Machine learning:

Machine learning is a subset of artificial intelligence in the field of computer science that often uses statistical techniques to give computers the ability to "learn" (i.e., progressively improve performance on a specific task) with data, without being explicitly programmed. In the past decade, machine learning has given us self-driving cars, practical speech recognition, effective web search, and a vastly improved understanding of the human genome.

Machine learning tasks:

Machine learning tasks are typically classified into two broad categories, depending on whether there is a learning "signal" or "feedback" available to a learning system: 1. Supervised learning: The computer is presented with example inputs and their desired outputs, given by a "teacher", and the goal is to learn a general rule that maps inputs to outputs. As special cases, the input signal can be only partially available, or restricted to special feedback: 2. Semi-supervised learning: the computer is given only an incomplete training signal: a training set with some (often many) of the target outputs missing.

Active learning:

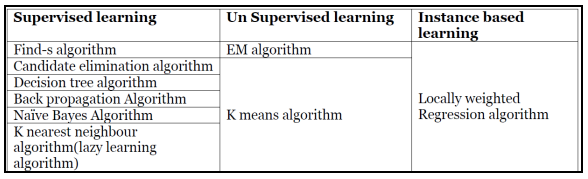
The computer can only obtain training labels for a limited set of instances (based on a budget), and also has to optimize its choice of objects to acquire labels for. When used interactively, these can be presented to the user for labelling.

Reinforcement learning:

Training data (in form of rewards and punishments) is given only as feedback to the program's actions in a dynamic environment, such as driving a vehicle or playing a game against an opponent.

Unsupervised learning:

No labels are given to the learning algorithm, leaving it on its own to find structure in its input. Unsupervised learning can be a goal in itself (discovering hidden patterns in data) or a means towards an end (feature learning).



Machine Learning Applications:

In classification, inputs are divided into two or more classes, and the learner must produce a model that assigns unseen inputs to one or more (multi-label classification) of these classes. This is typically tackled in a supervised manner. Spam filtering is an example of classification, where the inputs are email (or other) messages and the classes are "spam" and "not spam". In regression, also a supervised problem, the outputs are continuous rather than discrete. In clustering, a set of inputs is to be divided into groups. Unlike in classification, the groups are not known beforehand, making this typically an unsupervised task. Density estimation finds the distribution of inputs in some space. Dimensionality reduction simplifies inputs by mapping them into a lower dimensional space. Topic modeling is a related problem, where a program is given a list of human language documents and is tasked with finding out which documents cover similar topics.

Machine learning Approaches

1.Decision tree learning:

Decision tree learning uses a decision tree as a predictive model, which maps observations about an item to conclusions about the item's target value.

2. Association rule learning Association rule learning is a method for discovering interesting relations between variables in large databases.

3. Artificial neural networks An artificial neural network (ANN) learning algorithm, usually called "neural network" (NN), is a learning algorithm that is vaguely inspired by biological neural networks. Computations are structured in terms of an interconnected group of artificial neurons, processing information using a connectionist approach to computation. Modern neural networks are non-linear statistical data modeling tools. They are usually used to model complex relationships between inputs and outputs, to find patterns in data, or to capture the statistical structure in an unknown joint probability distribution between observed variables.

4. Deep learning Falling hardware prices and the development of GPUs for personal use in the last few years have contributed to the development of the concept of deep learning which consists of multiple hidden layers in an artificial neural network. This approach tries to model the way the human brain processes light and sound into vision and hearing. Some successful applications of deep learning are computer vision and speech Recognition.

5. Inductive logic programming :

Inductive logic programming (ILP) is an approach to rule learning using logic Programming as a uniform representation for input examples, background knowledge, and hypotheses. Given an encoding of the known background knowledge and a set of examples represented as a logical database of facts, an ILP system will derive a hypothesized logic program that entails all positive and no negative examples. Inductive programming is a related field that considers any kind of programming languages for representing hypotheses (and not only logic programming), such as functional programs.

6. Clustering:

Cluster analysis is the assignment of a set of observations into subsets (called clusters) so that observations within the same cluster are similar according to some pre designated criterion or criteria, while observations drawn from different clusters are dissimilar. Different clustering techniques make different assumptions on the structure of the data, often defined by some similarity metric and evaluated for example by internal compactness (similarity between members of the same cluster) and separation between different clusters. Other methods are based on estimated density and graph connectivity. Clustering is a method of unsupervised learning, and a common technique for statistical data analysis. 8. Bayesian networks:

A Bayesian network, belief network or directed acyclic graphical model is a probabilistic graphical model that represents a set of random variables and their conditional independencies via a directed acyclic graph (DAG). For example, a Bayesian network could represent the probabilistic relationships between diseases and symptoms. Given symptoms, the network can be used to compute the probabilities of the presence of various diseases. Efficient algorithms exist that perform inference and learning.

Reinforcement learning:

Reinforcement learning is concerned with how an agent ought to take actions in an environment so as to maximize some notion of long-term reward. Reinforcement learning algorithms attempt to find a policy that maps states of the world to the actions the agent ought to take in those states. Reinforcement learning differs from the supervised learning problem in that correct input/output pairs are never presented, nor sub-optimal actions explicitly corrected.

Genetic algorithms

A genetic algorithm (GA) is a search heuristic that mimics the process of natural selection, and uses methods such as mutation and crossover to generate new genotype in the hope of finding good solutions to a given problem. In machine learning, genetic algorithms found some uses in the 1980s and 1990s. Conversely, machine learning techniques have been used to improve the performance of genetic and evolutionary algorithms.

**EXP1:**

**a.The probability that it is Friday and that a student is absent is 3 %. Since there are 5 school days in a week, the probability that it is Friday is 20 %. What is the probability that a student is absent given that today is Friday? Apply Baye’s rule in python to get the result**

likelywood\_prob = float(input("Enter likelywood probability:"))

Prior\_prob = float(input("Enter prior probability:"))

posterior\_prob=0

posterior\_prob = ((likelywood\_prob) /(Prior\_prob)) \* 100

print(posterior\_prob)

**b.Dangerious fires are rare 1% but smoke is fairly common 10% is due to barbicues and 90% dangerious fire make somke then what the probability of dangerious fire when there is no somke.**

p\_fire = float(input("enter probability of the fire:"))

p\_smoke\_fire =float(input("Enter probability of smoke when given condition is fire:"))

p\_smoke = float(input("Enter probability of smoke:"))

p\_danger\_fire\_smoke= ((p\_fire\*p\_smoke\_fire)/p\_smoke)\*100

print('Rsult-',p\_danger\_fire\_smoke)

**C. oh no 50% of all rainy day start with cloudy, but cloudy mornings are common 40% days starts with cloudy and this is only dry month so 10% of the days are rainy then what is the chance of rain given cloudy.**

p\_rain = float(input("enter probability of the rain:"))

p\_rain\_cloud =float(input("Enter probability of cloud when given condition is rain:"))

p\_cloud = float(input("Enter probability of cloud:"))

p\_rain\_cloud= ((p\_rain\*p\_rain\_cloud)/p\_cloud)\*100

print('Rsult-',p\_rain\_cloud)

**EXP2:**

1. **Extract the data from database using python**

import mysql.connector

import sys

import csv

# Connect to server

cnx = mysql.connector.connect(

host="localhost",

#port=3306,

user="root",

password="1061",database="mastan")

# Get a cursor

mycursor = cnx.cursor()

#cur = cnx.cursor()

# Execute a query

mycursor.execute("SELECT \* from emp;")

myresult = mycursor.fetchall()

c = csv.writer(open("hot.csv","w"))

c.writerow(myresult)

print(myresult)

1. **Extracting data from CSV file in local machine**

import pandas as pd

import numpy as np

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

print(dataset)

dataset.head()

1. **Extracting data from txt file**

dataset\_txt = pd.read\_csv('Restaurant\_Reviews.tsv', delimiter = '\t', quoting = 3)

print(dataset\_txt)

1. **Extracting data from Excel**

df = pd.read\_excel ('marks.xls')

print (df)

df.head()

1. **importing data from remote location from sklearn data sets**

import pandas as pd

import numpy as np

from sklearn import datasets

iris = datasets.load\_iris()

print(iris)

1. **importing data from remote location**

import requests

import pandas as pd

import numpy as np

download\_url = "https://raw.githubusercontent.com/fivethirtyeight/data/master/nba-elo/nbaallelo.csv"

target\_csv\_path = "nba\_all\_elo.csv"

print(target\_csv\_path)

response = requests.get(download\_url)

print(response)

response.raise\_for\_status() # Check that the request was successful

with open(target\_csv\_path, "wb") as f:

f.write(response.content)

print("Download ready.")

nba = pd.read\_csv("nba\_all\_elo.csv")

type(nba)

1. **Reading image data from local system**

import cv2

img = cv2.imread('photo.jpg')

cv2.imshow('image', img)

# Maintain output window utill

# user presses a key

cv2.waitKey(0)

# Destroying present windows on screen

cv2.destroyAllWindows()

**EXP3: Implement k-nearest neighbours classification using python**

# 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.model\_selection 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 K-NN to the Training set

from sklearn.neighbors import KNeighborsClassifier

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

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)

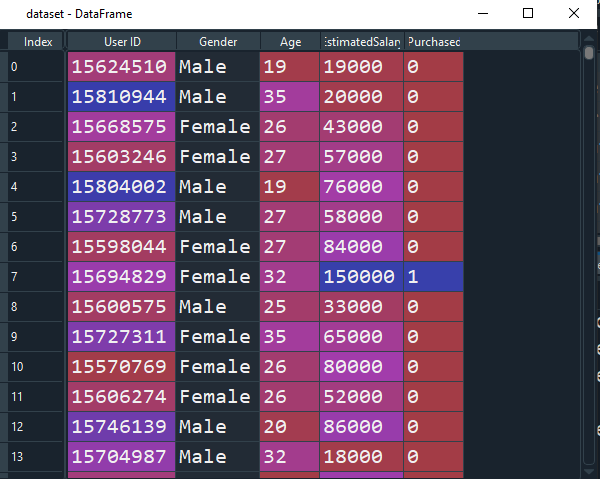
plt.title('K-NN (Test set)')

plt.xlabel('Age')

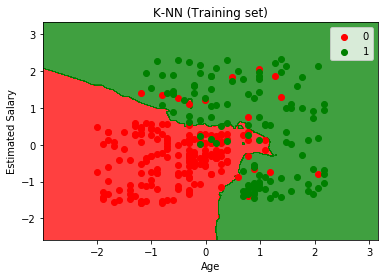
plt.ylabel('Estimated Salary')

plt.legend()

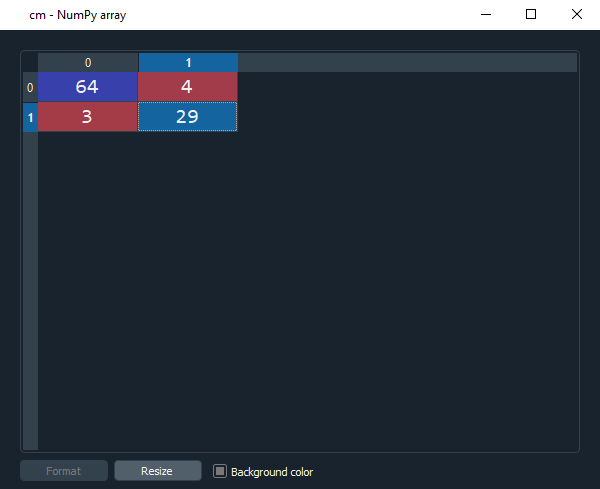
plt.show()

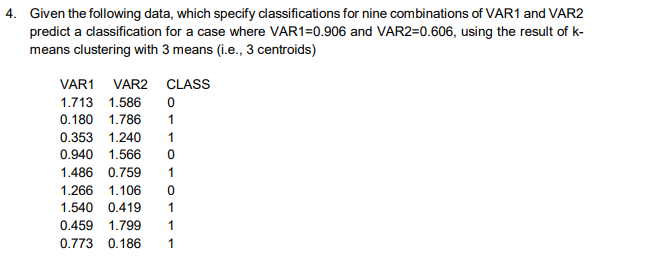
INPUT DATA SET: 

**CLASSIFIER RESULTS IN GRAPH:**



**CONFUSION MATRIX:**





#importing the libraries

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

#importing the Iris dataset with pandas

dataset = pd.read\_csv(r'C:\Users\DELL\Desktop\ml\N\_data.csv')

print(dataset)

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

print(x)

#Finding the optimum number of clusters for k-means classification

from sklearn.cluster import KMeans

wcss = []

for i in range(1, 8):

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

kmeans.fit(x)

wcss.append(kmeans.inertia\_)

print(wcss)

#Plotting the results onto a line graph, allowing us to observe 'The elbow'

plt.plot(range(1, 8), wcss)

plt.title('The elbow method')

plt.xlabel('Number of clusters')

plt.ylabel('WCSS') #within cluster sum of squares

plt.show()

#Applying kmeans to the dataset / Creating the kmeans classifier

kmeans = KMeans(n\_clusters = 3, init = 'k-means++', max\_iter = 300, n\_init = 10, random\_state = 0)

print(kmeans)

#test = pd.read\_csv(r'C:\Users\DELL\Desktop\ml\y.csv')

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

#z\_kmeans = kmeans.fit\_predict(test)

print(y\_kmeans)

#Visualising the clusters

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

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

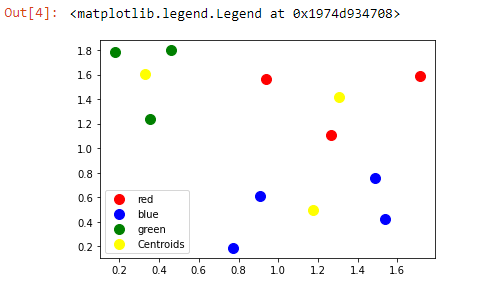
plt.scatter(x[y\_kmeans == 2, 0], x[y\_kmeans == 2, 1], s = 100, c = 'green', label = 'green')

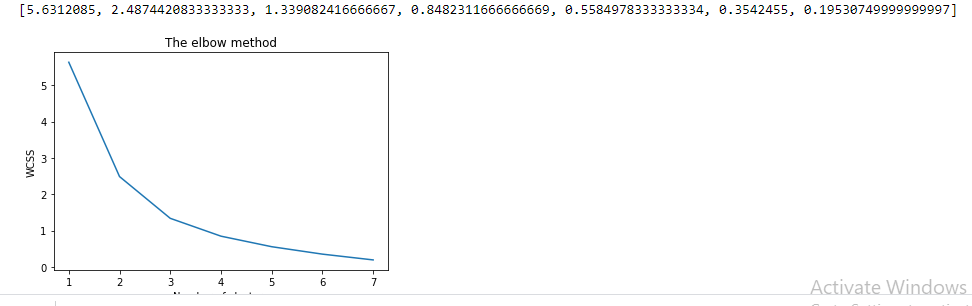
#Plotting the centroids of the clusters

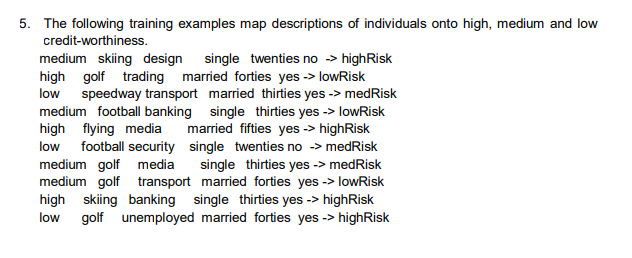
plt.scatter(kmeans.cluster\_centers\_[:, 0], kmeans.cluster\_centers\_[:,1], s = 100, c = 'yellow', label = 'Centroids')

plt.legend()

**CLUSTERS PREDICTED BY THE MODEL:**



**ELBOW METHOD TO FIND NUMBER OF CLUSTERS:**



Input attributes are (from left to right) income, recreation, job, status, age-group, home-owner. Find the unconditional probability of `golf' and the conditional probability of `single' given `medRisk' in the dataset?

"""

Created on Wed Mar 17 14:57:42 2021

@author: DELL

"""

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

import os

# Caluclating unconditional probability

dataset = pd.read\_csv('credit.csv')

total= len(dataset)

#print(os.getcwd())

k= dataset.recreation.value\_counts().golf

Unconditional\_probability= k/total\*100

print('Unconditional probability:',Unconditional\_probability)

# Caluclating conditional probability

# for i in dataset:

# if dataset.status.value == 'single' and dataset.risk == 'medRisk':

count=0

# count=count+1

#print(count)

def cond\_prob(dataset):

if dataset['status'] == 'single' and dataset['risk'] == 'medRisk':

global count

count=count+1

count\_value = dataset.apply(cond\_prob,axis=1)

#print(count)

conditional\_probability = (count/total)\*100

print('conditional probability:',conditional\_probability)

**6. Implement linear regression using python**

# Simple 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('bill.csv')

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

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

# Splitting the dataset into the Training set and Test set

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

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

sc\_y = StandardScaler()

y\_train = sc\_y.fit\_transform(y\_train)"""

# 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)

print('intercept:', regressor.intercept\_)

print('slope:', regressor.coef\_)

r\_sq = regressor.score(X\_train, y\_train)

print('r\_sq',r\_sq)

# 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('Bill vs Tip (Training set)')

plt.xlabel('bill')

plt.ylabel('tip')

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')

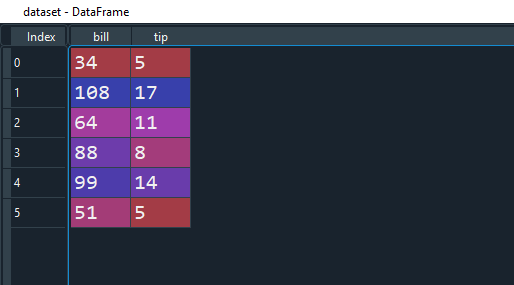
plt.title('Bill vs Tip (Test set)')

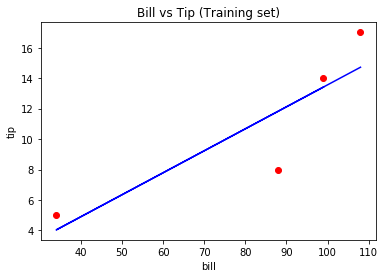
plt.xlabel('bill')

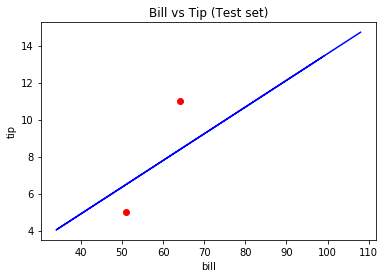
plt.ylabel('tip')

plt.show()

INPUT DATA SET:







**7. Implement Naïve Bayes theorem to classify the English text**

# Natural Language Processing

# Importing the libraries

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

# Importing the dataset

dataset = pd.read\_csv('Restaurant\_Reviews.tsv', delimiter = '\t', quoting = 3)

# Cleaning the texts

import re

import nltk

nltk.download('stopwords')

from nltk.corpus import stopwords

from nltk.stem.porter import PorterStemmer

corpus = []

for i in range(0, 1000):

review = re.sub('[^a-zA-Z]', ' ', dataset['Review'][i])

review = review.lower()

review = review.split()

ps = PorterStemmer()

review = [ps.stem(word) for word in review if not word in set(stopwords.words('english'))]

review = ' '.join(review)

corpus.append(review)

# Creating the Bag of Words model

from sklearn.feature\_extraction.text import CountVectorizer

cv = CountVectorizer(max\_features = 1500)

X = cv.fit\_transform(corpus).toarray()

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

# Splitting the dataset into the Training set and Test set

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

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.20, random\_state = 0)

# Fitting Naive Bayes to the Training set

from sklearn.naive\_bayes import GaussianNB

classifier = GaussianNB()

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

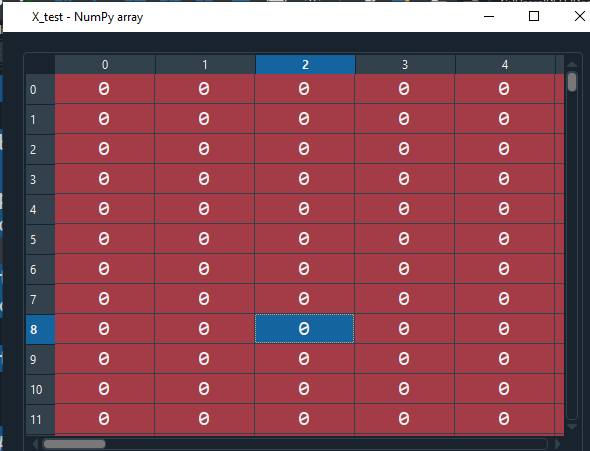
# Predicting the Test set results

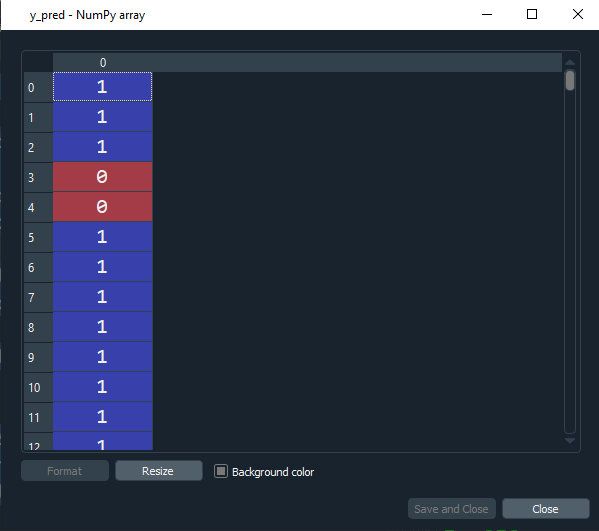
y\_pred = classifier.predict(X\_test)

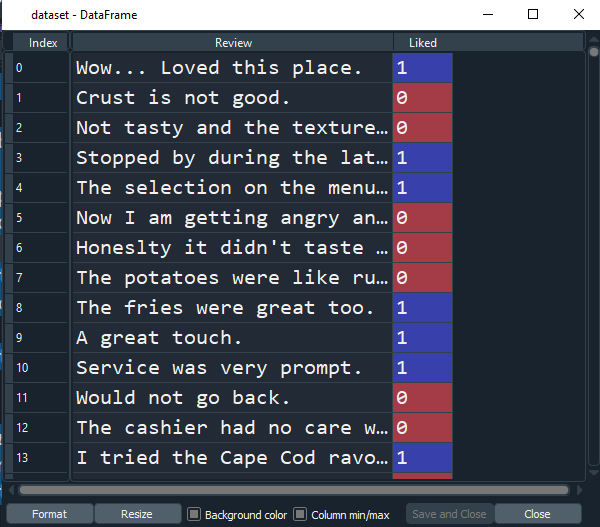
# Making the Confusion Matrix

from sklearn.metrics import confusion\_matrix

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







**8.** Implement an algorithm to demonstrate the significance of genetic algorithm

**"""# genetic algorithm to evaluates**

**# a binary string based on the number**

**#of 1's in the string.**

**# Example: a bitstring with**

**a length of 20 bits will**

**have a score of 20 for**

**# a string of all 1's in**

**the string.**

**(1111111111111111111 = 20,**

**11111111110000000000 =10)"""**

**from numpy.random import randint**

**from numpy.random import rand**

**# objective function**

**def onemax(x):**

**return -sum(x)**

**# tournament selection**

**def selection(pop, scores, k=3):**

**# first random selection**

**selection\_ix = randint(len(pop))**

**for ix in randint(0, len(pop), k-1):**

**# check if better (e.g. perform a tournament)**

**if scores[ix] < scores[selection\_ix]:**

**selection\_ix = ix**

**return pop[selection\_ix]**

**# crossover two parents to create two children**

**def crossover(p1, p2, r\_cross):**

**# children are copies of parents by default**

**c1, c2 = p1.copy(), p2.copy()**

**# check for recombination**

**if rand() < r\_cross:**

**# select crossover point that is not on the end of the string**

**pt = randint(1, len(p1)-2)**

**# perform crossover**

**c1 = p1[:pt] + p2[pt:]**

**c2 = p2[:pt] + p1[pt:]**

**return [c1, c2]**

**# mutation operator**

**def mutation(bitstring, r\_mut):**

**for i in range(len(bitstring)):**

**# check for a mutation**

**if rand() < r\_mut:**

**# flip the bit**

**bitstring[i] = 1 - bitstring[i]**

**# genetic algorithm**

**def genetic\_algorithm(objective, n\_bits, n\_iter, n\_pop, r\_cross, r\_mut):**

**# initial population of random bitstring**

**pop = [randint(0, 2, n\_bits).tolist() for \_ in range(n\_pop)]**

**# keep track of best solution**

**best, best\_eval = 0, objective(pop[0])**

**# enumerate generations**

**for gen in range(n\_iter):**

**# evaluate all candidates in the population**

**scores = [objective(c) for c in pop]**

**# check for new best solution**

**for i in range(n\_pop):**

**if scores[i] < best\_eval:**

**best, best\_eval = pop[i], scores[i]**

**print(">%d, new best f(%s) = %.3f" % (gen, pop[i], scores[i]))**

**# select parents**

**selected = [selection(pop, scores) for \_ in range(n\_pop)]**

**# create the next generation**

**children = list()**

**for i in range(0, n\_pop, 2):**

**# get selected parents in pairs**

**p1, p2 = selected[i], selected[i+1]**

**# crossover and mutation**

**for c in crossover(p1, p2, r\_cross):**

**# mutation**

**mutation(c, r\_mut)**

**# store for next generation**

**children.append(c)**

**# replace population**

**pop = children**

**return [best, best\_eval]**

**# define the total iterations**

**n\_iter = 100**

**# bits**

**n\_bits = 20**

**# define the population size**

**n\_pop = 100**

**# crossover rate**

**r\_cross = 0.9**

**# mutation rate**

**r\_mut = 1.0 / float(n\_bits)**

**# perform the genetic algorithm search**

**best, score = genetic\_algorithm(onemax, n\_bits, n\_iter, n\_pop, r\_cross, r\_mut)**

**print('Done!')**

**print('f(%s) = %f' % (best, score))**

**OUTPUT:**

**score))**

**>0, new best f([1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1]) = -11.000**

**>0, new best f([1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1]) = -12.000**

**>0, new best f([1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1]) = -14.000**

**>0, new best f([1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0]) = -15.000**

**>1, new best f([1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1]) = -16.000**

**>3, new best f([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1]) = -18.000**

**>5, new best f([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1]) = -19.000**

**>8, new best f([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]) = -20.000**

**Done!**

**f([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]) = -20.000000**

**9. Implement the finite words classification system using Back-propagation algorithm**

# Artificial Neural Network

# Installing Theano

# pip install --upgrade --no-deps git+git://github.com/Theano/Theano.git

# Installing Tensorflow

# Install Tensorflow from the website: https://www.tensorflow.org/versions/r0.12/get\_started/os\_setup.html

# Installing Keras

# pip install --upgrade keras

# Part 1 - Data Preprocessing

# Importing the libraries

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

# Importing the dataset

dataset = pd.read\_csv('Restaurant\_Reviews.tsv', delimiter = '\t', quoting = 3)

# Cleaning the texts

import re

import nltk

nltk.download('stopwords')

from nltk.corpus import stopwords

from nltk.stem.porter import PorterStemmer

corpus = []

for i in range(0, 1000):

review = re.sub('[^a-zA-Z]', ' ', dataset['Review'][i])

review = review.lower()

review = review.split()

ps = PorterStemmer()

review = [ps.stem(word) for word in review if not word in set(stopwords.words('english'))]

review = ' '.join(review)

corpus.append(review)

# Creating the Bag of Words model

from sklearn.feature\_extraction.text import CountVectorizer

cv = CountVectorizer(max\_features = 1500)

X = cv.fit\_transform(corpus).toarray()

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

# Splitting the dataset into the Training set and Test set

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

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.20, random\_state = 0)

# Part 2 - Now let's make the ANN!

# Importing the Keras libraries and packages

import keras

from keras.models import Sequential

from keras.layers import Dense

# Initialising the ANN

classifier = Sequential()

# Adding the input layer and the first hidden layer

classifier.add(Dense(output\_dim = 6, init = 'uniform', activation = 'relu', input\_dim = 1500))

# Adding the second hidden layer

classifier.add(Dense(output\_dim = 6, init = 'uniform', activation = 'relu'))

# Adding the output layer

classifier.add(Dense(output\_dim = 1, init = 'uniform', activation = 'sigmoid'))

# Compiling the ANN

classifier.compile(optimizer = 'adam', loss = 'binary\_crossentropy', metrics = ['accuracy'])

# Fitting the ANN to the Training set

classifier.fit(X\_train, y\_train, batch\_size = 10, nb\_epoch = 100)

# Part 3 - Making the predictions and evaluating the model

# Predicting the Test set results

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

y\_pred = (y\_pred > 0.5)

# Making the Confusion Matrix

from sklearn.metrics import confusion\_matrix

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