Namal Institute Mianwali
Computer Science Department
Artificial Intelligence
Fall 21

PROJECT :02

Train Your Dragon

Document:	AI Project	Date:	12-12-2021
Prepared by:	Jawad Ahmed (BSCS-2019-59) Sidrah Iqbal (BSCS-2019-27)	Prepared for:	Dr.Junaid
Document Details:	This document contains our understanding and explanation of classification problems or neural networks by experimenting different cases using train and validation data.		

Problem Statement:

- We have to design a model which takes an image of a hand written Roman Number as input and tells us which class of numbers it belongs to.
- It's a classification problem where the decision should be made on the basis of previous data.

Introduction:

- Brain is the most beautiful yet the most complex organ of human being. It is the brain that distinguishes us from other organisms. Have you ever wondered what makes Brain so powerful? Our brain consists of billions of neurons which work coherently and give us amazing capabilities. One of which is the capability to classify objects. Neuron is a single unit which combines with others and helps us make different classes. Humans can recognize which is a cat and which is a dog if a picture is shown. But can a machine guess it as a human does? Yes, it is possible through Artificial Intelligence. Artificial Intelligence is a branch of computer science where making machines intelligent is practised. AI is able to solve problems of classification before which Machines were not able to distinguish between two things and tell which belongs to which class.

.

1 Abstract / Overview of Project:

1.1 Context:

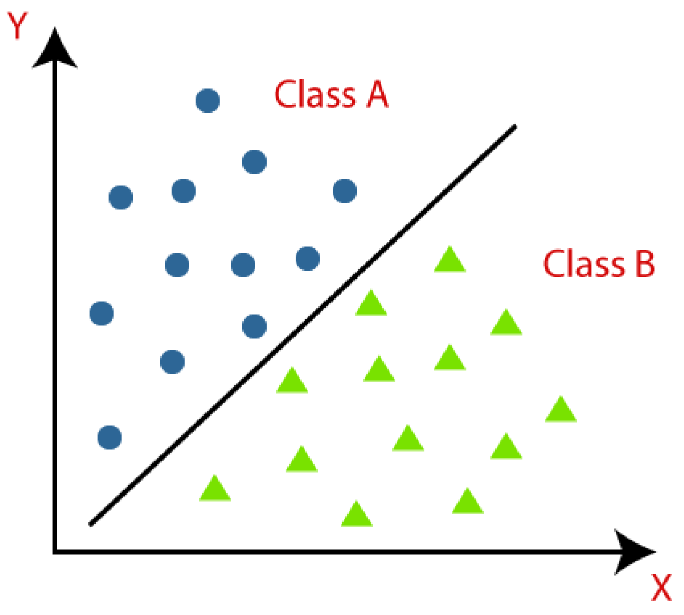
In the scope of this report, we will generally define the problem as such that we will train the machine in such a way that it will find the result that will be similar to the previous data, and we will accept a final result once we have either ran the classifier for some maximum number of iterations. This scenario clearly uses the classification algorithm which is based on following **Natural Phenomena.**

1.2 Natural Phenomena:

In nature nothing is perfectly the same. Trees and stars are not the same. But how can we say that they are not the same and more importantly how can we differentiate that this is a star and this is a tree. We humans are the best example to get inspiration from for solving classification problems. Naturally we are able to classify things. But this requires learning. Not by birth we have been embedded with this power of classification. All the way down the life human minds have been learning. In our childhood we might have mixed a cat with a dog but now we have trained our brain over hundreds of examples which makes our sense of recognition stronger than before. Neurons in our brain learned how a cat looks and similarly what the features of a dog are. Neurons differentiate things on the basis of features. Same neural network analogy could be adopted to help machines learn a concept.

1.3 Classification Theory:

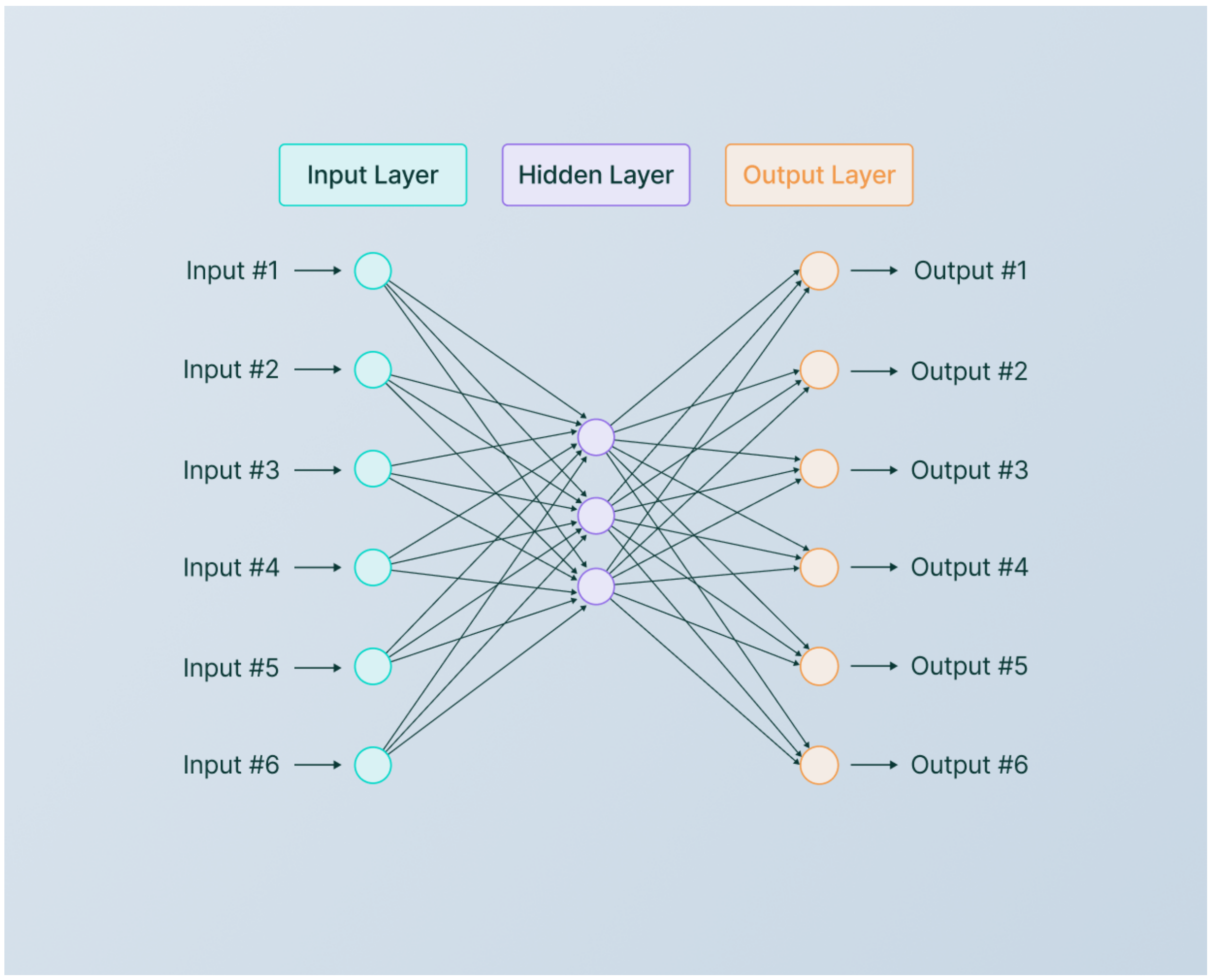
The theory of classification says objects can be divided into groups according to their similarities and differences. Two similar objects must belong to the same domain of object. Classification is to draw a border line between dissimilar objects in a way that boundary must hold all the objects belonging to the same domain. Classification requires a standard upon which other objects are judged and then classified into groups. This standard has to be learned from the data which is known as Learning General Concept.

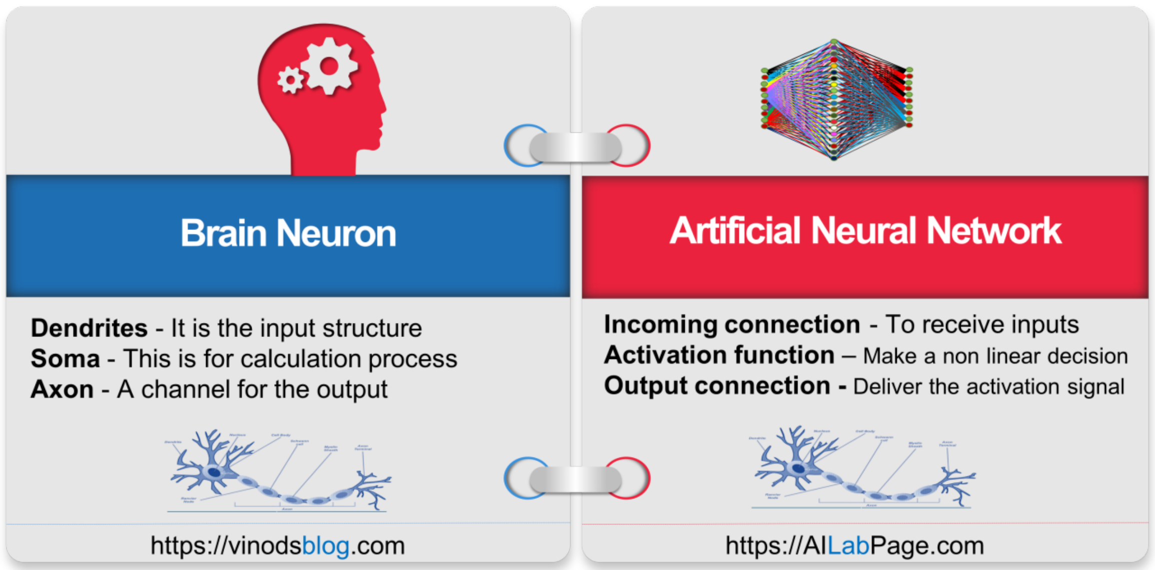
- 

    - Source: https://paperswithcode.com

1.4 What is a Neural Network?

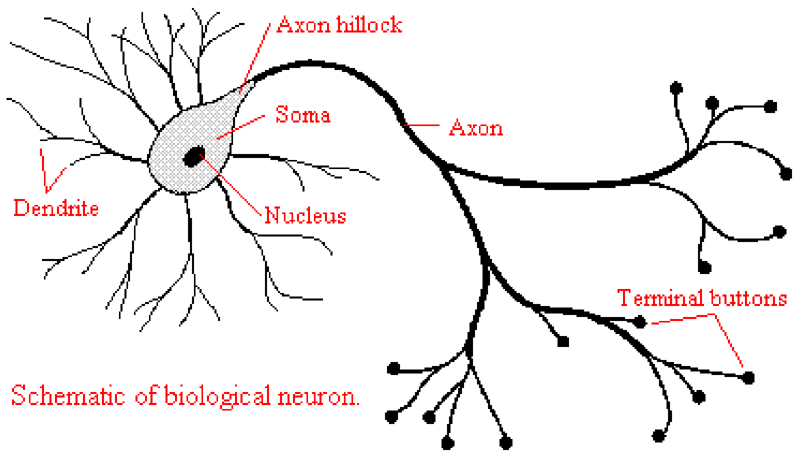
- Artificial Neural networks are computing systems inspired by the biological neural networks that constitute animal brains. As animal brains consist of billions of neurons in order to do classification and many other tasks.
- In the same way, an artificial neural network is inspired from the animal brains and it is based on a collection of connected units or nodes called artificial neurons, which loosely model the neurons in a biological brain.
- In artificial neural networks there is an input layer, hidden layer and output layer.
- Input layer is depending upon the data representation, some data may be in the form of digits or an array etc.
- A hidden layer in an artificial neural network is a layer in between input layers and output layers, where artificial neurons take in a set of weighted inputs and produce an output through an activation function.
- The output layer is responsible for producing the final result. There must always be one output layer in a neural network.
- Following diagram clearly shows the input, hidden and output layers.

- 
Input Layer
Hidden Layer
Output Layer
Input #1
Input #2
Input #3
Input #4
Input #5
Input #6
Output #1
Output #2
Output #3
Output #4
Output #5
Output #6

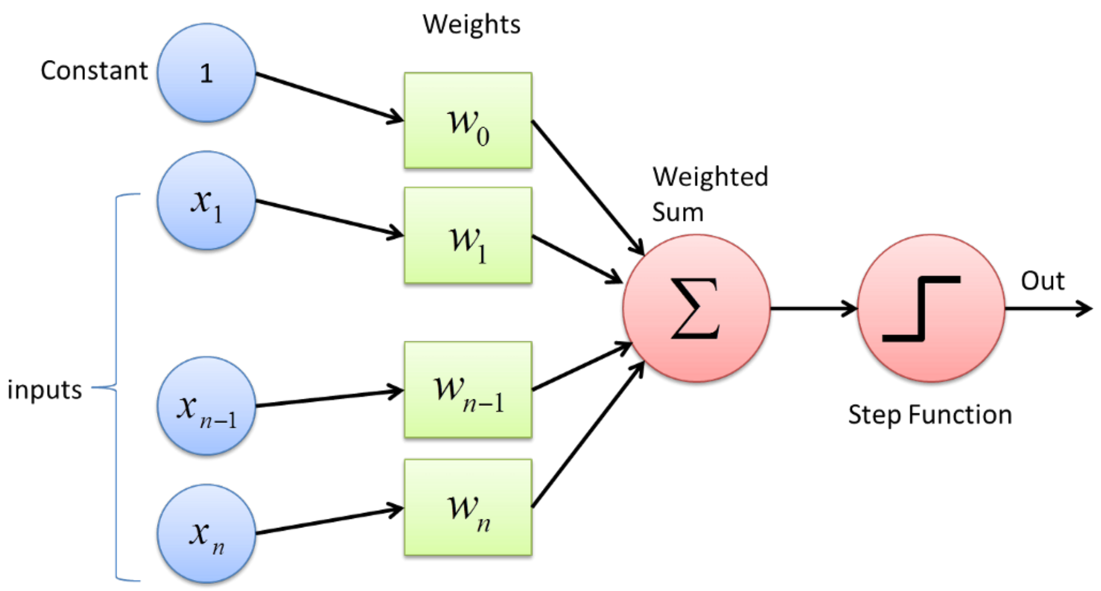

Brain Neuron
Dendrites - It is the input structure
Soma - This is for calculation process
Axon - A channel for the output
https://vinodsblog.com
Artificial Neural Network
Incoming connection - To receive inputs
Activation function – Make a non linear decision
Output connection - Deliver the activation signal
https://AILabPage.com

2 Computational Model:

Brain is not centralised. All the neurons work together. Output of a neuron could be input to another neuron. Our computational model for classification is derived from the biological drawing of a neuron. There are three basic parts of a neuron which are the nucleus, dendrite and terminals. Dendrites are used to fetch inputs when the process nucleus performs the function on Inputs and terminals pass out results.



Our computational model is derived from this biological neuron which acts in the same way. Artificial neural networks consist of the same basic units. It has inputs, it performs a dedicated operation on it and produces results. Number of inputs and functions performed vary according to the problem.



The above diagram shows a replica of a neuron it gets input from the data represented by x variables and w weights. By saying Learn a concept means find out the best possible weights for the training data which classifies it. The neuron space holds a function which performs combined operation on all inputs. In this figure the neuron calculates the dot product of x and w inputs which is then stepped up or stepped down by another active function to shrink the output of the neuron to a single integer. Step function also changes according to problem variation.

2.1 Characteristics of Neural Network :

- It is neurally implemented mathematical model
- It contains huge number of interconnected processing elements called neurons to do all operations
- Information stored in the neurons are basically the weighted linkage of neurons
- The input signals arrive at the processing elements through connections and connecting weights.
- It has the ability to learn , recall and generalise from the given data by suitable assignment and adjustment of weights.
- The collective behaviour of the neurons describes its computational power, and no single neuron carries specific information .

3 Implementation of Project:

3.1 Use of Language and libraries:

- This project is implemented in python 3.90. Python has powerful libraries to mimic neural networks. SciKit-learn is one of the libraries we used. It is a machine learning library which features classification, regression and clustering. Since our problem is classification related, we have used Multi Layer Perceptron (MLP) classifiers that come in SKlearn accompanied with helping libraries like numpy and matplotlib.

2.2 Sk-Learn MLPClassifier:

- MLP classifier is a supervised learning algorithm which means we train the perceptron to achieve a desired output. It is a non linear learning model when used with hidden layers. Major parameters of MLP classifier are:
  - **Hidden_layer_sizes:**
    - it sets size of layers as well as number of neurons in each layer. The first layer represents the input layer and the last one is the output layer and between them are considered hidden layers.
  - **Activation**:
    - An activation function in a neural network defines how the weighted sum of the input is transformed into an output from a node or nodes in a layer of the network.
    - We use the logistic activation function in our classifier because it is based on the sigmoid function which is a nonlinear function. And it is used for those models where we have to predict probabilities as an output.
  - **Solver**:
    - It is used for weight optimization.
    - The 'sgd' solver is an iterative method for optimising an objective function with suitable smoothness properties (e.g. differentiable or subdifferentiable).
  - **Learning_rate_init**:
    - It tells the difference to keep in the result of each iteration. We keep it minimum because bigger numbers can give a big jump

which eventually ends up losing the correct data which was correctly classified.

- **Random_state:**
  - It determines random number generation for weights and bias initialization.
- **Verbose** :
  - When its value is True it displays the result of each iteration and training loss.Training Loss represents a figure which does not help learning.
- **Max_iter:**
  - It represents total number of iterations
- **N_iter_no_change:**
  - MLP stops when a given number of iterations is reached without any improvement in loss.

2.3 Code:

2.3.1 Functions:

- **Read images:**
  - A function is defined to read all the images in the training data folder and validation folder. It returns a list of all images and count of images we have in the folder for each class.

```
def readImages(dirPath):
  all_images = list()
  imagesCount = []
  for (path, names, filenames) in sorted(os.walk(dirPath)):
    imagesCount.append(len(filenames))
    all_images += [os.path.join(path, file) for file in filenames]
  imagesCount = [i for i in imagesCount if i != 0]
  return all_images, imagesCount
```

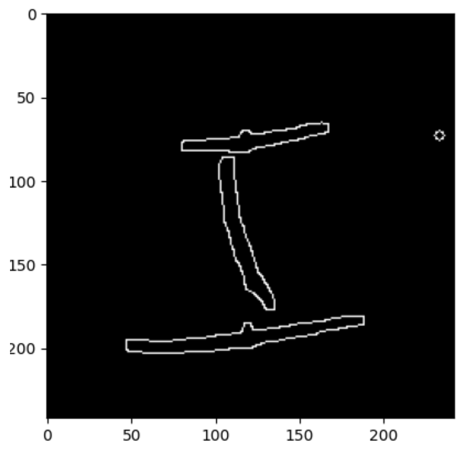
- **Feature Extraction:**
  - Another primary function helps extract features of all the images we read before from folders. Data from these features are then passed to the MLP classifier. It has two arguments, one is the list of images which it turns into an array and the other is the type of feature we want to extract. In this experimentation we have used 6 different types of features to be extracted. All are explained below:

1. **Black part:**
   - This feature returns the count of black portion in an image array.

```
feature == 'blackPart':
      imageRead = np.round(imageRead)
      count = np.count_nonzero(imageRead == 0)
      featureExtract.append(count)
```

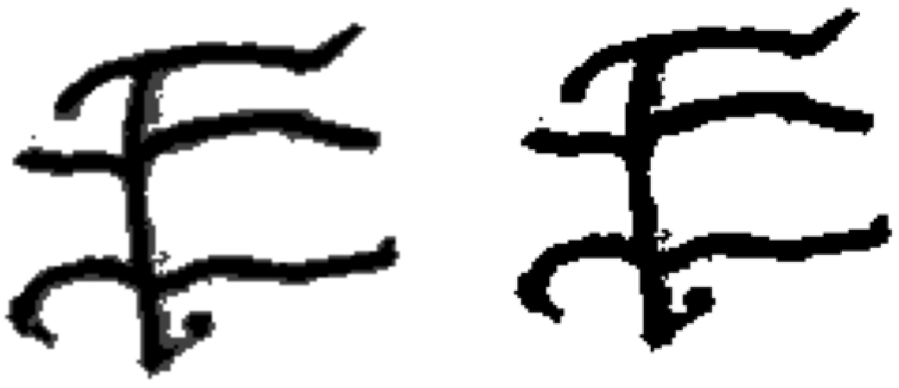
2. **Edge detection:**
    - This feature detects the edges of an image.

```
feature == 'edgeDetection':
        count = 0
        img = canny(imageRead)
        count = np.count_nonzero(imageRead == 0)
        featureExtract.append(count)
```



3. **Image threshold:**
    - This feature darkens the image making it more distinct.

```
feature == 'Imgthreshold':
        threshol_val = threshold_mean(imageRead)
        threshold_img = imageRead>threshol_val
        data = np.count_nonzero(threshold_img == False)
        img_size = imageRead.shape
        count = (img_size[0]*img_size[1])-data
        featureExtract.append(data)
```



4. **Diagonal:**

   It gets the diagonal of an image array.

```
feature == 'diagonal':
        imageRead = np.resize(imageRead, (300, 300))
        left_diag=[imageRead[j][j] for j in range(len(imageRead))]
        right_diag=[imageRead[len(imageRead)-1-i][i] for i in
range(len(imageRead)-1,-1,-1)]
        # left_diag=np.mean(left_diag)
        # right_diag=np.mean(right_diag)
        mid_pixel = len(left_diag)//2
        new_img = small[mid_pixel-10:mid_pixel+10]
        featureExtract.append([new_img,right_diag])
```

5. **Centroid:**

   This feature extracts the central part of an image.

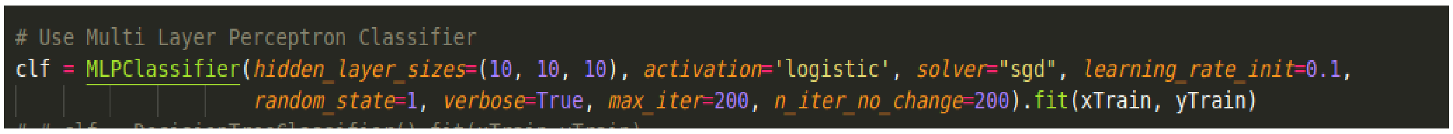
```
feature=='centroid':
        mid_pixel = len(imageRead)//2
        new_img = imageRead[mid_pixel-5:mid_pixel+5]
        train_img = []
        for i in new_img:
           train_img.append(i[(len(i)//2)-50:(len(i)//2)+50])
        train_img = np.array(train_img)
        train_img = np.reshape(train_img, (1, 1000))
        # print(train_img)
        featureExtract.append(train_img)
```

6. **Transformation:**

   This feature resizes the image and then gets its smaller part to transform an image into smaller parts.

```
feature=='transformation':
        # read in image
        image = np.resize(imageRead, (300, 300))
        # resize the image
        new_shape = (image.shape[0] // 2, image.shape[1] // 2)
        small = skimage.transform.resize(image=image, output_shape=new_shape)
        center = len(small)//2
        img = small[center-5:center+50, center+5:center+50]
        featureExtract.append(img)
```

- **Model Architecture:**
  Initially the following MLPClassifier setting is used, later on it is changed while experimenting for the better result.



```
# Use Multi Layer Perceptron Classifier
clf = MLPClassifier(hidden_layer_sizes=(10, 10, 10), activation='logistic', solver="sgd", learning_rate_init=0.1,
                    random_state=1, verbose=True, max_iter=200, n_iter_no_change=200).fit(xTrain, yTrain)
```

4 Experimentation:

4.1 Hypothesis No.1:

- Different feature extraction responds to different result accuracy because the classifier learns differently.
- If we train the MLP classifier on the basis of black count in image arrays, it may give some accurate result because somewhere in the data the darker part of train images may be the same as the darker part of validation images. Let's try it and see the result.

4.1.1 Experiment No.1:

- Keeping the parameters of the MLP classifier the same as shown in above model architecture.
- We see how the results differ with respect to different feature extraction.

- **1st Attempt:**

  In the first attempt we extract black part feature.

  - **Results**:

    Accuracy Of train_X with train_Y is: 14.000000000000002 %

    Accuracy Of test_X with test_Y is: 10.0 %

- **2nd Attempt:**

  In the 2nd attempt we extract edge detection features.

  - **Results**:

    Accuracy Of train_X with train_Y is: 15.0 %

    Accuracy Of test_X with test_Y is: 10.0 %

- **3rd Attempt:**

  In this attempt we chose to extract centroid feature

  - **Results**:

    Accuracy Of train_X with train_Y is: 73.0 %

    Accuracy Of test_X with test_Y is: 20.0 %

- **4th Attempt:**

In this attempt we chose to extract threshold feature

- **Results**:

  Accuracy Of train_X with train_Y is: 14.000000000000002 %

  Accuracy Of test_X with test_Y is: 10.0 %

- **Conclusion:**

  Centroid feature extraction improves the learning of classifiers.

4.2 Hypothesis No.2:

Size of hidden layers majorly affects the accuracy.

4.2.1 Experiment No.2:

In this experiment we manipulate hidden layer size parameters of MLP. keeping the extraction of features the same as centroid.

- **1st Attempt:**
  Size of hidden layers are kept as
  hidden_layer_sizes=(500, 250, 100)

  Here 500 represents the input size of the perceptron, 250 is the number of hidden layers and 100 is the output perceptron.

- **Results**:

  Accuracy Of train_X with train_Y is: 73.0 %

  Accuracy Of test_X with test_Y is: 20.0 %
- **2nd Attempt:**
  Size of hidden layers are kept as
  hidden_layer_sizes=(500, 300, 100)

  Here 500 represents the input size of the perceptron, 300 is the number of hidden layers and 100 is the output perceptron.

- **Results**:

  Accuracy Of train_X with train_Y is: 77.0 %

  Accuracy Of test_X with test_Y is: 19.0 %

- **Conclusion:**

  Increasing the number of hidden layers is able to improve the accuracy when compared with the desired output of data on which it is trained.

4.3 Hypothesis No.3:

Number of maximum iterations also affects learning of the classifier.

4.3.1 Experiment No.3:

Keeping other attributes like hidden layer size and learning rate same we change the max iteration parameter to observe the said hypothesis.

- **1st Attempt:**

Number of max iteration kept 800 which gives following result

  - **Results**:

    Accuracy Of train_X with train_Y is: 73.0 %

    Accuracy Of test_X with test_Y is: 20.0 %

- **2nd Attempt:**
  Number of max iteration is decreased to 500 which gives following result

  - **Results**:

    Accuracy Of train_X with train_Y is: 80.0 %

    Accuracy Of test_X with test_Y is: 22.0 %

- **3rd Attempt:**
  Number of max iteration is decreased to 300 which gives following result

  - **Results**:

    Accuracy Of train_X with train_Y is: 40.0 %

    Accuracy Of test_X with test_Y is: 23.0 %

- **Conclusion**

  A normal iteration size is required to reach optimal results. Either less or too large won't work.

4 Limitations and Recommendation:

- This project still needs improvement. The highest accuracy figure we have achieved is 20% for the validation data which is still a low number. It can be improved with more experimentation like: Merging all the features and then training the classifier other than training on individual features.

5 Conclusion:

- Artificial Neural Networks provide a new space to practice artificial intelligence. One of such kinds is also exhibited in our project documentation. With the help of MLP we are now able to make machine learning a generalised concept. This documentation also provides experiments done to check how the developed classifier performs when factor varies. In addition to this limitation and recommendation are also mentioned which could help improve the performance of MLP classifiers.