SYLLABUS

Course Prerequisite: 18CSE101 Programming for Problem Solving (Python)

Course Description:

Performing data labeling, building custom models, object recognition, speech recognition, building chatbot, configuring neural network, building virtual assistant, and building convolutional neural network.

Course Objectives: The objectives of this course are to

1. Perform data labelling
2. Develop custom models for object recognition
3. Build chatbot.
4. Configure neural network.

LIST OF EXPERIMENTS

1. Implement simple linear regression to predict profits for a food truck based on the population of the city that the truck would be placed in.
2. Build a classification model that estimates the probability of admission based on the exam scores using logistic regression.
3. Implement the unsupervised learning algorithm using K-means clustering
4. Implement an anomaly detection algorithm using a Gaussian model and apply it to detect failing servers on a network.
5. Liv.ai - App for Speech recognition and Synthesis through APIs
6. Building a Chatbot
7. Build a virtual assistant
8. Supervised Algorithm - Perform Data Labelling for various images using object recognition
9. Implement un-regularized and regularized versions of the neural network cost function and compute gradients via the backpropagation algorithm.
10. Build a Convolutional Neural Network for Cat vs Dog Image Classification

Course Outcomes:

At the end of the course student will be able to

1. Label the data based on object recognition
2. Develop virtual assistant using speech recognition
3. Develop Chatbots based on the user requirements
4. Design and configure Neural Networks for various real world applications
5. Create convolution neural network model for image classification

Textbooks:

1. Tom Markiewicz& Josh Zheng,Getting started with Artificial Intelligence,Published by O'Reilly Media,2017
2. Programming collective Intelligence: Building Smart Web 2.0 Applications-Toby Segaran
3. Building Machine Learning systems with Python, WilliRichart Luis Pedro Coelho
4. Python Machine Learning by Example, Liu, Yuxi(Hayden),Packt Publishers

5. Stuart J. Russell and Peter Norvig,Artificial Intelligence A Modern Approach

References:

1. AurélienGéron,Hands on Machine Learning with Scikit-Learn and TensorFlow [Concepts, Tools, and Techniques to Build Intelligent Systems], Published by O'Reilly Media,2017
2. Machine Learning with Python, AbhishekVijayvargia, BPB publications
3. Python Machine Learning, Sebastian Raschka, packt publishers

Mode of Evaluation: Observation, Record & End Semester Practical Exams.

Experiment 1

SIMPLE LINEAR REGRESSION MODEL

AIM

To implement linear regression with one variable to predict profits for a food truck. The file ex1data1.txt contains the dataset for our linear regression problem. The first column is the population of a city and the second column is the profit of a food truck in that city. A negative value for profit indicates a loss. Dataset is like below:

Population (10,000s)	Profit (10,000s $)
5.5277	13.662
8.5186	9.1302
6.1101	6.8233

DESCRIPTION

In this part of this exercise, we will implement linear regression with one variable to predict profits for a food truck. Suppose you are the CEO of a restaurant franchise and are considering different cities for opening a new outlet. The chain already has trucks in various cities and you have data for profits and populations from cities. We would like to use this data to help us select which city to expand to next.

The file ex1data1.txt contains the dataset for our linear regression problem. The first column is the population of a city and the second column is the profit of a food truck in that city. A negative value for profit indicates a loss. The first column refers to the population size in 10,000s and the second column refers to the profit in $10,000sThe dataset can be downloaded from below link.

https://www.kaggle.com/saxenapriyansh/coursera-machine-learning-andrew-ng

ALGORITHM

Step 1: Problem and load dataset

Step 2: Compute hypothesis function

Step 3: Compute cost function

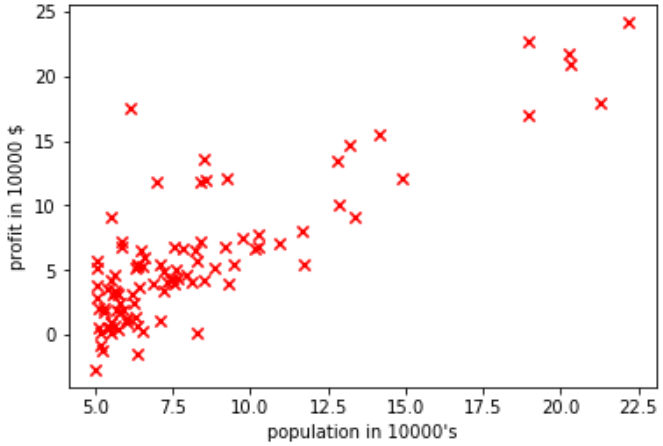
Step 4: Apply gradient descent algorithm

Step 5: Check for optimization

Step 6: Prediction

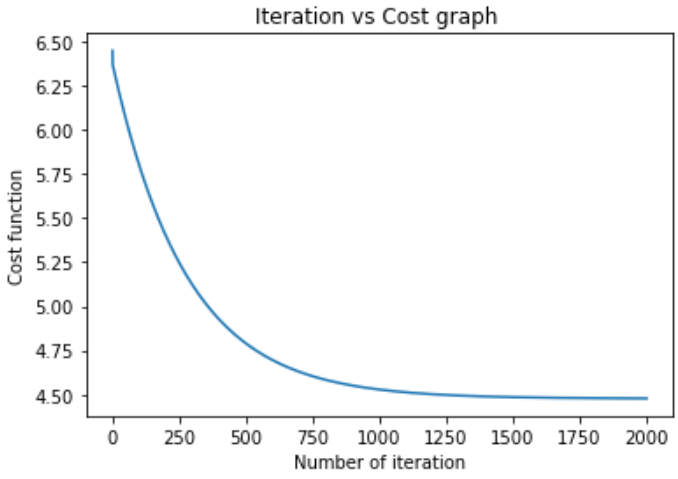
SOURCE CODE

```
import numpy as np
import matplotlib.pyplot as plt
#load data from text file
data = np.loadtxt("ex1data1.txt",delimiter=",")
data = np.array(data)
#seperate the input (X) and output (Y)
X = data[::,:1]
Y = data[::,1:]
plt.scatter(X.transpose(),Y.transpose(),40,color="red",marker="x")
plt.xlabel("population in 10000's")
plt.ylabel("profit in 10000 $")
plt.show()
```

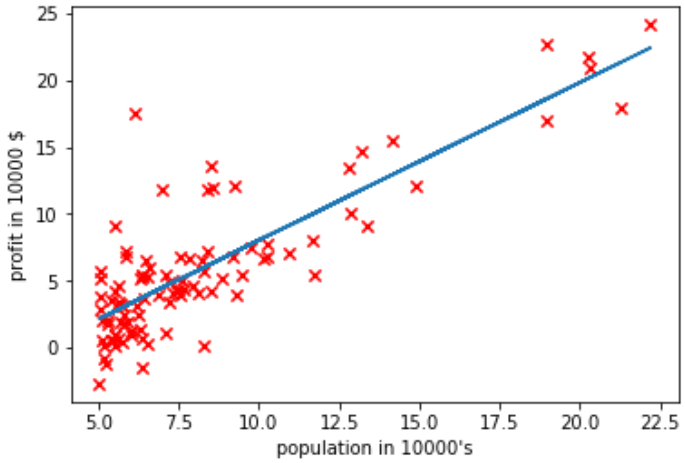



```
# introduce weights of hypothesis (randomly initialize)
Theta = np.random.rand(1,2)
# m is total example set , n is number of features
m,n = X.shape
# add bias to input matrix by simple make X0 = 1 for all
X_bias = np.ones((m,2))
X_bias[::,1:] = X
# output first 5 X_bias examples
X_bias[0:5,:]
#define function to find cost
def cost(X_bias,Y,Theta):
    m,n = X.shape
    hypothesis = X_bias.dot(Theta.transpose())
    return (1/(2.0*m))*((np.square(hypothesis-Y)).sum(axis=0))
#function gradient descent algorithm from minimizing theta
def gradientDescent(X_bias,Y,Theta,iterations,alpha):
    count = 1
    cost_log = np.array([])
    while(count <= iterations):
        hypothesis = X_bias.dot(Theta.transpose())
        temp0 = Theta[0,0] - alpha*(1.0/m)*((hypothesis-Y)*(X_bias[::,0:1])).sum(axis=0)
        temp1 = Theta[0,1] - alpha*(1.0/m)*((hypothesis-Y)*(X_bias[::,-1:])).sum(axis=0)
        Theta[0,0] = temp0
        Theta[0,1] = temp1
```

```
        cost_log = np.append(cost_log,cost(X_bias,Y,Theta))
        count = count + 1
    plt.plot(np.linspace(1,iterations,iterations,endpoint=True),cost_log)
    plt.title("Iteration vs Cost graph ")
    plt.xlabel("Number of iteration")
    plt.ylabel("Cost function")
    plt.show()
    return Theta
alpha = 0.01
iterations = 2000 #the value of iterations is 1500 enough. 2000 uses for demonstration
Theta = gradientDescent(X_bias,Y,Theta,iterations,alpha)
```




```
# Plot showing hypothesis
plt.scatter(X.transpose(),Y.transpose(),40,color="red",marker="x")
X_axis = X
Y_axis = X_bias.dot(Theta.transpose())
plt.plot(X_axis,Y_axis)
plt.xlabel("population in 10000's")
plt.ylabel("profit in 10000 $")
plt.show()
```



INTPUT & OUTPUT

Sample Input Program

```
# predict the profit for city with 35000 and 75000 people
X_test = np.array([[1,3.5],[1,7.5]])
hypothesis = X_test.dot(Theta.transpose())
print 'profit from 35000 people city is ',hypothesis[0,0]*10000,'$'
print 'profit from 35000 people city is ',hypothesis[1,0]*10000,'$'
```

Sample Output:

profit from 35000 people city is 3601.80572428 $ profit from 35000 people city is 50825.2145934 $

RESULT

The implementation of linear regression with one variable to predict profits for a food truck is done successfully.

Experiment 2

LOGISTIC REGRESSION MODEL

AIM

To Build a classification model that estimates the probability of admission based on the exam scores using logistic regression.

DESCRIPTION

A logistic regression model will be implemented to predict whether a student gets admitted into a university. Historical data from previous applicants will be used as a training set. Each training example includes the applicant's scores on two exams and the admissions decision. A classification model that estimates an applicant's probability of admission based on the scores from those two exams will be built.
The dataset can be downloaded from below link:
https://www.kaggle.com/saxenapriyansh/coursera-machine-learning-andrew-ng

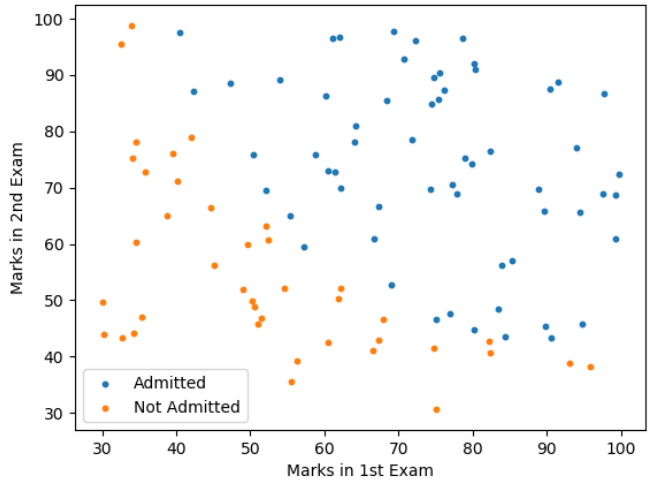
ALGORITHM

1.Load the dataset.
2.Compute hypothesis and cost function
3.Train the model
4.Plot the decision boundary
5.Check accuracy of the model

SOURCE CODE

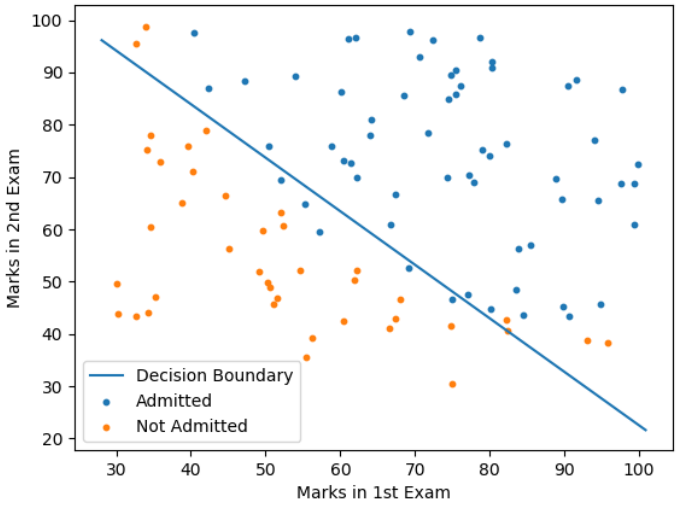
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
def load_data(path, header):
    marks_df = pd.read_csv(path, header=header)
    return marks_df

if __name__ == "__main__":
    # load the data from the file
    data = load_data("data/marks.txt", None)

    # X = feature values, all the columns except the last column
    X = data.iloc[:, :-1]

    # y = target values, last column of the data frame
    y = data.iloc[:, -1]

    # filter out the applicants that got admitted
    admitted = data.loc[y == 1]
```

```
    # filter out the applicants that din't get admission
    not_admitted = data.loc[y == 0]

    # plots
    plt.scatter(admitted.iloc[:, 0], admitted.iloc[:, 1], s=10, label='Admitted')
    plt.scatter(not_admitted.iloc[:, 0], not_admitted.iloc[:, 1], s=10, label='Not Admitted')
    plt.legend()
    plt.show()
```




```
X = np.c_[np.ones((X.shape[0], 1)), X]
y = y[:, np.newaxis]
theta = np.zeros((X.shape[1], 1))
def sigmoid(x):
    # Activation function used to map any real value between 0 and 1
    return 1 / (1 + np.exp(-x))

def net_input(theta, x):
    # Computes the weighted sum of inputs
    return np.dot(x, theta)

def probability(theta, x):
    # Returns the probability after passing through sigmoid
    return sigmoid(net_input(theta, x))
def cost_function(self, theta, x, y):
    # Computes the cost function for all the training samples
    m = x.shape[0]
    total_cost = -(1 / m) * np.sum(
        y * np.log(probability(theta, x)) + (1 - y) * np.log(
            1 - probability(theta, x)))
    return total_cost

def gradient(self, theta, x, y):
    # Computes the gradient of the cost function at the point theta
    m = x.shape[0]
    return (1 / m) * np.dot(x.T, sigmoid(net_input(theta,   x)) - y)
def fit(self, x, y, theta):
```

```
    opt_weights = fmin_tnc(func=cost_function, x0=theta,
                  fprime=gradient,args=(x, y.flatten()))
    return opt_weights[0]
parameters = fit(X, y, theta)
[-25.16131856 0.20623159 0.20147149]
x_values = [np.min(X[:, 1] - 5), np.max(X[:, 2] + 5)]
y_values = - (parameters[0] + np.dot(parameters[1], x_values)) / parameters[2]

plt.plot(x_values, y_values, label='Decision Boundary')
plt.xlabel('Marks in 1st Exam')
plt.ylabel('Marks in 2nd Exam')
plt.legend()
plt.show()
```




```
def predict(self, x):
    theta = parameters[:, np.newaxis]
    return probability(theta, x)
def accuracy(self, x, actual_classes, probab_threshold=0.5):
    predicted_classes = (predict(x) >=
                         probab_threshold).astype(int)
    predicted_classes = predicted_classes.flatten()
    accuracy = np.mean(predicted_classes == actual_classes)
    return accuracy * 100
accuracy(X, y.flatten())
```

OUTPUT:

The accuracy of the model is 89%.

RESULT:

The implementation of logistic regression to estimate the probability of admission based on the exam scores is done successfully.

Experiment 3:

UNSUPERVISED LEARNING ALGORITHM - K-MEANS CLUSTERING

AIM:

To implement the unsupervised learning algorithm using K-means clustering

DESCRIPTION

Assume you are owing a supermarket mall and through membership cards, you have some basic data about your customers like Customer ID, age, gender, annual income and spending score. You want to understand the customers like who are the target customers so that the sense can be given to marketing team and plan the strategy accordingly.

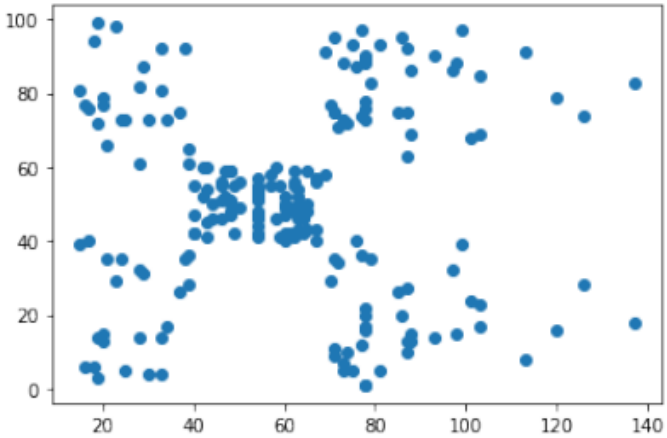
The dataset can be downloaded from the below link:
https://www.kaggle.com/shwetabh123/mall-customers

ALGORITHM

Step 1: Choose the number of clusters k. ...
Step 2: Select k random points from the data as centroids. ...
Step 3: Assign all the points to the closest cluster centroid. ...
Step 4: Recompute the centroids of newly formed clusters. ...
Step 5: Repeat steps 3 and Step 4.

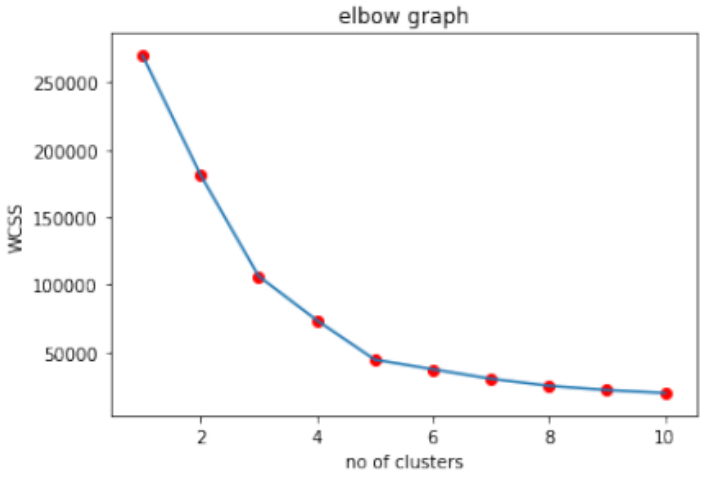
SOURCE CODE

```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
from sklearn.cluster import KMeans
data=pd.read_csv("Mall_Customers.csv")
x=data.iloc[:,3:].values
#plot the data among these two features
plt.scatter(x[:,0],x[:,1]
```



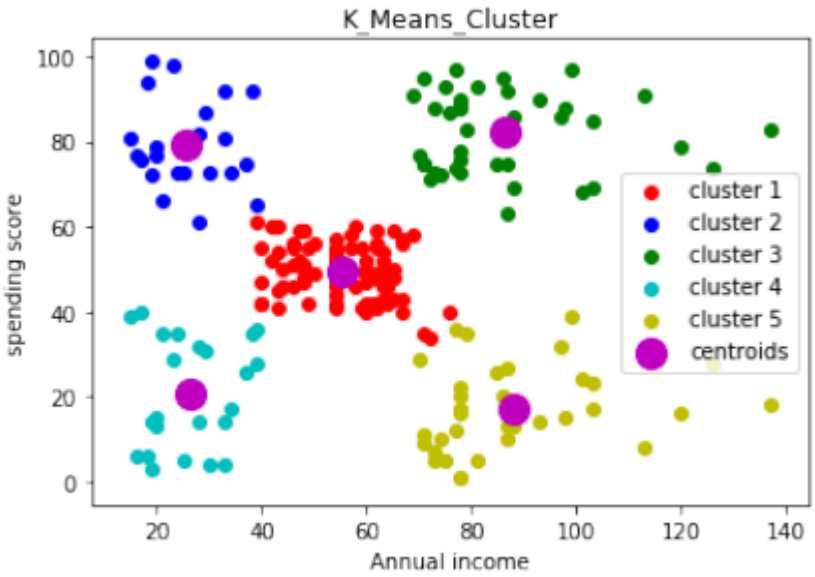

```
# to find the number of clusters WCSS(inertia) where we plot elbow graph to find k
```

```
wcss=[]
for i in range(1,11): # trial and error
 k=KMeans(n_clusters=i,init="k-means++",random_state=0)
 k.fit(x)
 wcss.append(k.inertia_)
#plot a graph by taking iteration values and WCSS
plt.plot(range(1,11),wcss)
plt.scatter(range(1,11),wcss,c='r')
plt.title("elbow graph")
plt.xlabel("no of clusters")
plt.ylabel("WCSS")
```



By looking into the elbow graph the optimal cluster values is 5 (K value)

```
# let us find which dataset belongs to clusters
k=KMeans(n_clusters=5,init="k-means++",random_state=0)
y=k.fit_predict(x) # will help us which data poins belongs to cluster
# position of the centroids at centres
k.cluster_centers_
array([[55.2962963 , 49.51851852],
 [25.72727273, 79.36363636],
 [86.53846154, 82.12820513],
 [26.30434783, 20.91304348],
 [88.2 , 17.11428571]])
# plot the final graph
plt.scatter(x[y==0,0],x[y==0,1],c='r',label="cluster 1")
plt.scatter(x[y==1,0],x[y==1,1],c='b',label="cluster 2")
plt.scatter(x[y==2,0],x[y==2,1],c='g',label="cluster 3")
plt.scatter(x[y==3,0],x[y==3,1],c='c',label="cluster 4")
plt.scatter(x[y==4,0],x[y==4,1],c='y',label="cluster 5")
plt.scatter(k.cluster_centers_[:,0],k.cluster_centers_[:,1],s=200,
plt.legend()
plt.title("K_Means_Cluster")
plt.xlabel("Annual income")
plt.ylabel("spending score")
```



OUTPUT

Interpretation of the graph
Cluster 4: Less salary and less spending score
Cluster 2: Less salary and high spending score
Cluster 1: Average salary and average spending score
Cluster 3: High salary and High spending score (Offers can be provided)
Cluster 5: High salary and less spending score (discounts can be provided)

RESULT:

The Implementation of the unsupervised learning algorithm using K-means clustering is done successfully.

Experiment 4

ANOMALY DETECTION ALGORITHM IMPLEMENTATION USING A GAUSSIAN MODEL

AIM

To implement an anomaly detection algorithm using a Gaussian model and apply it to detect failing servers on a network.

DESCRIPTION

Implement an anomaly detection algorithm using a Gaussian model and apply it to detect failing servers on a network. We'll also see how to build a recommendation system using collaborative filtering and apply it to a movie recommendations data set.

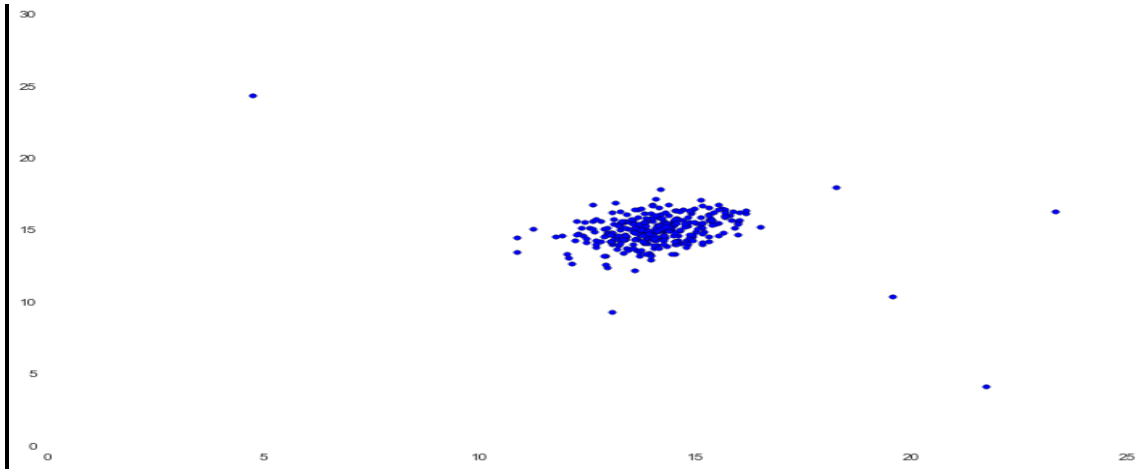
The dataset can be downloaded from below link:
https://github.com/jdwittenauer/ipython-notebooks/blob/master/exercises/ML/ex8.pdf

ALGORITHM

Step 1: Visualize the data
Step 2: Calculate mean and variance
Step 3: Calculate the probability that a data point belongs to a normal distribution
Step 4: Compute and save the probability density of each of the values in our data set
Step 5: Calculate the F1 score for varying values of epsilon.
Step 6: Apply the threshold to the data set and visualize the results.

SOURCE CODE

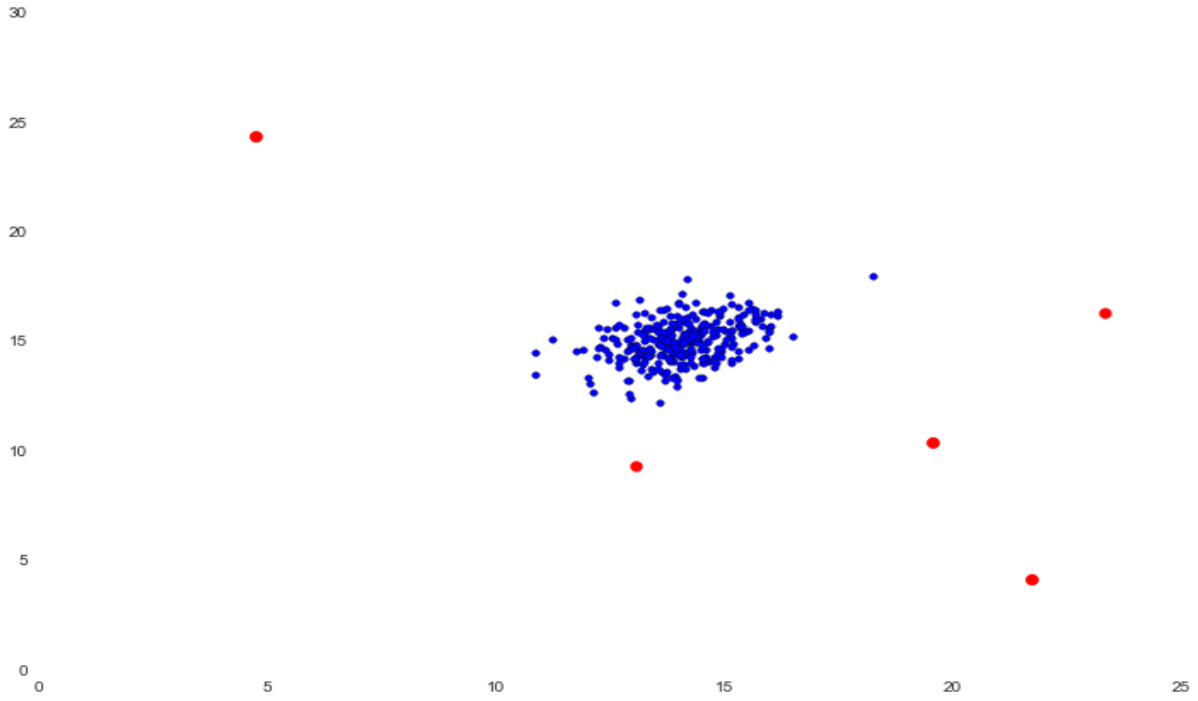
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sb
from scipy.io import loadmat
%matplotlib inline

data = loadmat('data/ex8data1.mat')
X = data['X']
X.shape
fig, ax = plt.subplots(figsize=(12,8))
ax.scatter(X[:,0], X[:,1])
```




```
def estimate_gaussian(X):
    mu = X.mean(axis=0)
    sigma = X.var(axis=0)

    return mu, sigma

mu, sigma = estimate_gaussian(X)
mu, sigma
(array([ 14.11222578,  14.99771051]), array([ 1.83263141,  1.70974533]))
Xval = data['Xval']
yval = data['yval']

Xval.shape, yval.shape
((307L, 2L), (307L, 1L))
from scipy import stats
dist = stats.norm(mu[0], sigma[0])
dist.pdf(X[:,0])[0:50]
p = np.zeros((X.shape[0], X.shape[1]))
p[:,0] = stats.norm(mu[0], sigma[0]).pdf(X[:,0])
p[:,1] = stats.norm(mu[1], sigma[1]).pdf(X[:,1])

p.shape
pval = np.zeros((Xval.shape[0], Xval.shape[1]))
pval[:,0] = stats.norm(mu[0], sigma[0]).pdf(Xval[:,0])
pval[:,1] = stats.norm(mu[1], sigma[1]).pdf(Xval[:,1])
def select_threshold(pval, yval):
    best_epsilon = 0
    best_f1 = 0
    f1 = 0

    step = (pval.max() - pval.min()) / 1000

    for epsilon in np.arange(pval.min(), pval.max(), step):
        preds = pval < epsilon
```

```
        tp = np.sum(np.logical_and(preds == 1, yval == 1)).astype(float)
        fp = np.sum(np.logical_and(preds == 1, yval == 0)).astype(float)
        fn = np.sum(np.logical_and(preds == 0, yval == 1)).astype(float)

        precision = tp / (tp + fp)
        recall = tp / (tp + fn)
        f1 = (2 * precision * recall) / (precision + recall)

        if f1 > best_f1:
            best_f1 = f1
            best_epsilon = epsilon

    return best_epsilon, best_f1

epsilon, f1 = select_threshold(pval, yval)
epsilon, f1
(0.0095667060059568421, 0.7142857142857143)
# indexes of the values considered to be outliers
outliers = np.where(p < epsilon)

fig, ax = plt.subplots(figsize=(12,8))
ax.scatter(X[:,0], X[:,1])
ax.scatter(X[outliers[0],0], X[outliers[0],1], s=50, color='r', marker='o')
```

OUTPUT



RESULT

The Implementation of the anomaly detection algorithm using a Gaussian model is done successfully.

Experiment 5

SPEECH RECOGNITION AND SYNTHESIS THROUGH APIS

AIM

To implement Speech recognition and Synthesis through APIs

DESCRIPTION

Speech recognition helps us to save time by speaking instead of typing. It also gives us the power to communicate with our devices without even writing one line of code. This makes technological devices more accessible and easier to use. Speech recognition is a great example of using machine learning in real life.
Sample .wav file can be downloaded from below link:
https://github.com/leocaseiro/Offline-free-audiotranscriber/blob/master/LongWelcome.wav

ALGORITHM

Step 1: Install Speech Recognition Libraries
Step 2: Recognizer Class
Step 3: Speech Recognition Functions
Step 4: Audio Preprocessing

SOURCE CODE

```
import speech_recognition as sr
!pip install SpeechRecognition
 #for recognising speech initialze recogniser
k=sr.Recognizer()
a=sr.AudioFile('LongWelcome.wav') # store audio in to a
with a as source:
 audio=k.record(source)
 try:
 t=k.recognize_google(audio)
 print("converting audio into text.....")
 print(t)
 except:
 print("not recognizing.....")
converting audio into text.....
with a as source:
 a1=k.record(source,duration=6)
k.recognize_google(a1)
```

Speech recognition using Microphones

```
pip install --upgrade pip
brew install portaudio
!pip install pyaudio
import speech_recognition as sr
```

```
r=sr.Recognizer()
mic=sr.Microphone()
sr.Microphone.list_microphone_names()
with mic as source:
 b=r.listen(source)
 try:
 text = r.recognize_google(b)
 print("say something")
 print("You said : {}".format(text))
 #text = r.recognize_google(audio)
 except:
 print("Sorry could not recognize your voice")
```

RESULT

The Implementation of the Speech recognition algorithm for converting audio to text and using microphones is done successfully.

Experiment 6

DESIGN OF CHATBOT

AIM

Design a chatbot using using the ChatterBot Python library.

DESCRIPTION:

A chatbot is an intelligent piece of software that is capable of communicating and performing actions similar to a human. Chatbots are used a lot in customer interaction, marketing on social network sites and instantly messaging the client. Chatbots can learn from behaviour and experiences, they can respond to a wide range of queries and commands. There are two basic types of chatbot models based on how they are built; Retrieval based and Generative based models.

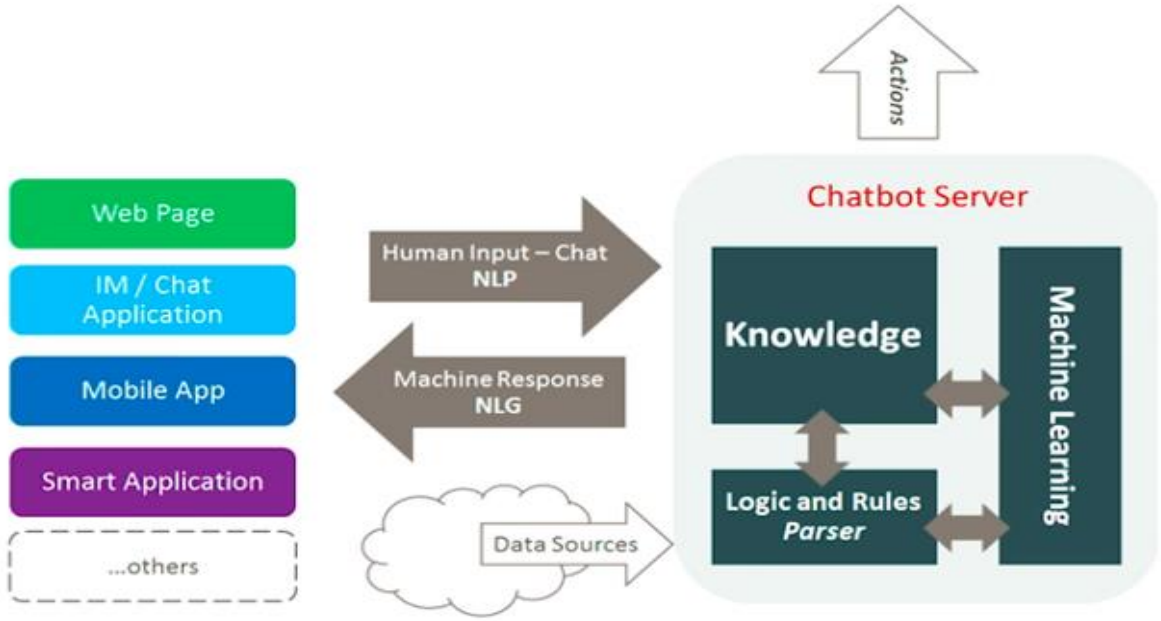
1. Retrieval based Chatbots

A retrieval-based chatbot uses predefined input patterns and responses. It then uses some type of heuristic approach to select the appropriate response. It is widely used in the industry to make goal-oriented chatbots where we can customize the tone and flow of the chatbot to drive our customers with the best experience.

2. Generative based Chatbots

Generative models are not based on some predefined responses.

They are based on sequence to sequence neural networks. It is the same idea as machine translation. In machine translation, we translate the source code from one language to another language but here, we are going to transform input into an output. It needs a large amount of data and it is based on Deep Neural networks.



ChatterBot Library

ChatterBot is a Python library that is designed to deliver automated responses to user inputs. It makes use of a combination of ML algorithms to generate many different types of responses. ChatterBot is programming language independence.

ChatterBot working principle

When a user enters a specific input in the chatbot (developed on ChatterBot), the bot saves the input along with the response for future use. This data (of collected experiences) allows the chatbot to generate automated responses each time a new input is fed into it. The program chooses the most-fitting response from the closest statement that matches the input, and then delivers a response from the already known selection of statements and responses. Over time, as the chatbot engages in more interactions, the accuracy of response improves.

Preprocess data: Tokenization is fragment the large text dataset into smaller, readable chunks (like words). Then transforms a word into its lemma form (lemmatization). Then it creates a pickle file to store the python objects that are used for predicting the responses of the bot.

chatterbot.logic.MathematicalEvaluation enables the bot to solve math problems, while chatterbot.logic.BestMatch chooses the best match from the already provided responses.

ALGORITHM

- Install chatterbot libraries
  - pip install chatterbot
  - pip install chatterbot_corpus
- Import the classes
  - from chatterbot import ChatBot
  - from chatterbot.trainers import ListTrainer
- Create and train the Chatbot

```
my_bot = ChatBot(name='PyBot', read_only=True,
                 logic_adapters=['chatterbot.logic.MathematicalEvaluation',
                                 'chatterbot.logic.BestMatch'])
```

- Build the model
- Predict the response

https://chatbotslife.com/how-to-create-an-intelligent-chatbot-in-python-c655eb39d6b1

SOURCE CODE

```
from chatterbot import ChatBot
from chatterbot.trainers import ListTrainer

my_bot = ChatBot(name='PyBot', read_only=True,
                 logic_adapters=['chatterbot.logic.MathematicalEvaluation',
                                 'chatterbot.logic.BestMatch'])

small_talk = ['hi there!',
              'hi!',
              'how do you do?',
              'how are you?',
              'i\'m cool.',
```

```
            'fine, you?',
            'always cool.',
            'i\'m ok',
            'glad to hear that.',
            'i\'m fine',
            'glad to hear that.',
            'i feel awesome',
            'excellent, glad to hear that.',
            'not so good',
            'sorry to hear that.',
            'what\'s your name?',
            'i\'m pybot. ask me a math question, please.']
math_talk_1 = ['pythagorean theorem',
             'a squared plus b squared equals c squared.']
math_talk_2 = ['law of cosines',
             'c**2 = a**2 + b**2 - 2 * a * b * cos(gamma)']

list_trainer = ListTrainer(my_bot)
for item in (small_talk, math_talk_1, math_talk_2):
   list_trainer.train(item)

from chatterbot.trainers import ChatterBotCorpusTrainer
corpus_trainer = ChatterBotCorpusTrainer(my_bot)
corpus_trainer.train('chatterbot.corpus.english')
```

INTPUT & OUTPUT

Sample Input Program

```
>>> print(my_bot.get_response("hi"))
how do you do?
>>> print(my_bot.get_response("i feel awesome today"))
excellent, glad to hear that.
 >>> print(my_bot.get_response("what's your name?"))
i'm pybot. ask me a math question, please.
>>> print(my_bot.get_response("show me the pythagorean theorem"))
a squared plus b squared equals c squared.
>>> print(my_bot.get_response("do you know the law of cosines?"))
c**2 = a**2 + b**2 - 2 * a * b * cos(gamma)
```

RESULT

The chatbot has been designed & tested using chatterbot library in python and output has been verified.

Experiment 7

VIRTUAL ASSISTANT

AIM

Build a personal assistant for weather searching using Python

DESCRIPTION

User interacts with laptop's microphone/console using virtual assistant. The response generated by the assistant will display on the console or as a speech via the speaker.

Applications: Weather forecasting, Launch Games, Launch Windows Applications, Open Websites, tells you about almost everything you ask, tells you date and time, greetings, news, etc.

Speech Recognition Package: The Speech Recognition package allows Python to access audio from your machine's microphone, transcribe audio, save audio to an audio file, and other similar tasks.

Text to Speech Package: gTTS package (Google Text-to-Speech) interfaces with Google Translate's API. It converts voiced question to a text one. Then, it will need to convert the response (answers) into a voiceable phrase.

Audio Playback Package: The mpyg321 package allows for Python to play MP3 files.

Create a personal digital assistant for establishing voice communication using the libraries such as speech_recognition, time, os, gtts , requests, json. SpeechRecognition activates machine's microphone, and then converts the audio to text in the form of a string.

The listen and respond functions establish the verbal interaction of a digital virtual assistant.

ALGORITHM

- Install required libraries.
- Import the libraries
- Create two functions Listening using SpeechRecognition library and Responding using gTTS library.
    - Listen function activate machine's microphone, and then converts the audio to text in the form of a string.
    - function respond that takes a string input, prints it, then converts the string to an audio file. This audio file is saved to the local directory and then played by operating system.
- Construct digital assistant

SOURCE CODE

```
import speech_recognition as sr
from time import ctime          //current time
import time
import os
from gtts import gTTS
import requests, json
//listen function

def listen():
    r = sr.Recognizer()
```

```
    with sr.Microphone() as source:
        print("I am listening...")
        audio = r.listen(source)
    data = ""
    try:
        data = r.recognize_google(audio)
        print("You said: " + data)
    except sr.UnknownValueError:
        print("Google Speech Recognition did not understand audio")
    except sr.RequestError as e:
        print("Request Failed; {0}".format(e))
    return data
//respond function
def respond(audioString):
    print(audioString)
    tts = gTTS(text=audioString, lang='en')
    tts.save("speech.mp3")
    os.system("mpg321 speech.mp3")

//construct our digital assistant
def digital_assistant(data):
    if "how are you" in data:
        listening = True
        respond("I am well")

    if "what time is it" in data:
        listening = True
        respond(ctime())

    if "what is the weather in" in data:
        listening = True
        api_key = "Your_API_key"
        weather_url = "http://api.openweathermap.org/data/2.5/weather?"
        data = data.split(" ")
        location = str(data[5])
        url = weather_url + "appid=" + api_key + "&q=" + location
        js = requests.get(url).json()
        if js["cod"] != "404":
            weather = js["main"]
            temp = weather["temp"]
            hum = weather["humidity"]
            desc = js["weather"][0]["description"]
```

```
            resp_string = " The temperature in Kelvin is " + str(temp) + " The humidity is " +
str(hum) + " and The weather description is "+ str(desc)
            respond(resp_string)
        else:
            respond("City Not Found")
    if "stop listening" in data:
        listening = False
        print('Listening stopped')
    return listening
time.sleep(2)
respond("Hi Dante, what can I do for you?")
listening = True
while listening == True:
    data = listen()
    listening = digital_assistant(data)
```

RESULT

The personal digital assistant for searching weather has been designed and output has been verified.