**Exp 1 --- Find-S algorithm**

import pandas as pd

import numpy as np

a = pd.read\_csv("enjoysport.csv")

print(a)

print("\n The Total Number of Training Instances are: ", len(a))

num\_attribute = len(a[0])-1

print("\n The initial hypothesis is: ")

hypothesis = ['0']\*num\_attribute

print(hypothesis)

for i in range(1, len(a)):

    if a[i][num\_attribute] != 'No':

        print('\n Vector {} instance is: '.format(i), a[i])

        for j in range(0, num\_attribute):

            if  hypothesis[j] == '0' or hypothesis[j] == a[i][j]:

                hypothesis[j] = a[i][j]

            else:

                hypothesis[j] = '?'

    print("\n The hypothesis for the training instance {} is : \n".format(i+1), hypothesis)

print("\n The Maximally specific hypothesis for the training instance is ")

print(hypothesis)

**Exp 2 --- Candidate Elimination Method**

import numpy as np

import pandas as pd

data = pd.DataFrame(data  = pd.read\_csv("enjoysport.csv"))

concepts = np.array(data.iloc[:,  0:-1])

print("Concepts are: \n", concepts)

target = np.array(data.iloc[:,-1])

print("Targets are\n", target)

def learn(concepts, target):

    specific\_h = concepts[0].copy()

    print("Initialization of specific\_h and general\_h")

    print(specific\_h)

    general\_h = [['?' for i in range(len(specific\_h))] for i in range(len(specific\_h))]

    print(general\_h)

    for i, h in enumerate(concepts):

        if target[i] == 'Yes':

            for x in range(len(specific\_h)):

                if h[x] != specific\_h[x]:

                    specific\_h[x] = '?'

                    general\_h[x][x] = '?'

        print(specific\_h)

        if target[i] == 'No':

            for x in range(len(specific\_h)):

                if h[x] != specific\_h[x]:

                    general\_h[x][x] = specific\_h[x]

                else:

                    general\_h[x][x] = '?'

        print("Steps of candidate elimination Algorithm: ", (i+1))

        print("values of Specific\_h\n", specific\_h)

    indices = [i for i, val in enumerate(general\_h) if val == ['?','?','?','?','?','?']]

    for i in  indices:

        general\_h.remove(['?','?','?','?','?','?'])

    return specific\_h, general\_h

s\_final, g\_final = learn(concepts, target)

print("Final Specific\_h: ", s\_final, sep = '\n')

print("Final General\_h: ", g\_final, sep = '\n')

**Exp 4 --- Linear Regression and Logistic Regression**

# Linear Regression

import numpy as np

import matplotlib.pyplot as plt

x = 2 \* np.random.rand(100,1)

y = 4 + 3 \* x+ np.random.rand(100,1)

from sklearn.linear\_model import LinearRegression

lr = LinearRegression()

lr.fit (x,y)

print ('Intercept value:',lr.intercept\_)

print ('Coefficient value:', lr.coef )

y\_pred = lr.predict (x)

plt.figure(figsize=(10,5))

plt.xlabel('x')

plt.ylabel('y')

plt.scatter (x,y)

plt.plot (x,y\_pred, color='red')

# Logistic Regression

from sklearn import datasets

import numpy as np

import matplotlib.pyplot as plt

iris = datasets.load\_iris()

print (iris.keys())

print (iris["feature\_names"])

iris

iris['target']

x = iris["data"][:,31]

y = (iris['target'] == 2).astype(np.int)

x.shape, y.shape

from sklearn.linear\_model import LogisticRegression

log\_reg = LogisticRegression ()

log\_reg.fit (x,y)

print ('Logistic Regression Accuracy Score:', log\_reg.score (x,y))

y\_pred = log\_reg.predict (x)

y\_pred

from sklearn.metrics import confusion\_matrix, accuracy\_score

print (confusion\_matrix (y,y\_pred))

accuracy\_score(y,y\_pred)

x\_new = np.linspace (0,3, 1000).reshape (-1, 1)

y\_proba = log\_reg.predict\_proba(x\_new)

plt.plot (x\_new, y\_proba[:,1],"g", label = "Iris-Virginica")

plt.plot (x\_new, y\_proba[:,0],"b--", label = "Not Iris-Virginica")

plt.xlabel ("Petal width (cm)")

plt.ylabel ("Probability")

plt.legend()

**Exp 5 --- Program for Bias, Variance, Remove duplicates, Cross Validation**

# 5.) Bias, Variance

from mlxtend.evaluate import bias\_variance\_decomp

import numpy as np

import pandas as pd

from sklearn.linear\_model import LinearRegression

from sklearn.utils import shuffle

from sklearn.metrics import mean\_squared\_error

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

data = pd.read\_csv("5student.csv")

data = data[["G1", "G2", "G3", "studytime", "failures", "absences"]]

data.head()

predict = "G3"

x = np.array(data.drop([predict], 1))

y = np.array(data[predict])

xtrain, xtest, ytrain, ytest = train\_test\_split(x, y, test\_size=0.2)

print (xtrain.shape, ytrain.shape, xtest.shape, ytest.shape)

lr = LinearRegression()

lr.fit(xtrain, ytrain)

y\_pred = lr.predict(xtest)

mse, bias, variance = bias\_variance\_decomp (lr, xtrain, ytrain, xtest, ytest, loss='mse', num\_rounds=150)

print ("MSE : %.3f"%mse)

print ("Average Bias : %.3f" %bias)

print ("Average Variance : %.3f" %variance)

# 5.) Remove duplicates

import pandas as pd

data ={'F1': [1, 1, 1, 2], 'F2': [2, 2, 2, 3], 'F3': [3, 3, 4, 5]}

df = pd.DataFrame(data)

print ('Source DataFrame:\n', df)

result\_df = df.drop\_duplicates()

print ('Result DataFrame:\n', result\_df)

result\_df = df.drop\_duplicates(keep=False)

print ('Result DataFrame:\n', result\_df)

result\_df = df.drop\_duplicates(subset=['F1', 'F2'])

print ('Result DataFrame:\n', result\_df)

result\_df = df.drop\_duplicates(subset=['F1', 'F2'], keep='last')

print ('Result DataFrame:\n', result\_df)

# 5.) Cross Validation

from sklearn import datasets

from sklearn.tree import DecisionTreeClassifier

from sklearn.model\_selection import KFold, cross\_val\_score

x, y = datasets.load\_iris(return\_X\_y=True)

print ("Size of independent and dependent features: ", x.shape, y.shape)

dtclf = DecisionTreeClassifier(random\_state=30)

cv\_feature = KFold(n\_splits = 4)

cv\_scores = cross\_val\_score(dtclf, x, y, cv = cv\_feature)

print ("Cross Validation Scores: ", cv\_scores)

print ("Average CV Score: ", cv\_scores.mean())

print ("Number of CV Scores used in Average: ", len(cv\_scores))

**Exp 6 --- Categorical Encoding, One-hot Encoding**

#Label encoding - Using Category codes approach

import pandas as pd

import numpy as np

bridge\_types = ('Arch', 'Beam', 'Truss', 'Cantilever', 'Tied Arch', 'Suspension', 'Cable')

bridge\_df = pd.DataFrame(bridge\_types, columns = ['Bridge\_Types'])

bridge\_df['Bridge\_Types'] = bridge\_df['Bridge\_Types'].astype('category')

bridge\_df['Bridge\_Types\_Car'] = bridge\_df['Bridge\_Types'].cat.codes

bridge\_df

#Label encoding - Using Sci-kit learn library approach

import pandas as pd

import numpy as np

from sklearn.preprocessing import LabelEncoder

bridge\_types = ('Arch', 'Beam', 'Truss', 'Cantilever', 'Tied Arch', 'Suspension', 'Cable')

bridge\_df = pd.DataFrame(bridge\_types, columns = ['Bridge\_Types'])

labelencoder = LabelEncoder()

bridge\_df['Bridge\_Types\_Cat'] = labelencoder.fit\_transform(bridge\_df['Bridge\_Types'])

bridge\_df

#One-Hot encoding - Using Sci-kit learn library approach

import pandas as pd

import numpy as np

from sklearn.preprocessing import OneHotEncoder

bridge\_types = ('Arch', 'Beam', 'Truss', 'Cantilever', 'Tied Arch', 'Suspension', 'Cable')

bridge\_df = pd.DataFrame(bridge\_types, columns=['Bridge\_Types'])

enc = OneHotEncoder(handle\_unknown='ignore')

enc\_df = pd.DataFrame(enc.fit\_transform(bridge\_df['Bridge\_Types\_Cat']).toarray())

bridge\_df = bridge\_df.join(enc\_df)

bridge\_df

#One-Hot Encoding - Using Dummies values approach

import pandas as pd

import numpy as np

bridge\_types = ('Arch', 'Beam', 'Truss', 'Cantilever', 'Tied Arch', 'Suspension', 'Cable')

bridge\_df = pd.DataFrame(bridge\_types, columns=['Bridge\_Types'])

bridge\_df

dum\_df = pd.get\_dummies(bridge\_df, columns=['Bridge\_Types'], prefix=["Type\_is"])

dum\_df

bridge\_df = bridge\_df.join(dum\_df)

bridge\_df

**Exp 7**

# Back Propagation Algorithm

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

print(x)

print(y)

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

lr = 0.1

inputlayer\_neurons = 2

hiddenlayer\_neurons = 3

output\_neurons = 1

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.unifrom(size = (1, output\_neurons))

for i in range(epoch):

    hinp1 = np.dot(x, wh)

    hinp = hinp1 + bh

    hlayer\_act = sigmoid(hinp)

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

    outinp = outinp1 + bout

    output = sigmoid(outinp)

    eo = y-output

    outgrad = derivatives\_sigmoid(output)

    d\_output = eo \* outgrad

    eh = d\_output.dot(wout.T)

    hiddengrad = derivatives\_sigmoid(hlayer\_act)

    d\_hiddenlayer = eh\*hiddengrad

    wout += hlayer\_act.T.dot(d\_output) \* lr

    wh += x.T.dot(d\_hiddenlayer) \* lr

print("Input: \n"+str(x))

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

print("Predicted Output:\n", output)

**Exp 8 --- KNN Algorithm**

import numpy as np

import pandas as pd

from sklearn.neighbors import KNeighborsClassifier

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

from sklearn import metrics

from sklearn import datasets

# Load Iris dataset

iris = datasets.load\_iris()

#The following variables x contains the four features and y contains the class labels

x =iris.data

y = iris.target

print('sepal-length', 'sepal-width', 'petal-length', 'petal-width')

print (x)

print ('class: 0-Iris-Setosa, 1-Iris-Versicolour, 2-Iris-Virginica')

print (y)

# Split the dataset into train and test data in 70:30 ratio.

# The train data contains 105 samples and test data contains 45 samples

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.3)

# Train the model using KNN classifier with neighbours =5

knnclf = KNeighborsClassifier (n\_neighbors = 5)

knnclf.fit (x\_train,y\_train)

# Make predictions for test data

y\_pred = knnclf.predict (x\_test)

print('--------------------------------------------')

print('\nConfusion Matrix for Test data: \n', metrics.confusion\_matrix(y\_test, y\_pred))

print('--------------------------------------------')

print('\nKNN Classification Report: \n', metrics.classification\_report(y\_test, y\_pred))

print('--------------------------------------------')

print('Test data Accuracy of the KNN Classifiers is %0.2f'%metrics.accuracy\_score(y\_test, y\_pred))

print('--------------------------------------------')

# Print both correct and wrong predictions

i=0

print('--------------------------------------------')

print('%0-25s %-25s %-235s' % ('Original Label, Predicted Label', 'Correct/Wrong'))

for label in y\_test:

    print ('%-258 %-25s' % (label, y\_pred[i]), end="")

    if label == y\_pred[i]:

        print(' %0-25s' % ("Correct"))

    else:

        print(' %-25s' % ("Wrong"))

    i=i+1

**Exp 10 --- Naïve Bayes**

import pandas as pd

msg = pd.read\_csv('filename.csv', names=['message', 'label'])

print('The dimension of the dataset', msg.shape)

msg['labelnum'] = msg.label.map({'pos':1, 'neg': 0})

X = msg.message

y = msg.labelnum

print(X)

print(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('\n The total number of Training Data: ', ytrain.shape)

print('\n The total number of Test Data: ', ytest.shape)

# Output of count vectorizer is a sparse matrix

from sklearn.feature\_extraction.text import CountVectorizer

count\_vect = CountVectorizer()

xtrain\_dtm = count\_vect.fit\_transform(xtrain)

xtest\_dtm = count\_vect.transform(xtest)

print('\n The words or Tokens in the text documents \n')

print(count\_vect.get\_feature\_names())

xtrain

xtrain\_dtm

df = pd.DataFrame(xtrain\_dtm.toarray(), columns=count\_vect.get\_feature\_names())

df.head(12)

# Training Naive Bayes Classifier on training data.

from sklearn.naive\_bayes import MultinomialNB

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

train\_predicted = clf.predict(xtrain\_dtm)

test\_predicted = clf.predict(xtest\_dtm)

# Printing accuracy, Confusion matrix, Precision and Recall

from sklearn import metrics

print('\n Accuracy of the classifier is for Train data: ', metrics.accuracy\_score(ytrain, train\_predicted))

print('\nAccuracy of the classifier is for Test data: ', metrics.accuracy\_score(ytest, test\_predicted))

print('\n Confusion matrix for Train Data')

print(metrics.confusion\_matrix(ytrain, train\_predicted))

print('\nThe value of Precision', metrics.precision\_score(ytrain, train\_predicted))

print('\nThe value of Recall', metrics.recall\_score(ytrain, train\_predicted))

print('\n Confusion matrix for Test data')

print(metrics.confusion\_matrix(ytest, test\_predicted))

print('\n The value of Precision', metrics.precision\_score(ytest, test\_predicted))

print('\n The value of Recall', metrics.recall\_score(ytest, test\_predicted))

**Exp 11 --- KMeans Algorithm**

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

from sklearn.cluster import KMeans

data = pd.read\_csv("file.csv")

df1 = pd.DataFrame(data)

print(df1)

f1 = df1['Distance\_Feature'].values

f2 = df1['Speeding\_Feature'].values

X = np.matrix(list(zip(f1, f2)))

plt.plot()

plt.xlim([0, 100])

plt.ylim([0, 50])

plt.title('Driver Dataset')

plt.ylabel('Speeding\_feature')

plt.xlabel('Distance\_Feature')

plt.scatter(f1, f2)

plt.show()

# KMeans Algorithm

# K = 3

kmeans\_model = KMeans(n\_clusters=3).fit(X)

kmeans\_model.cluster\_centers\_

kmeans\_model.transform(X)

colors = ['b', 'g', 'r']

markers = ['o', 'v', 's']

plt.xlim([0, 100])

plt.ylim([0, 50])

plt.title("K Means Clustering with K = 3")

for i, l in enumerate(kmeans\_model.labels\_):

    plt.plot(f1[i], f2[i], color = colors[l], marker = markers[l], ls = 'None')

centroids = kmeans\_model.cluster\_centers\_

plt.scatter(centroids[:,0], centroids[:,1], markers='x', s=100, linewidths=3, color = 'k', zorder = 10)

plt.show()