INDUSTRY INTERNSHIP

SUMMARY REPORT

Bootcamp Of AIML

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

Submitted by: -

Ishan Srivastava

(Admission no. :18SCSE1140007)

SCHOOL OF COMPUTING SCIENCE AND ENGINEERING
GREATER NOIDA, UTTAR PRADESH

Winter 2021– 2022

BONAFIDE CERTIFICATE

Certificate of Participation

PRESENTED TO

Ishan Srivastava

has attended online workshop on Artificial Intelligence and Machine Learning organised by YBI Foundation.

Friday, Mar 11 2022
12444000001761556

YBI Foundation
www.ybifoundation.org

CERTIFICATE

I hereby certify that the work which is being presented in the Internship project report entitled " Bootcamp in AIML "in partial fulfilment for the requirements for the award of the degree of Bachelor of Technology in the School of Computing Science and Engineering of Galgotias University , Greater Noida, is an authentic record of my own work carried out in the industry.

To the best of my knowledge, the matter embodied in the project report has not been submitted to any other University/Institute for the award of any Degree.

Ishan Srivastava

(Admission no:18SCSE1140007)

This is to certify that the above statement made by the candidate is correct and true to the best of my knowledge.

Signature of Internship Coordinator

Dr.N.Partheeban
Professor & IIIC
School of Computing Science & Engineering
Galgotias University
Greater Noida.

Signature of Dean (SCSE)

Dr. MUNISH SABHARWAL
Professor & Dean
School of Computing Science & Engineering
Galgotias University
Greater Noida.

ABSTRACT

AIML stands for **Artificial Intelligence Markup Language**. AIML was developed by the Alicebot free software community and Dr. Richard S. Wallace during 1995-2000. AIML is used to create or customize Alicebot which is a chat-box application based on A.L.I.C.E. (Artificial Linguistic Internet Computer Entity) free software. Machine Learning, as the name suggests, is the science of programming a computer by which they are able to learn from different kinds of data. A more general definition given by Arthur Samuel is – "Machine Learning is the field of study that gives computers the ability to learn without being explicitly programmed." They are typically used to solve various types of life problems.

AIML Tags

<aiml>

Defines the beginning and end of a AIML document.

<category>

Defines the **unit of knowledge** in Alicebot's knowledge base.

<pattern>

Defines the pattern to match what a user may input to an Alicebot.

<template>

Defines the response of an Alicebot to user's input.

<star>

Used to match wild card ***** character(s) in the <pattern> Tag.

<srai>

Multipurpose tag, used to call/match the other categories.

<random>

Used **<random>** to get random responses.

<li>

Used to represent multiple responses.

<set>

Used to set value in an AIML variable.

<get>

Used to get value stored in an AIML variable.

<that>

Used in AIML to respond based on the context.

AIML Libraries:

In the older days, people used to perform Machine Learning tasks by manually coding all the algorithms and mathematical and statistical formula. This made the process time consuming, tedious and inefficient. But in the modern days, it is become very much easy and efficient compared to the olden days by various python libraries, frameworks, and modules. Today, Python is one of the most popular programming languages for this task and it has replaced many languages in the industry, one of the reason is its vast collection of libraries. Python libraries that used in Machine Learning are:

- Numpy
- Scipy
- Scikit-learn
- Theano
- TensorFlow
- Keras
- PyTorch
- Pandas
- Matplotlib

NumPy : is a very popular python library for large multi-dimensional array and matrix processing, with the help of a large collection of high-level mathematical functions. It is very useful for fundamental scientific computations in Machine Learning. It is particularly useful for linear algebra, Fourier transform, and random number capabilities. High-end libraries like TensorFlow uses NumPy internally for manipulation of Tensors.

Python code Using Numpy:

```
import numpy as np

x = np.array([[1, 2], [3, 4]])
y = np.array([[5, 6], [7, 8]])
```

```
v = np.array([9, 10])
w = np.array([11, 12])

print(np.dot(v, w), "\n")

print(np.dot(x, v), "\n")

print(np.dot(x, y))
```

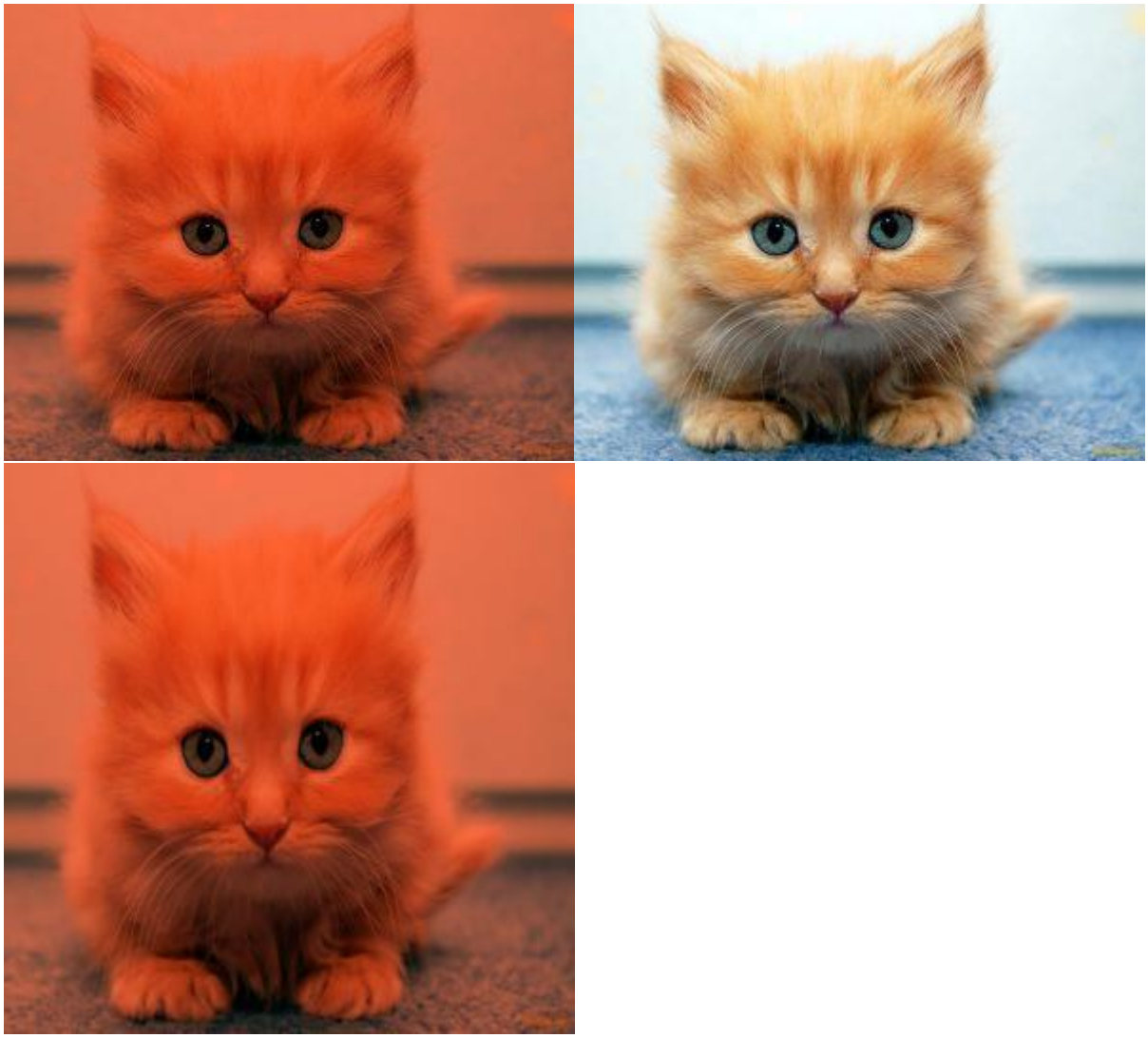
SciPy: is a very popular library among Machine Learning enthusiasts as it contains different modules for optimization, linear algebra, integration and statistics. There is a difference between the SciPy library and the SciPy stack. The SciPy is one of the core packages that make up the SciPy stack. SciPy is also very useful for image manipulation.

```
from scipy.misc import imread, imsave, imresize

img = imread('D:/Programs / cat.jpg') # path of the image
print(img.dtype, img.shape)
img_tint = img * [1, 0.45, 0.3]

imsave('D:/Programs / cat_tinted.jpg', img_tint)
img_tint_resize = imresize(img_tint, (300, 300))
imsave('D:/Programs / cat_tinted_resized.jpg', img_tint_resize)
```

Output Examples:



Scikit-learn : is one of the most popular ML libraries for classical ML algorithms. It is built on top of two basic Python libraries, viz., NumPy and SciPy. Scikit-learn supports most of the supervised and unsupervised learning algorithms. Scikit-learn can also be used for data-mining and data-analysis, which makes it a great tool who is starting out with ML.

Python Code:

```
from sklearn import datasets
from sklearn import metrics
from sklearn.tree import DecisionTreeClassifier
```

```
dataset = datasets.load_iris()
model = DecisionTreeClassifier()
model.fit(dataset.data, dataset.target)
print(model)
expected = dataset.target
predicted = model.predict(dataset.data)
print(metrics.classification_report(expected, predicted))
print(metrics.confusion_matrix(expected, predicted))
```

Output:

```
DecisionTreeClassifier(class_weight=None, criterion='gini',
max_depth=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best')
```

```
0       1.00      1.00      1.00        50
          1       1.00      1.00      1.00        50
          2       1.00      1.00      1.00        50
```

micro avg	1.00	1.00	1.00	150
macro avg	1.00	1.00	1.00	150
weighted avg	1.00	1.00	1.00	150

```
[[50  0  0]
 [ 0 50  0]
 [ 0  0 50]]
```

Theano :
is a popular python library that is used to define, evaluate and optimize mathematical expressions involving multi-dimensional arrays in an efficient manner. It is achieved by optimizing the utilization of CPU and GPU. It is extensively used for unit-testing and self-verification to detect and diagnose different types of errors. Theano is a very powerful library that has been used in large-scale computationally intensive scientific projects for a long time but is simple and approachable enough to be used by individuals for their own projects.

Python Code:

```
import theano
import theano.tensor as T
x = T.dmatrix('x')
s = 1 / (1 + T.exp(-x))
logistic = theano.function([x], s)
logistic([[0, 1], [-1, -2]])
```

Output:

```
array([[0.5, 0.73105858],
       [0.26894142, 0.11920292]])
```

TensorFlow is a very popular open-source library for high performance numerical computation developed by the Google Brain team in Google. As the name suggests, Tensorflow is a framework that involves defining and running computations involving tensors. It can train and run deep neural networks that can be used to develop several AI applications. TensorFlow is widely used in the field of deep learning research and application.

Python Code:

```
import tensorflow as tf

x1 = tf.constant([1, 2, 3, 4])
x2 = tf.constant([5, 6, 7, 8])
result = tf.multiply(x1, x2)
sess = tf.Session()
print(sess.run(result))
sess.close()
```

Output:

```
[ 5 12 21 32]
```

Keras is a very popular Machine Learning library for Python. It is a high-level neural networks API capable of running on top of TensorFlow, CNTK, or

Theano. It can run seamlessly on both CPU and GPU. Keras makes it really for ML beginners to build and design a Neural Network. One of the best thing about Keras is that it allows for easy and fast prototyping.
For more details refer to documentation.

PyTorch is a popular open-source Machine Learning library for Python based on Torch, which is an open-source Machine Learning library which is implemented in C with a wrapper in Lua. It has an extensive choice of tools and libraries that supports on Computer Vision, Natural Language Processing(NLP) and many more ML programs. It allows developers to perform computations on Tensors with GPU acceleration and also helps in creating computational graphs.

Python Code:

```
import torch



dtype = torch.float
device = torch.device("cpu")
# device = torch.device("cuda:0") Uncomment this to run on GPU

# N is batch size; D_in is input dimension;
# H is hidden dimension; D_out is output dimension.
N, D_in, H, D_out = 64, 1000, 100, 10

# Create random input and output data
x = torch.random(N, D_in, device = device, dtype = dtype)
y = torch.random(N, D_out, device = device, dtype = dtype)
```

```
# Randomly initialize weights
w1 = torch.random(D_in, H, device = device, dtype = dtype)
w2 = torch.random(H, D_out, device = device, dtype = dtype)

learning_rate = 1e-6
for t in range(500):
        # Forward pass: compute predicted y
        h = x.mm(w1)
        h_relu = h.clamp(min = 0)
        y_pred = h_relu.mm(w2)

        # Compute and print loss
        loss = (y_pred - y).pow(2).sum().item()
        print(t, loss)

        # Backprop to compute gradients of w1 and w2 with respect to loss
        grad_y_pred = 2.0 * (y_pred - y)
        grad_w2 = h_relu.t().mm(grad_y_pred)
        grad_h_relu = grad_y_pred.mm(w2.t())
        grad_h = grad_h_relu.clone()
        grad_h[h < 0] = 0
        grad_w1 = x.t().mm(grad_h)

        # Update weights using gradient descent
        w1 -= learning_rate * grad_w1
        w2 -= learning_rate * grad_w2
```

Output:

0 47168344.0

1 46385584.0

2 43153576.0

...

...

...

497 3.987660602433607e-05

498 3.945609932998195e-05

499 3.897604619851336e-05

Pandas is a popular Python library for data analysis. It is not directly related to Machine Learning. As we know that the dataset must be prepared before training. In this case, Pandas comes handy as it was developed specifically for data extraction and preparation. It provides high-level data structures and wide variety tools for data analysis. It provides many inbuilt methods for groping, combining and filtering data.

Python code:

```
# Python program using Pandas for

# arranging a given set of data

# into a table


# importing pandas as pd

import pandas as pd


data = {"country": ["Brazil", "Russia", "India", "China", "South Africa"],

        "capital": ["Brasilia", "Moscow", "New Delhi", "Beijing", "Pretoria"],

        "area": [8.516, 17.10, 3.286, 9.597, 1.221],
```

```
        "population": [200.4, 143.5, 1252, 1357, 52.98] }

data_table = pd.DataFrame(data)
print(data_table)
```

Output:

	country	capital	area	population
0	Brazil	Brasilia	8.516	200.40
1	Russia	Moscow	17.100	143.50
2	India	New Dehli	3.286	1252.00
3	China	Beijing	9.597	1357.00
4	South Africa	Pretoria	1.221	52.98

Matplotlib is a very popular Python library for data visualization. Like Pandas, it is not directly related to Machine Learning. It particularly comes in handy when a programmer wants to visualize the patterns in the data. It is a 2D plotting library used for creating 2D graphs and plots. A module named pyplot makes it easy for programmers for plotting as it provides features to control line styles, font properties, formatting axes, etc. It provides various kinds of graphs and plots for data visualization, viz., histogram, error charts, bar chats, etc,

Python Code:

```
import matplotlib.pyplot as plt
import numpy as np

# Prepare the data
```

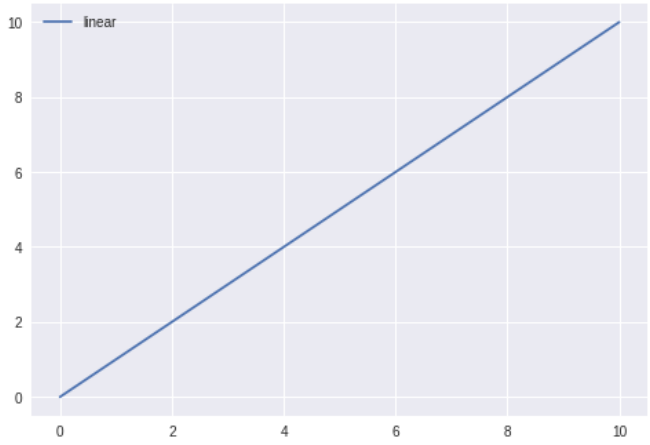
```
x = np.linspace(0, 10, 100)

# Plot the data
plt.plot(x, x, label ='linear')

# Add a legend
plt.legend()

# Show the plot
plt.show()
```

Output:



AIML project Using Python:

Chat Bot:

Create Standard Startup File
It is standard to create a startup file called **std-startup.xml** as the main entry point for loading AIML files. In this case we will create a basic file that matches one pattern and takes one action. We want to match the pattern **load aiml b**, and have it load our aiml brain in response. We will create the basic_chat.aiml file in a minute.

```
<aiml version="1.0.1" encoding="UTF-8">
   <!-- std-startup.xml -->

   <!-- Category is an atomic AIML unit -->
   <category>

      <!-- Pattern to match in user input -->
      <!-- If user enters "LOAD AIML B" -->
      <pattern>LOAD AIML B</pattern>

      <!-- Template is the response to the pattern -->
      <!-- This learn an aiml file -->
      <template>
         <learn>basic_chat.aiml</learn>
         <!-- You can add more aiml files here -->
         <!--<learn>more_aiml.aiml</learn>-->
      </template>

   </category>

</aiml>
```

Creating an AIML File

Above we created the AIML file that only handles one pattern, **load aiml b**. When we enter that command to the bot, it will try to load **basic_chat.aiml**. It won't work unless we actually create it. Here is what you can put inside **basic_chat.aiml**. We will match two basic patterns and respond.

```
<aiml version="1.0.1" encoding="UTF-8">
<!-- basic_chat.aiml -->

   <category>
      <pattern>HELLO</pattern>
      <template>
         Well, hello!
      </template>
   </category>

   <category>
```

```
    <pattern>WHAT ARE YOU</pattern>
    <template>
      I'm a bot, silly!
    </template>
  </category>

</aiml>
```

Random Responses

You can also add random responses like this. This one will respond randomly when it receives a message that starts with "One time I ". The ***** is a wildcard that matches anything.

```
<category>
  <pattern>ONE TIME I *</pattern>
  <template>
    <random>
      <li>Go on.</li>
      <li>How old are you?</li>
      <li>Be more specific.</li>
      <li>I did not know that.</li>
      <li>Are you telling the truth?</li>
      <li>I don't know what that means.</li>
      <li>Try to tell me that another way.</li>
      <li>Are you talking about an animal, vegetable or mineral?</li>
      <li>What is it?</li>
    </random>
  </template>
</category>
```

Use Existing AIML Files

It can be fun to write your own AIML files, but it can be a lot of work. I think it needs around 10,000 patterns before it starts to feel realistic. Fortunately, the ALICE foundation provides a number of AIML files for free. Browse the AIML files on the Alice Bot website. There was one floating around before called std-65-percent.xml that contained the most common 65% of phrases. There is also one that lets you play BlackJack with the bot.

Install Python AIML Module

So far, everything has been AIML XML files. All of that is important and will make up the brain of the bot, but it's just information right now. The bot needs to come to life. You could use any language to implement the AIML specification, but some nice person has already done that in Python.

PYTHON 3

the source code remains exactly the same. You still import the package as **aiml** but when installing it with pip you use the name **python-aiml**. The source code is available at https://github.com/paulovn/python-aiml.

pip install python-aiml

Simplest Python Program

This is the simplest program we can start with. It creates the aiml object, learns the startup file, and then loads the rest of the aiml files. After that, it is ready to chat, and we enter an infinite loop that will continue to prompt the user for a message. You will need to enter a pattern the bot recognizes. The patterns recognized depend on what AIML files you loaded.

We create the startup file as a separate entity so that we can add more aiml files to the bot later without having to modify any of the programs source code. We can just add more files to learn in the startup xml file.

```
import aiml

# Create the kernel and learn AIML files
kernel = aiml.Kernel()
kernel.learn("std-startup.xml")
kernel.respond("load aiml b")

# Press CTRL-C to break this loop
while True:
    print kernel.respond(raw_input("Enter your message >> "))
```

Speeding up Brain Load

When you start to have a lot of AIML files, it can take a long time to learn. This is where brain files come in. After the bot learns all the AIML files it can save its brain directly to a file which will drastically speed up load times on subsequent runs.

```
import aiml
import os
```

```
kernel = aiml.Kernel()

if os.path.isfile("bot_brain.brn"):
    kernel.bootstrap(brainFile = "bot_brain.brn")
else:
    kernel.bootstrap(learnFiles = "std-startup.xml", commands = "load aiml b")
    kernel.saveBrain("bot_brain.brn")

# kernel now ready for use
while True:
    print kernel.respond(raw_input("Enter your message >> "))
```

Reloading AIML While Running

You can send the load message to the bot while it is running and it will reload the AIML files. Keep in mind that if you are using the brain method as it is written above, reloading it on the fly will not save the new changes to the brain. You will either need to delete the brain file so it rebuilds on the next startup, or you will need to modify the code so that it saves the brain at some point after reloading. See the next section on creating Python commands for the bot to do that.

```
load aiml b
```

Adding Python Commands

If you want to give your bot some special commands that run Python functions, then you should capture the input message to the bot and process it before sending it to **kernel.respond()**. In the example above we are getting user input from **raw_input**. We could get our input from anywhere though. Perhaps a TCP socket, or a voice-to-text source. Process the message before it goes through AIML. You may want to skip the AIML processing on certain messages.

```
while True:
    message = raw_input("Enter your message to the bot: ")
    if message == "quit":
        exit()
    elif message == "save":
        kernel.saveBrain("bot_brain.brn")
    else:
        bot_response = kernel.respond(message)
        # Do something with bot_response
```

Sessions and Predicates

By specifying a session, the AIML can tailor different conversations to different people. For example, if one person tells the bot their name is Alice, and the other person tells the bot their name is Bob, the bot can differentiate the people. To specify which session you are using you pass it as a second parameter to **respond()**.

```
sessionId = 12345
kernel.respond(raw_input(">>>"), sessionId)
```

```
sessionId = 12345

# Get session info as dictionary. Contains the input
# and output history as well as any predicates known
sessionData = kernel.getSessionData(sessionId)

# Each session ID needs to be a unique value
# The predicate name is the name of something/someone
# that the bot knows about in your session with the bot
# The bot might know you as "Billy" and that your "dog" is named "Brandy"
kernel.setPredicate("dog", "Brandy", sessionId)
clients_dogs_name = kernel.getPredicate("dog", sessionId)

kernel.setBotPredicate("hometown", "127.0.0.1")
bot_hometown = kernel.getBotPredicate("hometown")
```

In the AIML we can set predicates using the set response in template.

```
<aiml version="1.0.1" encoding="UTF-8">
  <category>
    <pattern>MY DOGS NAME IS *</pattern>
    <template>
      That is interesting that you have a dog named <set
name="dog"><star/></set>
    </template>
  </category>
  <category>
    <pattern>WHAT IS MY DOGS NAME</pattern>
    <template>
      Your dog's name is <get name="dog"/>.
```

```
    </template>
  </category>
</aiml>
```

With the AIML above you could tell the bot:

My dogs name is Max

And the bot will respond with

That is interesting that you have a dog named Max

And if you ask the bot:

What is my dogs name?

The bot will respond:

Your dog's name is Max.