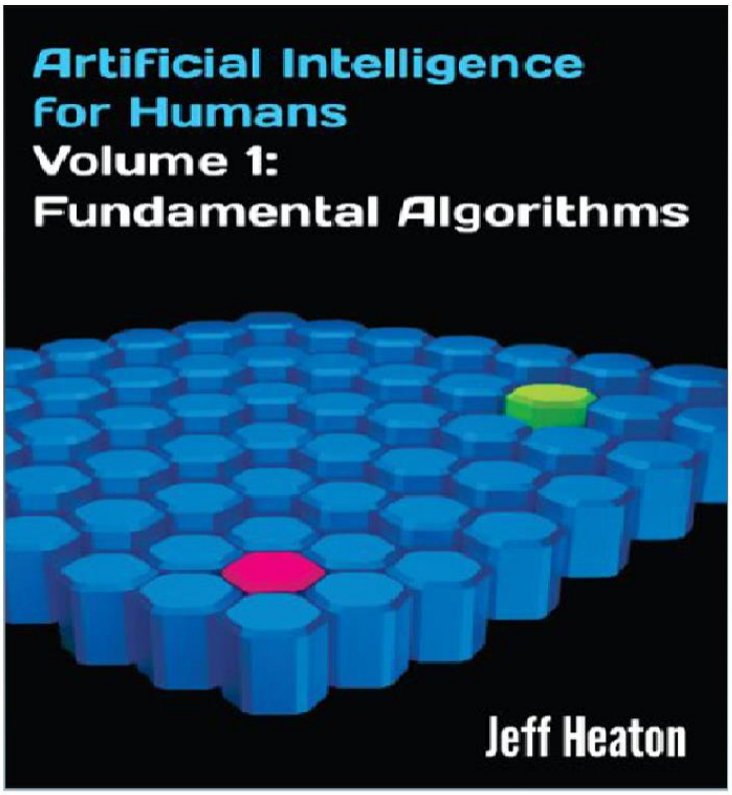
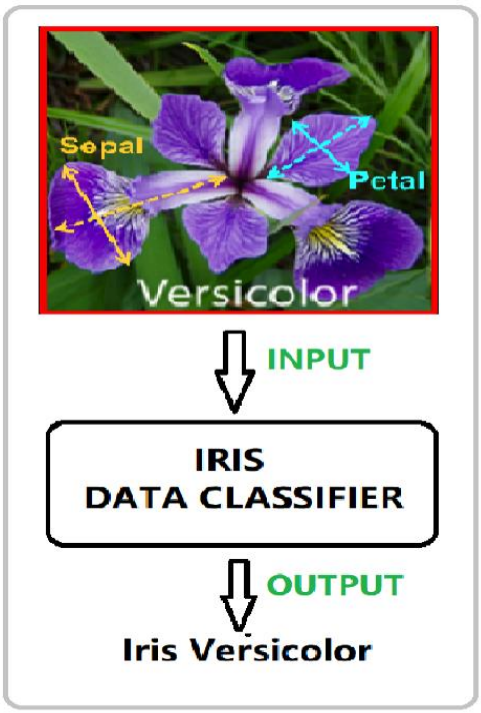
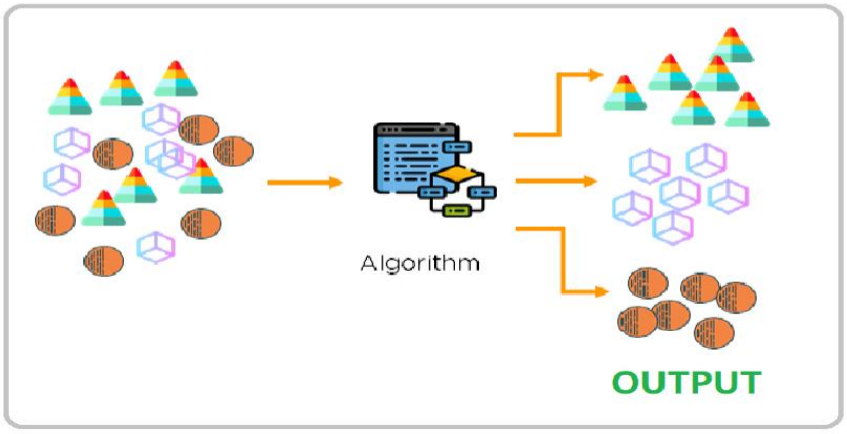
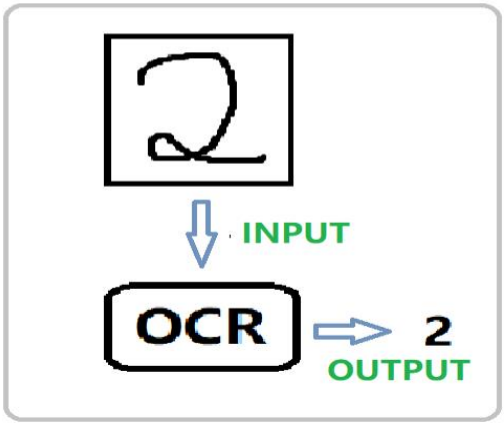








My take on AI

A comprehensive Summary of AIFH vol1 with further insights

Contents

What is AI?

Implementing various Human Intelligence features like understanding ability, thinking ability in a computer is AI. There are different ways an AI system can be implemented. One main feature which distinguishes a normal algorithm from AI is that most AI system have the ability to learn. Training is an important stage in development of an AI system. Any AI system can be trained continuously just like a human so that it becomes better and better at doing the task it is assigned to do.

The book "Artificial Intelligence for Humans vol.1" written by "Jeff Heaton" describes many algorithms to implement AI system. I have implemented AI system for the below mentioned "The IRIS dataset" problem based on algorithms described in that book. Also, I have tested & compared various algorithms and summarised my analysis of outputs & observations.

In this paper, we take the IRIS Dataset and see how various AI algorithms help in Classifying, Categorizing etc. Also, we see how these algorithm uses techniques like Supervised and unsupervised training to improve the outcome or accuracy of the algorithm. There are other techniques like normalization which is used in Machine Learning which we will have a brief review. I have also given various python code as reference for various algorithms.

The IRIS Dataset:

Iris is a common genus of flowers native to Northern America. There are 3 common species of iris namely the –

Iris setosa	Iris Versicolor	Iris Virginica
		

All 3 of the Irises look very similar, so classifying them would be a good problem we can give AI systems to solve.
Iris data set contains data of 150 iris flowers (50 from each variety) the data it contains is –
[sepal length, sepal width, petal length, petal width].

Our goal would be to classify the type of a given iris flower based on its parameter's sepal length, sepal width, petal length, petal width.

sepal length	sepal width	petal length	petal width	class
5.1	3.5	1.4	0.2	Iris-setosa
5.7	2.8	4.1	1.3	Iris-versicolor
6.3	3.3	6	2.5	Iris-virginica

Sample from The IRIS dataset –Programs\Iris Data Set.txt

Note: I have used the same iris data set to compare and test most of the implemented algorithms in this book.

Different Kinds of AI systems

In many ways an AI system can be implemented. Based on the type of input and output, and how it analyses the given problem we can come up with a few basic ways to model out problem.

- Data Classifiers
- Clustering
- Regression analysis

Data classifiers

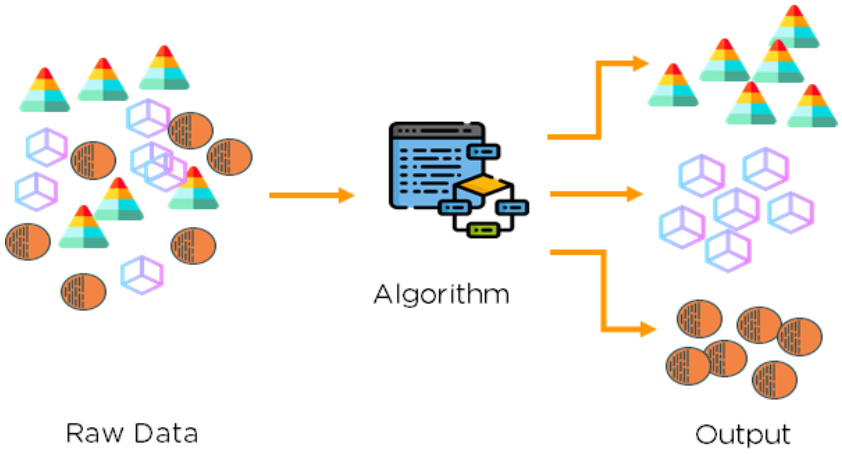
In this method, the data classification algorithm determines one of the pre-determined classes in which a given input data would fall into. To implement this algorithm the different available classes, nature of objects that are in each class, and the parameters that constitute a particular object (like for an iris flower the sepal & petal length & widths defines that flower) should be known clearly from the problem statement.

Example - classifying which type (versicolor, virginica, setosa) a given iris flower would belong to base on its parameters like sepal length, width & petal length & width.

Section Distance Metrices & Classification – shows various practical implementation of the Classification Algorithms

Clustering

Clustering is like an extension of data classification. Given a set of input objects it tries to group similar objects together. Usually it is also given the number of groups to be created.



Example - For a given set of different iris flowers' parameters the system tries to create groups with similar flowers as output

Section Clustering Algorithms – shows various practical implementation of the Classification Algorithms

Regression analysis

It aims to analyse the input data and provide an output value calculated from the input given. Unlike classification & clustering methods it does not deal with any classes. The output is in numerical form.

Example – Converting a temperature in degree Fahrenheit to obtain value in degree Celsius is also a form of linear regression. We give it a numerical input(degree Fahrenheit) and we obtain a numerical output(degree Celsius). This is a very basic example, we can use complex multivariate linear regression in artificial intelligence to compute various things like age of an oyster based on its number of shells, sex, weight etc. which we'll see in later sections.

Training an AI System

After creating our AI system, the next step is to train it so that it becomes better and better at doing the task it is assigned to do.

Based on the nature of training it can be classified as

Supervised Vs Unsupervised

a. **Supervised training** is when we explicitly give an AI system a definite output and train it according to it.
    - EX: classification algorithms - While training them we give them inputs and exactly specify the correct corresponding outputs. The whole structure of the algorithm is also well defined, we know the exact nature of its inputs and outputs

b. **Unsupervised training** is when we do not provide expected outputs to training algorithm.
    - Mostly the algorithm tries to figure this out by itself based on the input data.
    - Unsupervised training always occurs in clustering algorithms – we just give our algorithm a set of input data, we don't specify anything else. We also don't specify the nature of output; we just tell the algorithm to create groups.

Deterministic Vs Stochastic (Non-Deterministic)

a. **Deterministic algorithm** always performs exactly in the same way each time we run it. It performs like this because it does not involve any randomness (random numbers) in its training
    - For example, in Section *Distance Metrices & Classification* – we will see that the classifier always behaves in the same way as long as its input data members are the same

b. **Stochastic algorithm** uses random numbers for training, so it does not behave the same way each time we run it.
    - Most of the AI algorithms involves randomness in one form or the other, for example in Section *clustering algorithms* we will see that the first step involves initializing all groups randomly. As the first step itself involves randomness, so each time we run the algorithm we might get different outputs, even though input data members are the same

Any given AI system will have a set of input and output values. The input and output are represented as vectors in their own arrays. So, another important thing to be considered while modelling our problems is how to normalize the values so that they can be used in our AI system.

Data Normalization

Normalization is the process by which a set of real-life observations (inputs for an AI) is represented in the AI system. Usually the normalized values are always fit within a specific range (normalized low & high).

There are different normalization methods for different kinds of data:

1. Qualitative data
    a. Nominal data
    b. Ordinal data
2. Quantitative data

Qualitative data

Often any real-world problem we are trying to model will not always have numerical input and output, especially in data classification method we have different classes whose names might not be in numerical form. That is to classify Iris flowers, the output should be the type of the flower (versicolor, virginica or setosa) which is non numerical (Qualitative data). It is necessary to convert them and use them as numerical data, as only then we would be able to perform further computation on them with ease.

a) Nominal observations

Qualitative data in which no comparative (> or <) relationship can be established is called nominal data. The example of Iris classes specified above is an example for nominal data. We cannot tell weather Iris Versicolor > Iris Virginica. We can only establish equal to or not equal to relationship with nominal observations.

One of n encoding

-This effectively works for normalizing nominal observations mainly in data classification method. For modelling or representing the classes we create a vector with the n elements (n is the number of possible classes), such that each element refers to a class. So, while referring to a class only the element that refers to it is set to a high value such as 1 and all others that refer to different classes are set to a low value such as 0 or -1.

So, by just looking at which element of our encoded vector has a high value we can tell which class it is representing.

Example - encoding the Iris varieties:

Variety	Output
Iris setosa	[1,0,0]
Iris versicolor	[0,1,0]
Iris virginica	[0,0,1]

So, if our output vector is [0,1,0] we know our program is referring to Iris versicolor.

b) Ordinal observations

Qualitative data in which both comparative (> or <) relationship and equal to / not equal to relationships can be established are called Ordinal data. Example: Kindergarten, First Grade, Second Grade. All these are qualitative data. We can establish comparative relationships like we know that First Grade is greater than kindergarten. And we can always also establish equal to / not equal to relationships.

Simple One-of-n encoding can be used to normalize the ordinal data too, but the order of observation will be lost (That is, it will not retain comparative relationship)

Ordinal Data Normalization

A whole number is assigned to each of the classes. E.g., Kindergarten - 0, First Grade -1, Second Grade - 2

In this we can calculate the percentage completion of education by taking the whole number of a class and dividing it by the number of classes, then the result can be fit into any numerical limit we want.

$$f(x) = \frac{(nH - nL)x}{N} + nL$$

f(x) – gives the normalized output for given input's class number x
N – is the total number of classes
nH, nL – are the normalized output range High, Low respectively (usually it is 1, 0 respectively)

Ordinal Data Denormalization

We can easily calculate the number assigned to each class from its normalized value. From this class number we can easily determine, which class it refers to.

$$f(x) = \frac{N(x - nL)}{nH - nL}$$

f(x) – gives us the denormalized class number for input normalized value x
N – is the total Number of classes
nH, nL – are normalization range

Ordinal normalization program example – Programs\Normalization\Ordinal_normalization-EX.py

Quantitative data

Quantitative data is always numeric, so it is not always necessary to normalize them. However, it is a good idea to make the values fall with in specific range (normalization range), so that further comparison and computation will be simple.

Quantitative data Normalization

To convert a real-life observation of a given range into a normalized range the following formula can be used.

$$f(x) = \frac{(x - dL)(nH - nL)}{(dH - dL)} + nL$$

f(x) – gives the normalized value for the given input observation x
dH, dL – are input data's high and low range respectively
nH, nL – are normalization range

Quantitative data Denormalization

Getting back the original value from the normalized value is also possible if we know the real life & normalized ranges.

$$f(x) = \frac{(dL - dH)x - (nH.dL) + dH.nL}{(nL - nH)}$$

f(x) – gives us the output denormalized value for input normalized value x
dH, dL – are input data's high and low range respectively
nH, nL – are normalization range

Quantitative normalization program example: Programs\Normalization\Quantitative_Normalization-Ex.py

Implementation of AI Systems

Distance Metrices & Classification

Distance can be used to measure the degree of separation or similarity between 2 vectors (typically is a 1D array with certain values).
In our Iris data classification problem, an input for the classifier is a vector containing iris flower's sepal length, sepal width, petal length, petal width. So, by finding the distance between 2 flower vectors we can find how similar they are, so this in turn can be used for classification.

Ways to calculate vector distance:

1)Euclidean distance

The distance would be equal to the square root of sum of squares of differences of corresponding elements of vectors. Suppose the vectors are representable in a 2D graph format the Euclidean distance would-be straight-line path connecting both the vectors.

$$d(p,q) = \sqrt{\sum_{i=1}^{n}(q_i - p_i)^2}$$

d (p, q) – gives us the Euclidean distance between point(vectors) p & q

2)Manhattan distance (taxicab distance)

The distance would be equal to the sum of absolute value of the differences of corresponding elements of vectors.
If vectors are representable in 2D graph format, then the Manhattan distance would be the sum of their vertical (y axis) and horizontal (x-axis) distances.

$$d(p,q) = \sum_{i=1}^{n}|q_i - p_i|$$

d (p, q) – gives us the Manhattan distance between point(vectors) p & q

3)Chebyshev distance (chess board distance)

The distance would be equal to highest one of the absolute values of differences of corresponding elements of vectors. So, it only considers of 1 dimension of the vectors which have the highest difference, unlike Manhattan & Euclidean it does not depend on all dimensions.

$$d(p,q) = \max_i(|q_i - p_i|)$$

d (p, q) – gives us the Chebyshev distance between point(vectors) p & q

distance metrices programs – Programs\Distance Metrices\distance_algorithms.py

The iris data classifier - A simple Distance Metrices data classifier

I have developed an AI system to identify the flower species type for a given Iris flower (as per the problem statement described above). For implementing this I have used simple Data classifier algorithm based on different distance metrices discussed above.

How does it work

1) **Find Ideal Flower Vectors**: Take all the flowers of a particular species from iris data set in vector form and find an average vector, this will be the ideal flower vector for that particular species. We find the ideal vectors for all 3 species.
2) Then we can easily classify which class (species) a given flower would fall into by just calculating distance between the given flower vector and each of ideal flower vectors, which ever ideal vector results in lowest distance will be most similar flower species for the given flower vector.

Iris data classifier program – Programs\Distance Metrices\Iris data classifier(Distance metrices based)\iris_data_classifier.py

Program output analysis

The IRIS data classifier has been run using different distance metrices and combination of those (when using multiple distance metrices, the final output distance is equal to the sum of different metrices). The accuracy of the classifier is calculated based on how many of the iris flowers in iris input data set it classifies correctly.

Distance Metrices algorithm	Score	accuracy
Euclidean	139/150	92.6%
Chebyshev	137/150	91.3%
Manhattan	138/150	92%
Euclidean + Chebyshev + Manhattan	141/150	94%
Manhattan + Chebyshev	140/150	93.3%
Euclidean + Manhattan	141/150	94%
Euclidean + Chebyshev	138/150	92%

Observation

- When using a single distance metrices method alone it is clear that Euclidean distance works the best followed by Manhattan and then Chebyshev
- If we analyse the input iris data set, we can see that petal length is a factor which is different for each species, so pretty much just by comparing the differences between petal length alone we can classify which group it belongs to. That is why Chebyshev has got quite a good accuracy of 91.3% even though it only operates on 1 dimension. But just comparing petal length does not always work, some flowers are abnormally huge, so we need to relate other dimensions too

while classifying, that is why the Euclidean and Manhattan have indeed performed quite nicely and better than Chebyshev.

- When we combine different distances metrices methods for classifying, it is clear that Euclidean + Manhattan is the best choice.
  Best distance formula used (Euclidean + Manhattan):

$$d(p,q) = Euclidean(p,q) + Manhattan(p,q)$$

$$d(p,q) = \sqrt{\sum_{i=1}^{n}(q_i - p_i)^2} + \sum_{i=1}^{n}|q_i - p_i|$$

Additional Ideas

My hypothesis - distance between an average Iris Virginica & Versicolor vector is always less than the distance between Virginica & setosa or Versicolor & setosa. So, it is possible that Virginica and versicolor have a closer evolutionary origin.

Distance between (average):	Euclidean distance
Iris Setosa <> Iris Versicolor	3.2
Iris Setosa <> Iris Virginica	4.75
Iris Versicolor <> Iris Virginica	1.62

Further Analysis

The Iris flowers are found in different parts of united states.

Flower type	Climatic condition
Virginica	wet ditches, swamps
Versicolor	wet, wet mesic
Setosa	Colder, dry regions

We can see that Versicolor & virginica grow in similar climatic conditions so it is possible that they might have evolved more similar genes to adapt to that environment better.

Beauty of nature & distance algorithms – It is really amazing to see that by just comparing flower dimensions of different species using distance algorithm we could establish a higher-level relationship like common growing conditions.

OCR - Optical Character recognition (Distance Metrices Based)

OCR is a very popular application of AI. For a given picture of character drawn by human the OCR application tries to identify the character.
This problem can also be approached using distance metrices. It basically works by comparing a given image to a list of known images. To do this we have to first convert an image to a vector format (array). If we accomplish this then we can create a database of pre drawn characters in vector form along with their character names (These are ideal vectors) Then the OCR algorithm can compare a given drawn character with each ideal vector in database and the least distance vector would give the character drawn by user.

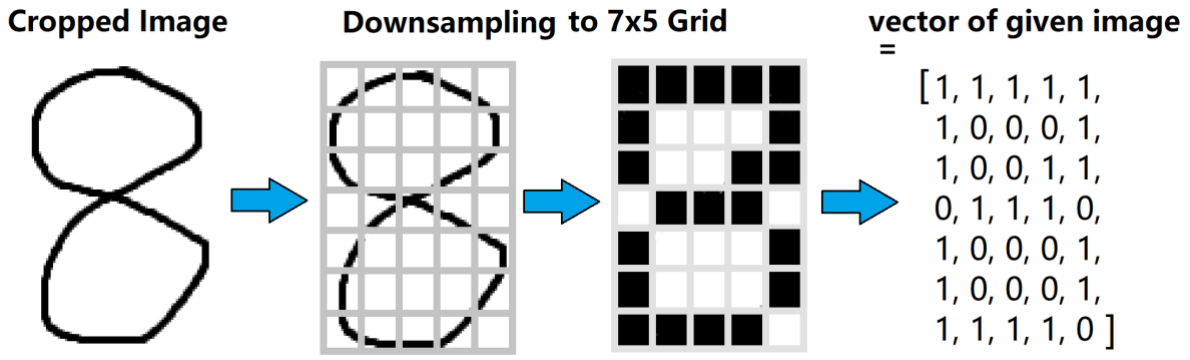
OCR implementation using distance metrices

Step 1 - Cropping the drawn character

- First, we crop only the drawn image from the entire given image. This helps the OCR concentrate only on the drawn image and factor out all white undrawn parts

Step 2 - Down sampling character

- We then try to reduce the resolution of cropped image to a fixed resolution. Ideally, we do this by superimposing a grid (like 7*5 grid) on cropped image. Even if 1 line of image passes through a cell of the grid then that cell is given a high value.
- This step makes sure a drawn image of any size can be converted to a fixed sized vector as only then we would be able to compare it with other vectors.
- Decreasing resolution of image makes it way easier for our distance metrices algorithm to process it.
- This will automatically give same output for similar images with minor changes, thus additionally helps our distance metrices algorithm to identify the given character image



Step 3 – Storing image as Vector

- As down sampled image is in the form of a grid it can directly be used as a vector by joining each row together. Ex: a 7x 5 grid would give a vector with 35 elements

Step 4 – Finding what character vector of given image resembles

- In our character database each character has its own ideal vector, this ideal vector is derived from the drawn image of that particular character from our dataset
- We find distance between given image's vector and vector of each character in our data base, which ever character's vector results least distance is more likely to be similar to our given image, so thus that character is the one present in our given image

OCRv1 - Program analysis

It is specifically written to recognise only numbers (0-9). All images are of greyscale bitmap format(.bmp), this makes it easier to convert the image into a 2D array using PIL & NumPy libraries. All input images are drawn using MS paint in black colour only. By default, the program considers the bmp image at "data/Character_bitmap/draw_pad_1.bmp" as given image. All other images part of image data base is present in "data/Character_bitmap". OCRv1 only uses 'a' image for all numbers. So, in total its image database has only 10 images (one for each character).

To run the program the user has to erase the image in draw_pad_1.bmp and redraw any number using black brush, save it and them run the program. Or simply give the file name of another bitmap image while calling the recognise() function.

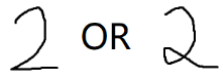
Output: the program prints the down sampled image along with what character it has identified in given input image

OCR version 1 program - *Programs\Distance Metrices Distance Metrices\Optical Character Recoganiser(Distance metrices based)\OCRv1.py*

OCRv2 – program analysis

How to make it better?

By increasing the ideal vectors database of predefined character images, the accuracy of identifying the given character image can be increased. We can add character images of more different ways to write a character to our character dataset, so thus our OCR will have more images to compare to and identify. For example: we know that people usually write the character '2' in 2 different ways:



If we add both the ways of writing '2' to our data base then whenever given image similar to either one of them, it will be recognized

The OCRv2.py can hold multiple images for each character in its database making it way more accurate All other functioning of OCRv2 is same as v1. The OCRv2 has a total of 38 images in its image database.

What is the catch?

adding multiple images has made OCRv2 better at recognizing images, but at the same time it also slows it down as more the number of images more time the OCR will take to compare given image to each image. Having 38 images OCRv2 is (38/10) >> 3.8 times slower than OCRv1.

OCR version 2 program - Programs\Distance Metrices Distance Metrices\Optical Character Recoganiser(Distance metrices based)\OCRv2.py

OCRv3- program analysis

Making it faster?

So, to make the OCR more accurate and at the same time to maintain speed -We can combine multiple images' vectors there for a particular character into 1 image vector. We do this by computing an

average image vector of all the images present for a particular character. So finally, all multiple images there for a character are combined and at the end a total of 38 images in database is reduced to only a 10 average image vectors. So, our new database in OCRv3 takes into account of all elements of 38 images thus making our OCR as accurate as V2 and at the same keeping it as fast as V1.

OCR version 3 faster program - Programs\Distance Metrices Distance Metrices\Optical Character Recoganiser(Distance metrices based)\OCRv3.py

Clustering Algorithms

Clustering can be used to place similar items into groups. Some clustering algorithms automatically determine the optimal number of groups and place input data in their corresponding groups. Most other clustering algorithm require the user to give the number of groups to be created as input. The process of clustering a finite number of observations into a specified number of groups is called **NP-Hard (Non-Deterministic Polynomial time)**. Clustering is a type of **unsupervised training** as we do not explicitly specify the expected ideal output, we only give it the number of groups to be created.

K-Means – Clustering algorithm

This is a simple and effective clustering algorithm.

Step - 1 - We take given input items and divide them equally into number of specified groups
We randomly place input items in groups such that each group has equal number of input items.
Step - 2- Calculate (Update) Centroid for each group
Centroid is a point which is an imaginary center point for all observations in a group. We find centroid by deriving an average vector for all vectors in a group.
We find Centroid for each group.
Step - 3 - Reorganize items based on new Centroid.
We iterate through all items and compare their vector with the centroid vectors of each group and place them in the group whose centroid results in least distance with an object.
Step -4 - We repeat Step 2 & Step 3 again.
We repeat Step 2 & Step 3 till there is no need to reorganize items in Step 3 i.e. all items in the groups are closest to the centroid of that group.

Naturally what happens is as we iterate through steps 2 & 3, more similar object tends to be placed or get accumulated in each group. This happens as initially when we randomly group objects (in step 1) usually 1 group can have a greater number of certain kind of objects, another group with a greater number of another kind of object. So, the centroid of these groups tends to be slightly inclined towards 1 kind of object. then as we reorganize items similar object tend to get accumulated with similar centroids as we continue the centroid becomes more and more closer to 1 kind of object.

Iris data clustering program(K-mean) - Programs\K-Means Clustering (For Iris Data Set)\k_means1.2.py

Program output analysis

This program has been specifically written for clustering iris flower vectors.

The program is given the flower of vectors of flowers in iris data set and told to create 3 groups of 150 input flowers.
These are the output groups created by the clustering program

Group 1	Group 2	Group 3
50 setosa	36 Virginica 3 Versicolor	47 Versicolor 14 virginica

We can see that the clustering program has performed pretty well it has made clusters of same flowers. It has just made a few mistakes while placing virginica flowers
Here we can say that it is *88.6% accurate*, this is lower than the accuracy of our *iris data classifier(distance based)* but we should also realize that this clustering algorithm is **unsupervised** we gave it no information about how our output should look like but on the other hand in our data classifier we gave the average vectors.

Forgy clustering algorithm

This algorithm is very similar to K-means -Steps 2, 3 & 4 are exactly the same. Only the initial starting point of the algorithm is different.
First it creates empty groups with any random observation (item from given items) as its centroid. It's made sure that no observation is chosen again as the centroid. After that Step 3 is executed all items are arranged in groups based on lowest centroid distance then. Step 2 & Step 3 are repeated again and again as in k-means.

The problem with this algorithm is that sometimes it might choose observations of the same kind as centroid of different groups, this does not promote proper clustering of objects.

Reference programs

Reference programs: https://github.com/AAA-2003/AIFH-1.git

Zip file containing all reference programs: https://ayngaran2003.wixsite.com/projects/my-take-on-ai

- *Datasets*
  - *Sample from The IRIS dataset –*Programs\Iris Data Set.txt
  - *Hand drawn numerical dataset (for OCR) –* Programs\Distance Metrices\Optical Character Recogniser(Distance metrices based)\data\Character bitmap
- *Normalization*
  - *Ordinal normalization program example* Programs\Normalization\Ordinal_normalization-EX.py
  - *Quantitative normalization program example:* Programs\Normalization\Quantitative_Normalization-Ex.py
- *Distance metrices*
  - *distance metrices programs:* Programs\Distance Metrices\distance_algorithms.py
  - *Iris data classifier program:* Programs\Distance Metrices\Iris data classifier(Distance metrices based)\iris_data_classifier.py
  - *OCR program v1:* Programs\Distance Metrices\Optical Character Recogniser(Distance metrices based)\OCRv2.py
  - *OCR program v2(improved accuracy):* Programs\Distance Metrices Distance Metrices\Optical Character Recoganiser(Distance metrices based)\OCRv2.py
  - *OCR program v3(improves speed):* Programs\Distance Metrices Distance Metrices\Optical Character Recoganiser(Distance metrices based)\OCRv3.py
- *Clustering algorithm*
  - *Iris data clustering program(K-mean) -* Programs\K-Means Clustering (For Iris Data Set)\k_means1.2.py