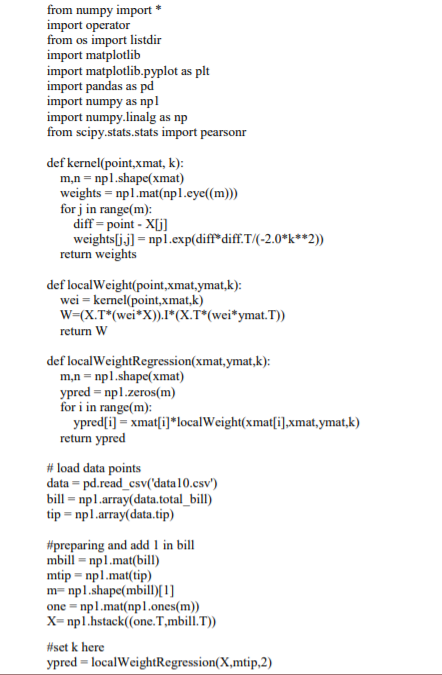




8)Build an Artificial Neural Network by implementing Back-Propagation algorithm and test the same using appropriate data set.



9)Implement the Non-Parametric Locally Weighted Regression Algorithm in order to fit data points. Select appropriate data set for your experiment and draw graph.



SortIndex = X[:,1].argsort(0)

xsort = X[SortIndex][:,0

10) Build a Face detection system to recognize faces in a frame or image. You can use OpenCV for this task.

Import cv2

face\_cascade= cv2.CascadeClassifier(‘haarcascade\_frontalface\_default.xml’)

eye\_cascade= cv2.CascadeClassifier(‘haarcascade\_eye.xml’)

img= cv2.imread(‘baby.jpg’)

gray= cv2.cvtColor(img,cv2.COLOR\_BGR2GRAY)

faces= face\_Casecade.detectMultiScale(gray,1.3,5)

for(x,y,w,h) in faces:

cv2.rectangle(img,(x,y),(x+w,y+h)),(255,0,0),2)

roi\_gray=gray[y:y+h, x:x+w]

roi\_color=img[y:y+h, x:x+w]

eye= eye\_cascade.detectMultiScale(roi\_gray)

for(ex,ey,ew,eh) in eyes:

cv.rectangle(roi\_color,(ex,ey),(ex+ew,ey+eh),(0,255,0),2)

cv2.imshow(‘img’,img)

k=cv2.waitKey(0)

stop=ord(“S”)

if k==stop:

cv2.destroyAllWindows()

elif( k == ord(‘q’)):

cv2.imwrite(‘marked1.png’,img)

cv2.destroyAllWindows()