**Linear Regression**

# Importing the libraries

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

# Importing the dataset

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

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

y = dataset.iloc[:, 1].values

# Splitting the dataset into the Training set and Test set

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

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 1/3, random\_state = 0)

#Fitting Simple Linear Regression to the training set

from sklearn.linear\_model import LinearRegression

regressor = LinearRegression()

regressor.fit(X\_train, y\_train)

#Predicting the test set results

y\_pred = regressor.predict(X\_test)

#Visualising the training set results

plt.scatter(X\_train, y\_train, color = 'red')

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

plt.title('Salary vs Experience (Training set)')

plt.xlabel('Years of Experience')

plt.ylabel('Salary')

plt.show()

#Visualising the test set results

plt.scatter(X\_test, y\_test, color = 'red')

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

plt.title('Salary vs Experience (Test set)')

plt.xlabel('Years of Experience')

plt.ylabel('Salary')

plt.show()

**Multiple Linear Regression**

# Importing the libraries

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

# Importing the dataset

dataset = pd.read\_csv('50\_Startups.csv')

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

y = dataset.iloc[:, 4].values

# Encoding the Independent Variable

from sklearn.preprocessing import LabelEncoder, OneHotEncoder

labelencoder\_X = LabelEncoder()

X[:, 3] = labelencoder\_X.fit\_transform(X[:, 3])

onehotencoder = OneHotEncoder(categorical\_features = [3])

X = onehotencoder.fit\_transform(X).toarray()

#Avoiding the Dummy Variable Trap

X = X[:, 1:]

# Splitting the dataset into the Training set and Test set

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

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.2, random\_state = 0)

#Fitting Multiple linear regression to the training set

from sklearn.linear\_model import LinearRegression

regressor = LinearRegression()

regressor.fit(X\_train, y\_train)

#Predicting the test set results

y\_pred = regressor.predict(X\_test)

#Using Backward Elimination

import statsmodels.formula.api as sm

X = np.append(arr = np.ones((50, 1)).astype(int), values = X, axis = 1)

X\_opt = X[:, [0, 1, 2, 3, 4, 5]]

regressor\_OLS = sm.OLS(endog = y, exog = X\_opt).fit()

regressor\_OLS.summary()

X\_opt = X[:, [0, 1, 3, 4, 5]]

regressor\_OLS = sm.OLS(endog = y, exog = X\_opt).fit()

regressor\_OLS.summary()

X\_opt = X[:, [0, 3, 4, 5]]

regressor\_OLS = sm.OLS(endog = y, exog = X\_opt).fit()

regressor\_OLS.summary()

X\_opt = X[:, [0, 3, 5]]

regressor\_OLS = sm.OLS(endog = y, exog = X\_opt).fit()

regressor\_OLS.summary()

X\_opt = X[:, [0, 3]]

regressor\_OLS = sm.OLS(endog = y, exog = X\_opt).fit()

regressor\_OLS.summary()

**Decision Tree (Regression)**

# Importing the libraries

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

# Importing the dataset

dataset = pd.read\_csv('Position\_Salaries.csv')

X = dataset.iloc[:, 1:2].values

y = dataset.iloc[:, 2].values

# Fitting the Decision Tree Regression Model to the dataset

from sklearn.tree import DecisionTreeRegressor

regressor = DecisionTreeRegressor(random\_state = 0)

regressor.fit(X, y)

# Predicting a new result

y\_pred = regressor.predict(6.5)

# Visualising the Decision Tree Regression results (for higher resolution and smoother curve)

X\_grid = np.arange(min(X), max(X), 0.0001)

X\_grid = X\_grid.reshape((len(X\_grid), 1))

plt.scatter(X, y, color = 'red')

plt.plot(X\_grid, regressor.predict(X\_grid), color = 'blue')

plt.title('Truth or Bluff (Decision Tree Regression Model)')

plt.xlabel('Position level')

plt.ylabel('Salary')

plt.show()

**Logistic Regression**

# Importing the libraries

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

# Importing the dataset

dataset = pd.read\_csv('Social\_Network\_Ads.csv')

X = dataset.iloc[:, [2,3]].values

y = dataset.iloc[:, 4].values

# Splitting the dataset into the Training set and Test set

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

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.25, random\_state = 0)

# Feature Scaling

from sklearn.preprocessing import StandardScaler

sc\_X = StandardScaler()

X\_train = sc\_X.fit\_transform(X\_train)

X\_test = sc\_X.transform(X\_test)

#fitting Logistic Regression to the training set

from sklearn.linear\_model import LogisticRegression

classifier = LogisticRegression(random\_state = 0)

classifier.fit(X\_train,y\_train)

#Predicting the test set results

y\_pred = classifier.predict(X\_test)

#Making Confusion Matrix

from sklearn.metrics import confusion\_matrix

cm = confusion\_matrix(y\_test, y\_pred)

#visualizing the Training set results

from matplotlib.colors import ListedColormap

X\_set, y\_set = X\_train, y\_train

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

np.arange(start = X\_set[:, 1].min() - 1, stop = X\_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(y\_set)):

plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1],

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

plt.title('Logistic Regression (Training set)')

plt.xlabel('Age')

plt.ylabel('Estimated Salary')

plt.legend()

plt.show()

#visualizing the Test set results

from matplotlib.colors import ListedColormap

X\_set, y\_set = X\_test, y\_test

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

np.arange(start = X\_set[:, 1].min() - 1, stop = X\_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(y\_set)):

plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1],

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

plt.title('Logistic Regression (Test set)')

plt.xlabel('Age')

plt.ylabel('Estimated Salary')

plt.legend()

plt.show()

**Decision Tree (Classification)**

# Importing the libraries

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

# Importing the dataset

dataset = pd.read\_csv('Social\_Network\_Ads.csv')

X = dataset.iloc[:, [2, 3]].values

y = dataset.iloc[:, 4].values

# Splitting the dataset into the Training set and Test set

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

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.25, random\_state = 0)

# Feature Scaling

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

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

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

# Fitting classifier to the Training set

from sklearn.tree import DecisionTreeClassifier

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

classifier.fit(X\_train,y\_train)

# Predicting the Test set results

y\_pred = classifier.predict(X\_test)

# Making the Confusion Matrix

from sklearn.metrics import confusion\_matrix

cm = confusion\_matrix(y\_test, y\_pred)

# Visualising the Training set results

from matplotlib.colors import ListedColormap

X\_set, y\_set = X\_train, y\_train

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

np.arange(start = X\_set[:, 1].min() - 1, stop = X\_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(y\_set)):

plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1],

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

plt.title('Decision Tree Classifier (Training set)')

plt.xlabel('Age')

plt.ylabel('Estimated Salary')

plt.legend()

plt.show()

# Visualising the Test set results

from matplotlib.colors import ListedColormap

X\_set, y\_set = X\_test, y\_test

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

np.arange(start = X\_set[:, 1].min() - 1, stop = X\_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(y\_set)):

plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1],

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

plt.title('Decision Tree Classifier (Test set)')

plt.xlabel('Age')

plt.ylabel('Estimated Salary')

plt.legend()

plt.show()

**K NN**

**CODE-1**

import csv

import random

import math

import operator

def loadDataset(filename, split, trainingSet=[] , testSet=[]):

with open(filename, 'rb') as csvfile:

lines = csv.reader(csvfile)

dataset = list(lines)

for x in range(2,len(dataset)):

for y in range(4):

dataset[x][y] = float(dataset[x][y])

if random.random() < split:

trainingSet.append(dataset[x])

else:

testSet.append(dataset[x])

def euclideanDistance(instance1, instance2, length):

distance = 0

for x in range(length):

distance += pow((instance1[x] - instance2[x]), 2)

return math.sqrt(distance)

def getNeighbors(trainingSet, testInstance, k):

distances = []

length = len(testInstance)-1

for x in range(len(trainingSet)):

dist = euclideanDistance(testInstance, trainingSet[x], length)

distances.append((trainingSet[x], dist))

distances.sort(key=operator.itemgetter(1))

neighbors = []

for x in range(k):

neighbors.append(distances[x][0])

return neighbors

def getResponse(neighbors):

classVotes = {}

for x in range(len(neighbors)):

response = neighbors[x][-1]

if response in classVotes:

classVotes[response] += 1

else:

classVotes[response] = 1

sortedVotes = sorted(classVotes.iteritems(), key=operator.itemgetter(1), reverse=True)

return sortedVotes[0][0]

def getAccuracy(testSet, predictions):

correct = 0

for x in range(len(testSet)):

if testSet[x][-1] == predictions[x]:

correct += 1

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

def main():

# prepare data

trainingSet=[]

testSet=[]

split = 0.67

loadDataset('data.csv', split, trainingSet, testSet)

print 'Train set: ' + repr(len(trainingSet))

print 'Test set: ' + repr(len(testSet))

# generate predictions

predictions=[]

k = 3

for x in range(len(testSet)):

neighbors = getNeighbors(trainingSet, testSet[x], k)

result = getResponse(neighbors)

predictions.append(result)

print('> predicted=' + repr(result) + ', actual=' + repr(testSet[x][-1]))

accuracy = getAccuracy(testSet, predictions)

print('Accuracy: ' + repr(accuracy) + '%')

main()

**CODE-2**

# Importing the libraries

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

# Importing the dataset

dataset = pd.read\_csv('Social\_Network\_Ads.csv')

X = dataset.iloc[:, [2, 3]].values

y = dataset.iloc[:, 4].values

# Splitting the dataset into the Training set and Test set

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

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.25, random\_state = 0)

# Feature Scaling

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

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

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

# Fitting classifier to the Training set

from sklearn.neighbors import KNeighborsClassifier

classifier = KNeighborsClassifier(n\_neighbors=5, p=2, metric='minkowski')

classifier.fit(X\_train,y\_train)

# Predicting the Test set results

y\_pred = classifier.predict(X\_test)

# Making the Confusion Matrix

from sklearn.metrics import confusion\_matrix

cm = confusion\_matrix(y\_test, y\_pred)

# Visualising the Training set results

from matplotlib.colors import ListedColormap

X\_set, y\_set = X\_train, y\_train

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

np.arange(start = X\_set[:, 1].min() - 1, stop = X\_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(y\_set)):

plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_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

from matplotlib.colors import ListedColormap

X\_set, y\_set = X\_test, y\_test

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

np.arange(start = X\_set[:, 1].min() - 1, stop = X\_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(y\_set)):

plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1],

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

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

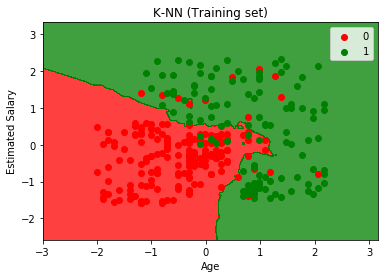
plt.xlabel('Age')

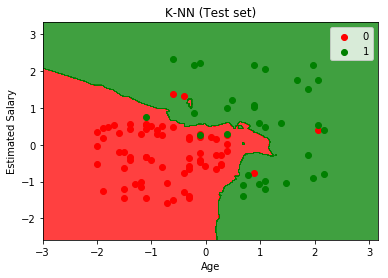
plt.ylabel('Estimated Salary')

plt.legend()

plt.show()

**OUTPUT:**





**K Means**

# -\*- coding: utf-8 -\*-

"""

Created on Wed Jan 31 22:18:37 2018

@author: Piyushjaiswal

"""

# K-Means Clustering

# Importing the libraries

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

# Importing the dataset

dataset = pd.read\_csv('Mall\_Customers.csv')

X = dataset.iloc[:, [3, 4]].values

#Using the elbow method to find the optimal number of clusters

from sklearn.cluster import KMeans

wcss = []

for i in range(1,11):

kmeans = KMeans(n\_clusters = i, init = 'k-means++', max\_iter = 300, n\_init = 10,random\_state = 0)

kmeans.fit(X)

wcss.append(kmeans.inertia\_)

plt.plot(range(1,11),wcss)

plt.title('The Elbow Method')

plt.xlabel('Number of Clusters')

plt.ylabel('WCSS')

plt.show()

#Applying K-means to the Dataset

kmeans = KMeans(n\_clusters = 5, max\_iter = 300, n\_init = 10,random\_state = 0)

y\_kmeans = kmeans.fit\_predict(X)

#Visualizing the clusters

plt.scatter(X[y\_kmeans == 0, 0], X[y\_kmeans == 0, 1], s = 100,c ='red', label = 'Careful')

plt.scatter(X[y\_kmeans ==1, 0], X[y\_kmeans ==1, 1], s = 100,c ='blue', label = 'Standard')

plt.scatter(X[y\_kmeans ==2, 0], X[y\_kmeans ==2, 1], s = 100,c ='green', label = 'Target')

plt.scatter(X[y\_kmeans ==3, 0], X[y\_kmeans ==3, 1], s = 100,c ='cyan', label = 'Careless')

plt.scatter(X[y\_kmeans ==4, 0], X[y\_kmeans ==4, 1], s = 100,c ='magenta', label = 'Sensible')

plt.scatter(kmeans.cluster\_centers\_[:, 0], kmeans.cluster\_centers\_[:, 1], s = 300,c ='yellow', label = 'Centroid')

plt.title('Clusters pf clients')

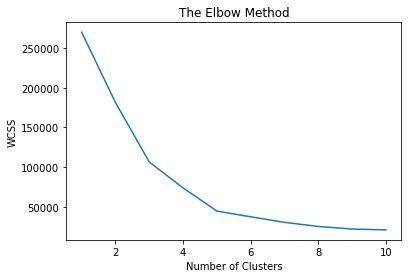
plt.xlabel('Annual Income (k$)')

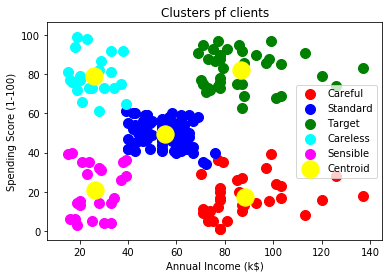
plt.ylabel('Spending Score (1-100)')

plt.legend()

plt.show()

**OUTPUT:**





**K Modes**

from kmodes import kmodes

from sklearn.datasets import load\_iris

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import accuracy\_score

import pandas as pd

import numpy as np

iris=load\_iris()

irisX=pd.DataFrame(iris.data,columns=iris.feature\_names)

irisY=pd.DataFrame(iris.target,columns=["Target"])

xTrain,xTest,yTrain,yTest=train\_test\_split(irisX,irisY, test\_size=0.2)

print(xTrain.shape)

print(yTrain.shape)

print(xTest.shape)

print(yTest.shape)

cl=kmodes.KModes(n\_clusters=3)

get\_ipython().magic(u'pinfo cl')

clusters=cl.fit\_predict(xTrain)

pred4=cl.predict(xTest)

accuracy=accuracy\_score(yTest,pred4)

print(accuracy)

**OUTPUT:**

(120, 4)

(120, 1)

(30, 4)

(30, 1)

**0.3**

**SVM REGRESSION**

**R Code:**

setwd("Z:\\MACHINE LEARNING")

install.packages("e1071")

library(e1071)

data <- read.csv( "svm\_regression.csv", header = TRUE)

plot(data, pch=16)

model <- svm(Y ~ X , data)

abline(model)

predictedY <- predict(model, data)

points(data$X, predictedY, col = "red", pch=4)

error <- data$Y - predictedY

svrPredictionRMSE <- sqrt(mean(error^2))

svrPredictionRMSE

tuneResult <- tune(svm, Y ~ X, data = data, ranges = list(epsilon = seq(0,1,0.1), cost = 2^(2:9)) )

print(tuneResult)

tuneResult <- tune(svm, Y ~ X, data = data, ranges = list(epsilon = seq(0,0.2,0.01), cost = 2^(2:9)) )

print(tuneResult)

tunedModel <- tuneResult$best.model

tunedModelY <- predict(tunedModel, data)

error <- data$Y - tunedModelY

plot(data, pch=16)

points(data$X, tunedModelY, col = "green", pch=4)

**OUTPUT:**

svrPredictionRMSE

[1] 3.157061

print(tuneResult)

Parameter tuning of ‘svm’:

- sampling method: 10-fold cross validation

- best parameters:

epsilon cost

0 4

- best performance: 14.52175

print(tuneResult)

Parameter tuning of ‘svm’:

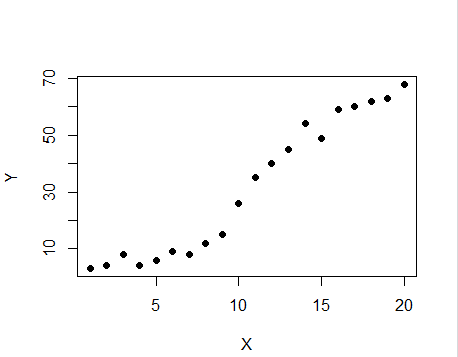
- sampling method: 10-fold cross validation

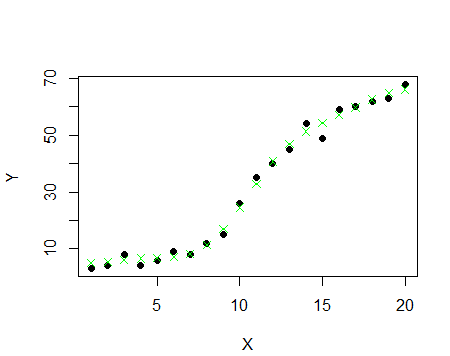
- best parameters:

epsilon cost

0.08 32

- best performance: 8.161363





**SVM CLASSIFICATION**

# Support Vector Machine(SVM)

# Importing the libraries

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

# Importing the dataset

dataset = pd.read\_csv('Social\_Network\_Ads.csv')

X = dataset.iloc[:, [2, 3]].values

y = dataset.iloc[:, 4].values

# Splitting the dataset into the Training set and Test set

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

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.25, random\_state = 0)

# Feature Scaling

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

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

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

# Fitting SVM to the Training set

from sklearn.svm import SVC

classifier = SVC(kernel = 'linear', random\_state = 0)

classifier.fit(X\_train,y\_train)

# Predicting the Test set results

y\_pred = classifier.predict(X\_test)

# Making the Confusion Matrix

from sklearn.metrics import confusion\_matrix

cm = confusion\_matrix(y\_test, y\_pred)

# Visualising the Training set results

from matplotlib.colors import ListedColormap

X\_set, y\_set = X\_train, y\_train

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

np.arange(start = X\_set[:, 1].min() - 1, stop = X\_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(y\_set)):

plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_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

X\_set, y\_set = X\_test, y\_test

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

np.arange(start = X\_set[:, 1].min() - 1, stop = X\_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(y\_set)):

plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1],

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

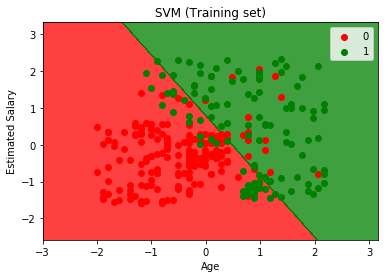
plt.title('SVM (Test set)')

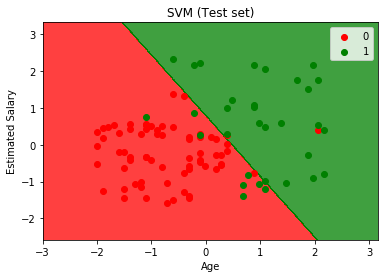
plt.xlabel('Age')

plt.ylabel('Estimated Salary')

plt.legend()

plt.show()





**VOTING**

from sklearn import datasets

from sklearn.base import BaseEstimator

from sklearn.base import ClassifierMixin

import operator

from sklearn import cross\_validation

iris = datasets.load\_iris()

X, y = iris.data[:, 1:3], iris.target

from sklearn import model\_selection

from sklearn.linear\_model import LogisticRegression

from sklearn.naive\_bayes import GaussianNB

from sklearn.ensemble import RandomForestClassifier

import numpy as np

clf1 = LogisticRegression(random\_state=1)

clf2 = RandomForestClassifier(random\_state=1)

clf3 = GaussianNB()

print('5-fold cross validation:\n')

labels = ['Logistic Regression', 'Random Forest', 'Naive Bayes']

for clf, label in zip([clf1, clf2, clf3], labels):

scores = model\_selection.cross\_val\_score(clf, X, y,

cv=5,

scoring='accuracy')

print("Accuracy: %0.2f (+/- %0.2f) [%s]"

% (scores.mean(), scores.std(), label))

class EnsembleClassifier(BaseEstimator, ClassifierMixin):

"""

Ensemble classifier for scikit-learn estimators.

Parameters

----------

clf : `iterable`

A list of scikit-learn classifier objects.

weights : `list` (default: `None`)

If `None`, the majority rule voting will be applied to the predicted class labels.

If a list of weights (`float` or `int`) is provided, the averaged raw probabilities (via `predict\_proba`)

will be used to determine the most confident class label.

"""

def \_\_init\_\_(self, clfs, weights=None):

self.clfs = clfs

self.weights = weights

def fit(self, X, y):

"""

Fit the scikit-learn estimators.

Parameters

----------

X : numpy array, shape = [n\_samples, n\_features]

Training data

y : list or numpy array, shape = [n\_samples]

Class labels

"""

for clf in self.clfs:

clf.fit(X, y)

def predict(self, X):

"""

Parameters

----------

X : numpy array, shape = [n\_samples, n\_features]

Returns

----------

maj : list or numpy array, shape = [n\_samples]

Predicted class labels by majority rule

"""

self.classes\_ = np.asarray([clf.predict(X) for clf in self.clfs])

if self.weights:

avg = self.predict\_proba(X)

maj = np.apply\_along\_axis(lambda x: max(enumerate(x), key=operator.itemgetter(1))[0], axis=1, arr=avg)

else:

maj = np.asarray([np.argmax(np.bincount(self.classes\_[:,c])) for c in range(self.classes\_.shape[1])])

return maj

def predict\_proba(self, X):

"""

Parameters

----------

X : numpy array, shape = [n\_samples, n\_features]

Returns

----------

avg : list or numpy array, shape = [n\_samples, n\_probabilities]

Weighted average probability for each class per sample.

"""

self.probas\_ = [clf.predict\_proba(X) for clf in self.clfs]

avg = np.average(self.probas\_, axis=0, weights=self.weights)

return avg

np.random.seed(123)

eclf = EnsembleClassifier(clfs=[clf1, clf2, clf3], weights=[1,1,1])

for clf, label in zip([clf1, clf2, clf3, eclf], ['Logistic Regression', 'Random Forest', 'naive Bayes', 'Ensemble']):

scores = cross\_validation.cross\_val\_score(clf, X, y, cv=5, scoring='accuracy')

print("Accuracy: %0.2f (+/- %0.2f) [%s]" % (scores.mean(), scores.std(), label))

**OUTPUT:**

5-fold cross validation:

Accuracy: 0.90 (+/- 0.05) [Logistic Regression]

Accuracy: 0.93 (+/- 0.05) [Random Forest]

Accuracy: 0.91 (+/- 0.04) [Naive Bayes]

**MAJORITY VOTING:**

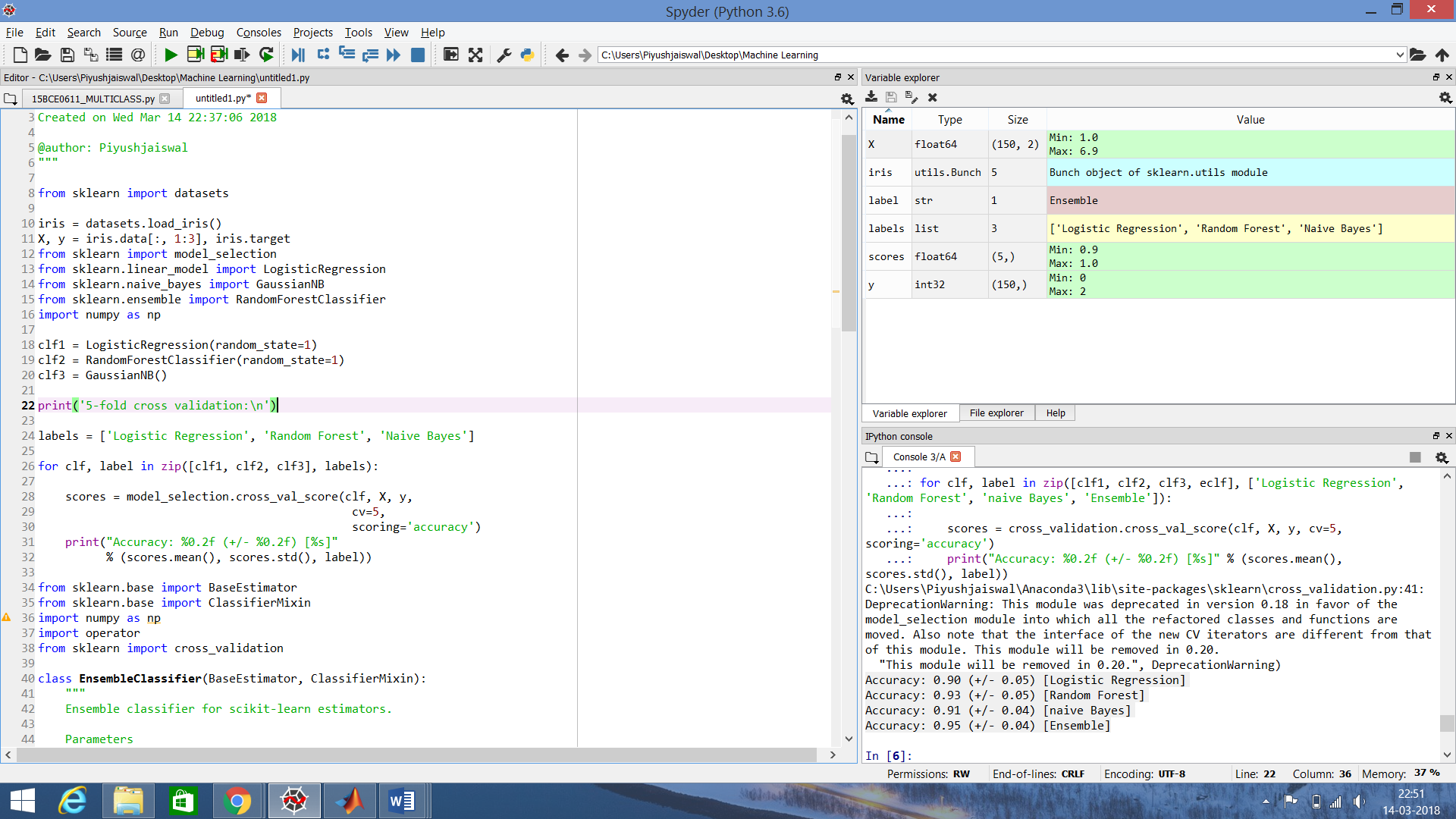
Accuracy: 0.90 (+/- 0.05) [Logistic Regression]

Accuracy: 0.93 (+/- 0.05) [Random Forest]

Accuracy: 0.91 (+/- 0.04) [naive Bayes]

Accuracy: 0.95 (+/- 0.04) [Ensemble]

**SCREENSHOT:**



**Random Forest Regression**

# Importing the libraries

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

# Importing the dataset

dataset = pd.read\_csv('Position\_Salaries.csv')

X = dataset.iloc[:, 1:2].values

y = dataset.iloc[:, 2].values

# Fitting the Random Forest Regression Model to the dataset

from sklearn.ensemble import RandomForestRegressor

regressor = RandomForestRegressor(n\_estimators = 300, random\_state = 0)

regressor.fit(X, y)

# Predicting a new result

y\_pred = regressor.predict(6.5)

# Visualising the Random Forest Regression results (for higher resolution and smoother curve)

X\_grid = np.arange(min(X), max(X), 0.01)

X\_grid = X\_grid.reshape((len(X\_grid), 1))

plt.scatter(X, y, color = 'red')

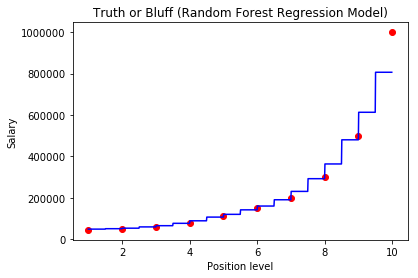
plt.plot(X\_grid, regressor.predict(X\_grid), color = 'blue')

plt.title('Truth or Bluff (Random Forest Regression Model)')

plt.xlabel('Position level')

plt.ylabel('Salary')

plt.show()



**Random Forest Classifier**

# Importing the libraries

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

# Importing the dataset

dataset = pd.read\_csv('Social\_Network\_Ads.csv')

X = dataset.iloc[:, [2, 3]].values

y = dataset.iloc[:, 4].values

# Splitting the dataset into the Training set and Test set

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

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.25, random\_state = 0)

# Feature Scaling

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

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

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

# Fitting classifier to the Training set

from sklearn.ensemble import RandomForestClassifier

classifier = RandomForestClassifier(n\_estimators = 10, criterion = 'entropy', random\_state = 0 )

classifier.fit(X\_train,y\_train)

# Predicting the Test set results

y\_pred = classifier.predict(X\_test)

# Making the Confusion Matrix

from sklearn.metrics import confusion\_matrix

cm = confusion\_matrix(y\_test, y\_pred)

# Visualising the Training set results

from matplotlib.colors import ListedColormap

X\_set, y\_set = X\_train, y\_train

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

np.arange(start = X\_set[:, 1].min() - 1, stop = X\_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(y\_set)):

plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1],

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

plt.title('Random Forest Classifier (Training set)')

plt.xlabel('Age')

plt.ylabel('Estimated Salary')

plt.legend()

plt.show()

# Visualising the Test set results

from matplotlib.colors import ListedColormap

X\_set, y\_set = X\_test, y\_test

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

np.arange(start = X\_set[:, 1].min() - 1, stop = X\_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(y\_set)):

plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1],

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

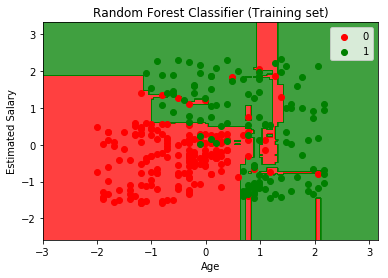
plt.title('Random Forest Classifier (Test set)')

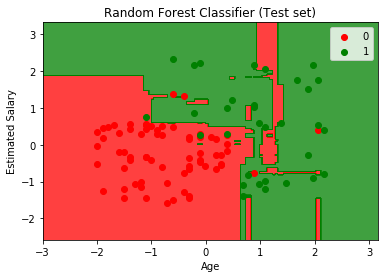
plt.xlabel('Age')

plt.ylabel('Estimated Salary')

plt.legend()

plt.show()





**ADABOOST**

import pandas as pd

import numpy as np

from sklearn.tree import DecisionTreeClassifier

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

from sklearn.datasets import make\_hastie\_10\_2

import matplotlib.pyplot as plt

""" HELPER FUNCTION: GET ERROR RATE ========================================="""

def get\_error\_rate(pred, Y):

return sum(pred != Y) / float(len(Y))

""" HELPER FUNCTION: PRINT ERROR RATE ======================================="""

def print\_error\_rate(err):

print ('Error rate: Training: %.4f - Test: %.4f' % err)

""" HELPER FUNCTION: GENERIC CLASSIFIER ====================================="""

def generic\_clf(Y\_train, X\_train, Y\_test, X\_test, clf):

clf.fit(X\_train,Y\_train)

pred\_train = clf.predict(X\_train)

pred\_test = clf.predict(X\_test)

return get\_error\_rate(pred\_train, Y\_train), \

get\_error\_rate(pred\_test, Y\_test)

""" ADABOOST IMPLEMENTATION ================================================="""

def adaboost\_clf(Y\_train, X\_train, Y\_test, X\_test, M, clf):

n\_train, n\_test = len(X\_train), len(X\_test)

# Initialize weights

w = np.ones(n\_train) / n\_train

pred\_train, pred\_test = [np.zeros(n\_train), np.zeros(n\_test)]

for i in range(M):

# Fit a classifier with the specific weights

clf.fit(X\_train, Y\_train, sample\_weight = w)

pred\_train\_i = clf.predict(X\_train)

pred\_test\_i = clf.predict(X\_test)

# Indicator function

miss = [int(x) for x in (pred\_train\_i != Y\_train)]

# Equivalent with 1/-1 to update weights

miss2 = [x if x==1 else -1 for x in miss]

# Error

err\_m = np.dot(w,miss) / sum(w)

# Alpha

alpha\_m = 0.5 \* np.log( (1 - err\_m) / float(err\_m))

# New weights

w = np.multiply(w, np.exp([float(x) \* alpha\_m for x in miss2]))

# Add to prediction

pred\_train = [sum(x) for x in zip(pred\_train,

[x \* alpha\_m for x in pred\_train\_i])]

pred\_test = [sum(x) for x in zip(pred\_test,

[x \* alpha\_m for x in pred\_test\_i])]

pred\_train, pred\_test = np.sign(pred\_train), np.sign(pred\_test)

# Return error rate in train and test set

return get\_error\_rate(pred\_train, Y\_train), \

get\_error\_rate(pred\_test, Y\_test)

""" PLOT FUNCTION ==========================================================="""

def plot\_error\_rate(er\_train, er\_test):

df\_error = pd.DataFrame([er\_train, er\_test]).T

df\_error.columns = ['Training', 'Test']

plot1 = df\_error.plot(linewidth = 3, figsize = (8,6),

color = ['lightblue', 'darkblue'], grid = True)

plot1.set\_xlabel('Number of iterations', fontsize = 12)

plot1.set\_xticklabels(range(0,450,50))

plot1.set\_ylabel('Error rate', fontsize = 12)

plot1.set\_title('Error rate vs number of iterations', fontsize = 16)

plt.axhline(y=er\_test[0], linewidth=1, color = 'red', ls = 'dashed')

""" MAIN SCRIPT ============================================================="""

if \_\_name\_\_ == '\_\_main\_\_':

# Read data

x, y = make\_hastie\_10\_2()

df = pd.DataFrame(x)

df['Y'] = y

# Split into training and test set

train, test = train\_test\_split(df, test\_size = 0.2)

X\_train, Y\_train = train.ix[:,:-1], train.ix[:,-1]

X\_test, Y\_test = test.ix[:,:-1], test.ix[:,-1]

# Fit a simple decision tree first

clf\_tree = DecisionTreeClassifier(max\_depth = 1, random\_state = 1)

er\_tree = generic\_clf(Y\_train, X\_train, Y\_test, X\_test, clf\_tree)

# Fit Adaboost classifier using a decision tree as base estimator

# Test with different number of iterations

er\_train, er\_test = [er\_tree[0]], [er\_tree[1]]

x\_range = range(10, 410, 10)

for i in x\_range:

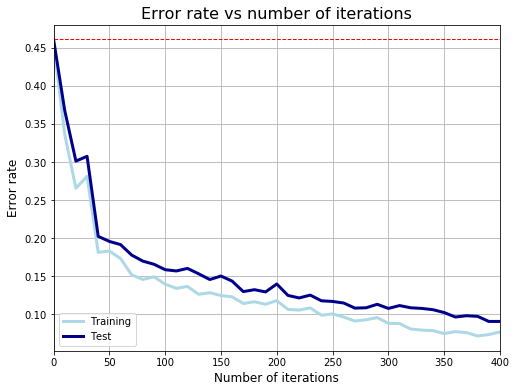
er\_i = adaboost\_clf(Y\_train, X\_train, Y\_test, X\_test, i, clf\_tree)

er\_train.append(er\_i[0])

er\_test.append(er\_i[1])

# Compare error rate vs number of iterations

plot\_error\_rate(er\_train, er\_test)



**CODE – 2**

# AdaBoost Classification

import pandas

from sklearn import model\_selection

from sklearn.ensemble import AdaBoostClassifier

url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv"

names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']

dataframe = pandas.read\_csv(url, names=names)

array = dataframe.values

X = array[:,0:8]

Y = array[:,8]

seed = 7

num\_trees = 30

kfold = model\_selection.KFold(n\_splits=10, random\_state=seed)

model = AdaBoostClassifier(n\_estimators=num\_trees, random\_state=seed)

results = model\_selection.cross\_val\_score(model, X, Y, cv=kfold)

print(results.mean())

**0.760457963089542**

**MLP**

**R CODE**

library(RSNNS)

data(iris)

iris

iris <- iris[sample(1:nrow(iris),length(1:nrow(iris))),1:ncol(iris)]

irisValues <- iris[,1:4]

irisTargets <- decodeClassLabels(iris[,5])

iris <- splitForTrainingAndTest(irisValues, irisTargets, ratio=0.15)

iris <- normTrainingAndTestSet(iris)

model <- mlp(iris$inputsTrain, iris$targetsTrain, size=5, learnFuncParams=c(0.1),maxit=50, inputsTest=iris$inputsTest, targetsTest=iris$targetsTest)

summary(model)

model

weightMatrix(model)

extractNetInfo(model)

par(mfrow=c(2,2))

plotIterativeError(model)

predictions <- predict(model,iris$inputsTest)

plotRegressionError(predictions[,2], iris$targetsTest[,2])

confusionMatrix(iris$targetsTrain,fitted.values(model))

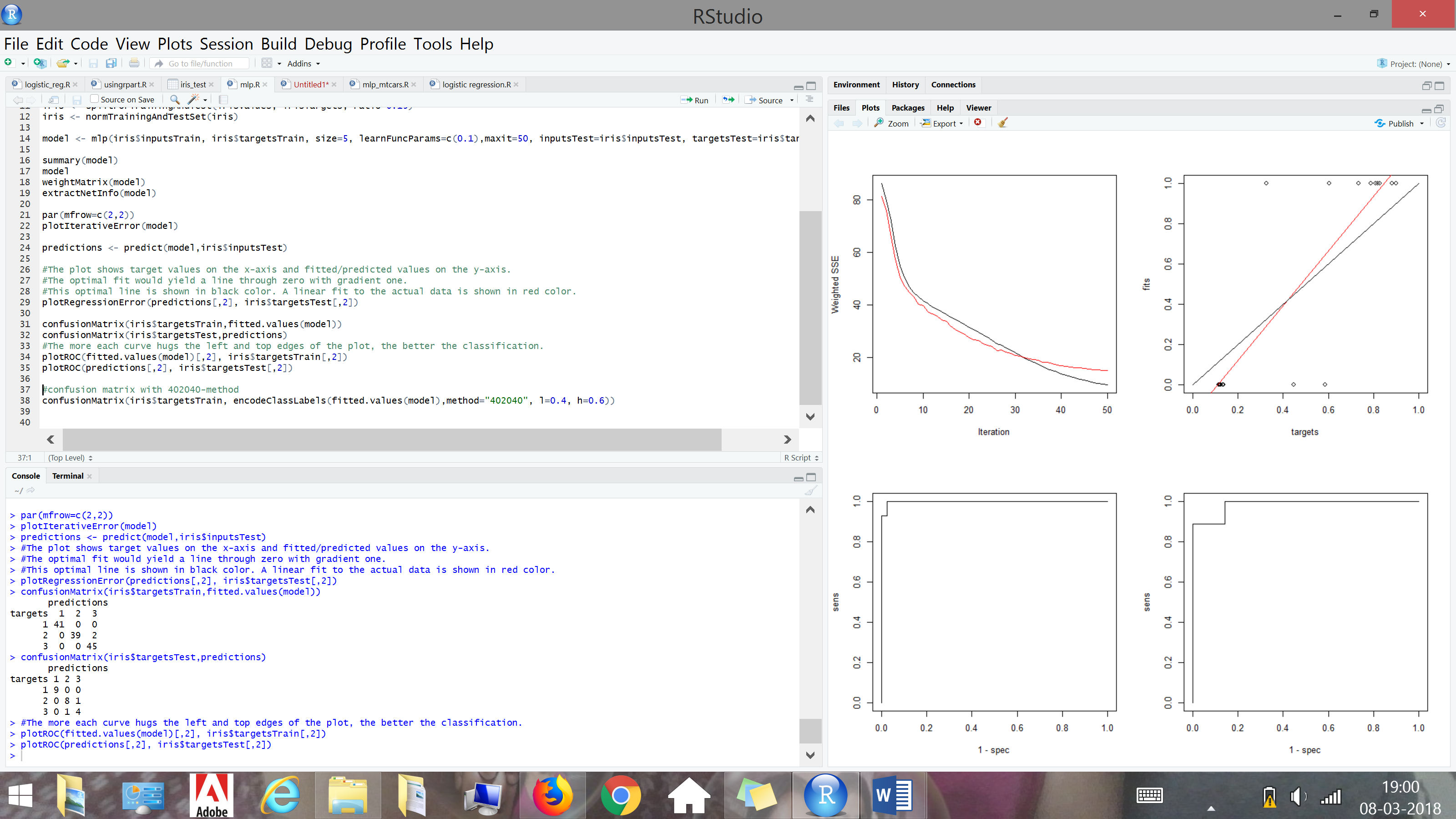
confusionMatrix(iris$targetsTest,predictions)

#The more each curve hugs the left and top edges of the plot, the better the classification.

plotROC(fitted.values(model)[,2], iris$targetsTrain[,2])

plotROC(predictions[,2], iris$targetsTest[,2])

confusionMatrix(iris$targetsTrain, encodeClassLabels(fitted.values(model),method="402040", l=0.4, h=0.6))



**Python**

from random import seed

from random import randrange

from csv import reader

# Load a CSV file

def load\_csv(filename):

dataset = list()

with open(filename, 'r') as file:

csv\_reader = reader(file)

for row in csv\_reader:

if not row:

continue

dataset.append(row)

return dataset

# Convert string column to float

def str\_column\_to\_float(dataset, column):

for row in dataset:

row[column] = float(row[column].strip())

# Convert string column to integer

def str\_column\_to\_int(dataset, column):

class\_values = [row[column] for row in dataset]

unique = set(class\_values)

lookup = dict()

for i, value in enumerate(unique):

lookup[value] = i

for row in dataset:

row[column] = lookup[row[column]]

return lookup

# Split a dataset into k folds

def cross\_validation\_split(dataset, n\_folds):

dataset\_split = list()

dataset\_copy = list(dataset)

fold\_size = int(len(dataset) / n\_folds)

for i in range(n\_folds):

fold = list()

while len(fold) < fold\_size:

index = randrange(len(dataset\_copy))

fold.append(dataset\_copy.pop(index))

dataset\_split.append(fold)

return dataset\_split

# Calculate accuracy percentage

def accuracy\_metric(actual, predicted):

correct = 0

for i in range(len(actual)):

if actual[i] == predicted[i]:

correct += 1

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

# Evaluate an algorithm using a cross validation split

def evaluate\_algorithm(dataset, algorithm, n\_folds, \*args):

folds = cross\_validation\_split(dataset, n\_folds)

scores = list()

for fold in folds:

train\_set = list(folds)

train\_set.remove(fold)

train\_set = sum(train\_set, [])

test\_set = list()

for row in fold:

row\_copy = list(row)

test\_set.append(row\_copy)

row\_copy[-1] = None

predicted = algorithm(train\_set, test\_set, \*args)

actual = [row[-1] for row in fold]

accuracy = accuracy\_metric(actual, predicted)

scores.append(accuracy)

return scores

# Make a prediction with weights

def predict(row, weights):

activation = weights[0]

for i in range(len(row)-1):

activation += weights[i + 1] \* row[i]

return 1.0 if activation >= 0.0 else 0.0

# Estimate Perceptron weights using stochastic gradient descent

def train\_weights(train, l\_rate, n\_epoch):

weights = [0.0 for i in range(len(train[0]))]

for epoch in range(n\_epoch):

for row in train:

prediction = predict(row, weights)

error = row[-1] - prediction

weights[0] = weights[0] + l\_rate \* error

for i in range(len(row)-1):

weights[i + 1] = weights[i + 1] + l\_rate \* error \* row[i]

return weights

# Perceptron Algorithm With Stochastic Gradient Descent

def perceptron(train, test, l\_rate, n\_epoch):

predictions = list()

weights = train\_weights(train, l\_rate, n\_epoch)

for row in test:

prediction = predict(row, weights)

predictions.append(prediction)

return(predictions)

# Test the Perceptron algorithm on the sonar dataset

seed(1)

# load and prepare data

filename = 'C:\Users\HarshitSharma\Desktop\sonar.csv'

dataset = load\_csv(filename)

for i in range(len(dataset[0])-1):

str\_column\_to\_float(dataset, i)

# convert string class to integers

str\_column\_to\_int(dataset, len(dataset[0])-1)

# evaluate algorithm

n\_folds = 3

l\_rate = 0.01

n\_epoch = 500

scores = evaluate\_algorithm(dataset, perceptron, n\_folds, l\_rate, n\_epoch)

print('Scores: %s' % scores)

print('Mean Accuracy: %.3f%%' % (sum(scores)/float(len(scores))))

**Output:**

Scores: [73.91304347826086, 78.26086956521739, 68.11594202898551]

Mean Accuracy: 73.430%

**Error Correcting for Multi Class Learning Problem**

import pandas as pd

import numpy as np

from sklearn.metrics import accuracy\_score

from sklearn.model\_selection import train\_test\_split

from sklearn.multiclass import OutputCodeClassifier

from sklearn import tree

from sklearn.datasets import load\_iris

iris= load\_iris()

X=iris.data

y=iris.target

Xtrain,Xtest,ytrain,ytest=train\_test\_split(X,y)

OC=OutputCodeClassifier(tree.DecisionTreeClassifier(),code\_size=2,random\_state=0)

OC.fit(Xtrain,ytrain)

res= OC.predict(Xtest)

print(res)

acc=accuracy\_score(res,ytest)

print(acc)

**OUTPUT:**

[2 2 0 0 0 1 2 2 0 2 1 2 0 1 0 2 1 2 1 1 2 1 1 0 0 0 2 0 1 0 0 1 0 1 1 0 2 2]

**ACCURACY:**

0.9473684210526315

**GAUSSIAN MIXTURE MODEL USING EXPECTATION MAXIMIZATION**

import numpy as np

class GMM:

def \_\_init\_\_(self, k = 3, eps = 0.0001):

self.k = k ## number of clusters

self.eps = eps ## threshold to stop `epsilon`

# All parameters from fitting/learning are kept in a named tuple

from collections import namedtuple

def fit\_EM(self, X, max\_iters = 1000):

# n = number of data-points, d = dimension of data points

n, d = X.shape

# randomly choose the starting centroids/means

## as 3 of the points from datasets

mu = X[np.random.choice(n, self.k, False), :]

# initialize the covariance matrices for each gaussians

Sigma= [np.eye(d)] \* self.k

# initialize the probabilities/weights for each gaussians

w = [1./self.k] \* self.k

# responsibility matrix is initialized to all zeros

# we have responsibility for each of n points for eack of k gaussians

R = np.zeros((n, self.k))

### log\_likelihoods

log\_likelihoods = []

P = lambda mu, s: np.linalg.det(s) \*\* -.5 \*\* (2 \* np.pi) \*\* (-X.shape[1]/2.) \

\* np.exp(-.5 \* np.einsum('ij, ij -> i',\

X - mu, np.dot(np.linalg.inv(s) , (X - mu).T).T ) )

# Iterate till max\_iters iterations

while len(log\_likelihoods) < max\_iters:

# E - Step

## Vectorized implementation of e-step equation to calculate the

## membership for each of k -gaussians

for k in range(self.k):

R[:, k] = w[k] \* P(mu[k], Sigma[k])

### Likelihood computation

log\_likelihood = np.sum(np.log(np.sum(R, axis = 1)))

log\_likelihoods.append(log\_likelihood)

## Normalize so that the responsibility matrix is row stochastic

R = (R.T / np.sum(R, axis = 1)).T

## The number of datapoints belonging to each gaussian

N\_ks = np.sum(R, axis = 0)

# M Step

## calculate the new mean and covariance for each gaussian by

## utilizing the new responsibilities

for k in range(self.k):

## means

mu[k] = 1. / N\_ks[k] \* np.sum(R[:, k] \* X.T, axis = 1).T

x\_mu = np.matrix(X - mu[k])

## covariances

Sigma[k] = np.array(1 / N\_ks[k] \* np.dot(np.multiply(x\_mu.T, R[:, k]), x\_mu))

## and finally the probabilities

w[k] = 1. / n \* N\_ks[k]

# check for onvergence

if len(log\_likelihoods) < 2 : continue

if np.abs(log\_likelihood - log\_likelihoods[-2]) < self.eps: break

## bind all results together

from collections import namedtuple

self.params = namedtuple('params', ['mu', 'Sigma', 'w', 'log\_likelihoods', 'num\_iters'])

self.params.mu = mu

self.params.Sigma = Sigma

self.params.w = w

self.params.log\_likelihoods = log\_likelihoods

self.params.num\_iters = len(log\_likelihoods)

return self.params

def plot\_log\_likelihood(self):

import pylab as plt

plt.plot(self.params.log\_likelihoods)

plt.title('Log Likelihood vs iteration plot')

plt.xlabel('Iterations')

plt.ylabel('log likelihood')

plt.show()

def predict(self, x):

p = lambda mu, s : np.linalg.det(s) \*\* - 0.5 \* (2 \* np.pi) \*\*\

(-len(x)/2) \* np.exp( -0.5 \* np.dot(x - mu , \

np.dot(np.linalg.inv(s) , x - mu)))

probs = np.array([w \* p(mu, s) for mu, s, w in \

zip(self.params.mu, self.params.Sigma, self.params.w)])

return probs/np.sum(probs)

def demo\_2d():

# Load data

#X = np.genfromtxt('data1.csv', delimiter=',')

### generate the random data

np.random.seed(3)

m1, cov1 = [9, 8], [[.5, 1], [.25, 1]] ## first gaussian

data1 = np.random.multivariate\_normal(m1, cov1, 90)

m2, cov2 = [6, 13], [[.5, -.5], [-.5, .1]] ## second gaussian

data2 = np.random.multivariate\_normal(m2, cov2, 45)

m3, cov3 = [4, 7], [[0.25, 0.5], [-0.1, 0.5]] ## third gaussian

data3 = np.random.multivariate\_normal(m3, cov3, 65)

X = np.vstack((data1,np.vstack((data2,data3))))

np.random.shuffle(X)

# np.savetxt('sample.csv', X, fmt = "%.4f", delimiter = ",")

####

gmm = GMM(3, 0.000001)

params = gmm.fit\_EM(X, max\_iters= 100)

print (params.log\_likelihoods)

import pylab as plt

from matplotlib.patches import Ellipse

def plot\_ellipse(pos, cov, nstd=2, ax=None, \*\*kwargs):

def eigsorted(cov):

vals, vecs = np.linalg.eigh(cov)

order = vals.argsort()[::-1]

return vals[order], vecs[:,order]

if ax is None:

ax = plt.gca()

vals, vecs = eigsorted(cov)

theta = np.degrees(np.arctan2(\*vecs[:,0][::-1]))

# Width and height are "full" widths, not radius

width, height = 2 \* nstd \* np.sqrt(abs(vals))

ellip = Ellipse(xy=pos, width=width, height=height, angle=theta, \*\*kwargs)

ax.add\_artist(ellip)

return ellip

def show(X, mu, cov):

plt.cla()

K = len(mu) # number of clusters

colors = ['b', 'k', 'g', 'c', 'm', 'y', 'r']

plt.plot(X.T[0], X.T[1], 'm\*')

for k in range(K):

plot\_ellipse(mu[k], cov[k], alpha=0.6, color = colors[k % len(colors)])

fig = plt.figure(figsize = (13, 6))

fig.add\_subplot(121)

show(X, params.mu, params.Sigma)

fig.add\_subplot(122)

plt.plot(np.array(params.log\_likelihoods))

plt.title('Log Likelihood vs iteration plot')

plt.xlabel('Iterations')

plt.ylabel('log likelihood')

plt.show()

print (gmm.predict(np.array([1, 2])))

if \_\_name\_\_ == "\_\_main\_\_":

demo\_2d()

from optparse import OptionParser

parser = OptionParser()

parser.add\_option("-f", "--file", dest="filepath", help="File path for data")

parser.add\_option("-k", "--clusters", dest="clusters", help="No. of gaussians")

parser.add\_option("-e", "--eps", dest="epsilon", help="Epsilon to stop")

parser.add\_option("-m", "--maxiters", dest="max\_iters", help="Maximum no. of iteration")

options, args = parser.parse\_args()

if not options.filepath : raise('File not provided')

if not options.clusters :

print("Used default number of clusters = 3" )

k = 3

else: k = int(options.clusters)

if not options.epsilon :

print("Used default eps = 0.0001" )

eps = 0.0001

else: eps = float(options.epsilon)

if not options.max\_iters :

print("Used default maxiters = 1000" )

max\_iters = 1000

else: eps = int(options.maxiters)

X = np.genfromtxt(options.filepath, delimiter=',')

gmm = GMM(k, eps)

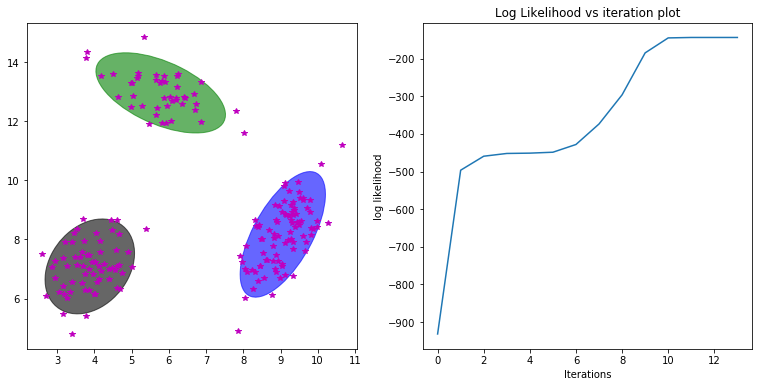
params = gmm.fit\_EM(X, max\_iters)

print (params.log\_likelihoods)

gmm.plot\_log\_likelihood()

print (gmm.predict(np.array([1, 2])))

**OUTPUT:**



[2.20783559e-036 1.00000000e+000 1.55285737e-109]