**Implementation of a handwriting recognition AI algorithm using Hidden Markov Models (HMM), using Python in jupyter notebook.**

**Student Name: Gargi Darade.**

**Subject/Class: CMPS 451 Artificial Intelligence.**

#Import the required libraries

import cv2

import numpy as np

from PIL import Image

import matplotlib.pyplot as plt

import os

from collections import Counter

from hmmlearn import hmm

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

from sklearn.metrics import accuracy\_score

from colorama import init, Style

#define helper functions

#To use image sequences as input for a Hidden Markov Model (HMM),

#you need to transform the multi-dimensional

#image data into a 1D array or a sequence of vectors,

#which can be understood as a time series by the HMM.

#Convert the multi-dimensional data into a 1D array,

#where each element represents a pixel or feature vector.

#Python NumPy: Use the .flatten() or .reshape() methods.

#Keras/TensorFlow: Employ the Flatten() layer in your neural network architecture.

# Flatten image sequences for HMM input

def flatten\_image\_sequence(image\_sequence):

n\_samples, seq\_length, n\_features = image\_sequence.shape

return image\_sequence.reshape(n\_samples \* seq\_length, n\_features)

def resize\_image(image, new\_width, new\_height):

"""resize an image by keeping aspect ratio and atlest keep new width or height."""

#if not isinstance(image, np.ndarray):

# image = np.array(image)

#if image.dtype != np.uint8:

# image = image.astype(np.uint8)

width, height = image.size

#print("Original Image shape: W:"+str(width)+" H:"+str(height))

aspect\_ratio = width / height

if width > new\_width or height > new\_height:

if width / new\_width >= height / new\_height:

final\_width = new\_width

final\_height = int(new\_width / aspect\_ratio)

else:

final\_height = new\_height

final\_width = int(new\_height \* aspect\_ratio)

else:

final\_width = width

final\_height = height

resized\_image = image.resize((final\_width, final\_height))

width, height = resized\_image.size

#print("Resized Image shape: W:"+str(width)+" H:"+str(height))

return resized\_image

def pad\_image(image, target\_size, color=(0, 0, 0)):

"""Pads an image to a target size, maintaining aspect ratio."""

w, h = image.size[:2]

target\_w, target\_h = target\_size

#print("H:"+str(h)+" W:"+str(w)+" TH:"+str(target\_h)+" TW:"+str(target\_w))

# Calculate padding values

delta\_w = target\_w - w

delta\_h = target\_h - h

top, bottom = delta\_h // 2, delta\_h - (delta\_h // 2)

left, right = delta\_w // 2, delta\_w - (delta\_w // 2)

#print("top:"+str(top)+" bottom:"+str(bottom)+" left:"+str(left)+" right:"+str(right))

# Add padding using cv2.copyMakeBorder

padded\_image = cv2.copyMakeBorder(np.array(image), top, bottom, left, right, cv2.BORDER\_CONSTANT, value=color)

return padded\_image

def resize\_and\_pad\_image(image, new\_width, new\_height):

"""resize an image by keeping aspect ratio and atlest keep new width or height."""

"""than Pads an image to a target size, maintaining aspect ratio."""

resized\_image = resize\_image(image, new\_width, new\_height)

# Specify the target size (width, height)

target\_size = (new\_width, new\_height)

# Pad the image

resized\_padded\_image = pad\_image(resized\_image, target\_size)

return resized\_padded\_image

def group\_and\_count(data\_list):

"""Groups list elements and counts duplicates.

Args:

data\_list: The input list. eg. my\_list = ['a', 'b', 'a', 'c', 'c', 'a']

Returns:

A dictionary where keys are unique elements and values are their counts.

eg. Expected output: {'a': 3, 'b': 1, 'c': 2}

"""

return dict(Counter(data\_list))

def filter\_lables\_based\_on\_count(label\_count\_dict,count=2):

"""filter dictionary based on equal or greate than count.

Args:label\_count\_dict: The input dictionary.

count: Input min count

Returns:

A list of filtered keys."""

filtered\_list = []

for index, (key, value) in enumerate(DictLabelCount.items()):

#print(f"Index: {index}, Key: {key}, Value: {value}")

if value>=count :

filtered\_list.append (key)

return filtered\_list

def filter\_dataset\_based\_on\_lables(dir\_image\_dict,lables\_list):

"""filter dictionary based on list of lables.

Args:dir\_image\_dict: The input dictionary.

lables\_list: Input lable list

Returns:

A filtered dir and image dictionary."""

filteredlabledataset =[]

for i in range(len(dir\_image\_dict)-1):

image\_path = dir\_image\_dict[i][0]

image\_Label= dir\_image\_dict[i][1]

if image\_Label in lables\_list:

filteredlabledataset.append([image\_path, image\_Label])

return filteredlabledataset

def display\_grid\_of\_images(image\_paths, labels=None, col=5, row=5, figsize=(15,15),save=False,filename="temp.png",fixsize=False,H=32,W=128):

"""

Displays a grid of images, optionally with labels.

Args:

image\_paths: A list of paths to image files.

labels: An optional list of labels for the images (same length as image\_paths).

col: Number of columns in the grid.

row: Number of rows in the grid.

figsize: Tuple representing the size of the figure (width, height) in inches.

"""

if len(image\_paths) != row \* col:

raise ValueError("The number of images must be equal to the product of rows and columns.")

fig, axes = plt.subplots(row, col, figsize=figsize)

axes = axes.flatten()

for i, image\_path in enumerate(image\_paths):

try:

img = Image.open(image\_path)

if(fixsize):

img = resize\_and\_pad\_image(img, W, H)

axes[i].imshow(img)

axes[i].axis('off') # Turn off axis ticks and labels

if labels and i < len(labels):

axes[i].set\_title(labels[i])

except FileNotFoundError:

print(f"Warning: Image not found at {image\_path}")

# Handle missing image, e.g., display a placeholder or skip it

axes[i].imshow(np.zeros((100,100,3))) # Placeholder

axes[i].axis('off')

if labels and i < len(labels):

axes[i].set\_title(labels[i])

plt.tight\_layout() # Adjust subplot parameters for a tight layout

if(save):

plt.savefig(filename)

plt.show()

def display\_grid\_of\_images(predict\_images, predicted\_labels,org\_labels, col=5, row=5, figsize=(15,15),save=False,filename="temp.png"):

"""

Displays a grid of images, with labels.

"""

display\_n\_images= row \* col

fig, axes = plt.subplots(row, col, figsize=figsize)

axes = axes.flatten()

for i in range(display\_n\_images):

axes[i].imshow(predict\_images[i])

axes[i].axis('off') # Turn off axis ticks and labels

axes[i].set\_title(org\_labels[i] + ", Pred:"+predicted\_labels[i])

plt.tight\_layout() # Adjust subplot parameters for a tight layout

if(save):

plt.savefig(filename)

plt.show()

#### IAM\_Words dataset from Kaggle is downloaded and used in this implementation Only few images are used in this implementation

#This code preprocesses a dataset by reading and parsing a words.txt file

#and adding information to three variables: dataset, vocab, and max\_len.

#The dataset is a list of lists, each inner list containing a file path and label.

#The vocab is a set containing all the unique label characters. The max\_len is the maximum length of the labels.

dataset, vocab, max\_len = [], set(), 0

labelset=set()

labelsList = []

# Define dataset path

dataset\_path = ".\\"

assert os.path.exists(dataset\_path), "Dataset not found!"

#print("Dataset path is correct .")

current\_path = os.getcwd()

#print(current\_path)

# Correct path to words.txt inside iam\_words folder

transcriptions\_path = os.path.join(dataset\_path, "iam\_words", "words.txt")

print(transcriptions\_path)

# Read transcription file

with open(transcriptions\_path, "r") as f:

lines = f.readlines()

for line in lines:

if not line.startswith("#"): # Ignore comment lines

line\_split = line.strip().split(" ")

if line\_split[1] == "err":

continue

folder1 = line\_split[0][:3]

folder2 = line\_split[0][:8]

file\_name = line\_split[0] + ".png"

label = line\_split[-1].rstrip('\n')

#rel\_path = stow.join(dataset\_path, "words", folder1, folder2, file\_name)

rel\_path =os.path.join(dataset\_path, "iam\_words", folder1, folder2, file\_name)

#print(rel\_path)

#assert os.path.exists(rel\_path), "Dataset not found!"

if not os.path.exists(rel\_path):

continue

dataset.append([rel\_path, label])

vocab.update(list(label))

labelset.add(label)

labelsList.append (label)

max\_len = max(max\_len, len(label))

print("words.txt from iam\_words dataset parsed" )

#A dictionary where keys are unique elements and values are their counts.

#DictLabelCount = group\_and\_count(labelsList)

#filter\_min\_count\_lables= filter\_lables\_based\_on\_count(DictLabelCount,count=50)

#print("filter lables len:"+ str(len(filter\_min\_count\_lables)))

.\iam\_words\words.txt

words.txt from iam\_words dataset parsed

#### using only few hand writen words, given in the list, to train HMM, ['any','Peers','put','subject','tomorrow','into','should','now','while','knows']

# using only words given in the list to train HMM

filteredlabledataset =[]

filter\_given\_lables= ['any','Peers','put','subject','tomorrow','into','should','now','while','knows']

filteredlabledataset=filter\_dataset\_based\_on\_lables(dataset,filter\_given\_lables)

#print("filteredlabledataset.type:"+ str(type(filteredlabledataset)))

print("Number pairs/images in the filtered data set: ", len(filteredlabledataset))

#print("filter\_given\_lables.type:"+ str(type(filter\_given\_lables)))

print("Number words in filtered iam\_words data set: ", len(filter\_given\_lables))

print(filter\_given\_lables)

Number pairs/images in the filtered data set: 84

Number words in filtered iam\_words data set: 10

['any', 'Peers', 'put', 'subject', 'tomorrow', 'into', 'should', 'now', 'while', 'knows']

#### Display few original images from dataset

image\_paths=[]

image\_labels=[]

# show first 10 images

for i in range(10):

image\_path = filteredlabledataset[i][0]

image\_Label= filteredlabledataset[i][1]

image\_paths.append (image\_path)

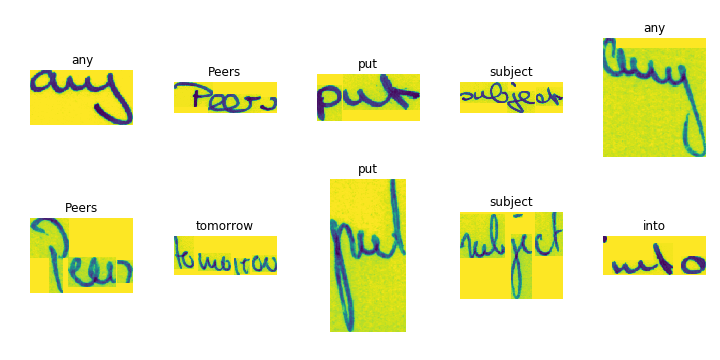
image\_labels.append (image\_Label)

#init()

print(Style.BRIGHT + " original images with lable from dataset." + Style.RESET\_ALL)

display\_grid\_of\_images(image\_paths, labels=image\_labels, col=5, row=2, figsize=(10,5),save=True,filename="OrgImages.png")

**original images with lable from dataset.**

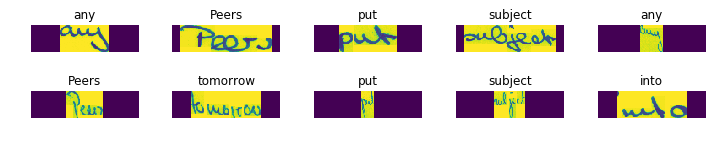


**Images are of different size, need to process them into fix size of 32x128 by maintaining aspect ratio, pad if needed**

print(Style.BRIGHT + " Processed images with lable from dataset." + Style.RESET\_ALL)

display\_grid\_of\_images(image\_paths, labels=image\_labels, col=5, row=2, figsize=(10,2),save=True,filename="ProcessImages.png",fixsize=True,H=32,W=128)

**Processed images with lable from dataset.**



#### create list of images and labels

images = []

labels = []

new\_width = 128

new\_height = 32

#for i in range(len(filteredMinCountdataset)-1):

for i in range(len(filteredlabledataset)-1):

image\_path = filteredlabledataset[i][0]

image\_Label= filteredlabledataset[i][1]

#image\_path = filteredMinCountdataset[i][0]

#image\_Label= filteredMinCountdataset[i][1]

img = Image.open(image\_path)

resized\_padded\_image = resize\_and\_pad\_image(img, new\_width, new\_height)

images.append (resized\_padded\_image)

labels.append (image\_Label)

#### convert images to N-dimensional array

#convert to N-dimensional array

images = np.array(images)

#labels = np.array(images)

print("images array size:"+str(images.size))

print("images array type:"+ str(type(images)))

print("images array shape:"+ str(images.shape))

print("images array, holding " + str(images.shape[0])+" images, each with a size of "+str(images.shape[1])+"x"+str(images.shape[2])+" pixels.")

#print("labels.size:"+str(labels.size))

print("labels type:"+ str(type(labels)))

print("labels .length:"+str(len(labels)))

#print("labels.shape:"+ str(labels.shape))

print("first label in list:"+ str(labels[0]))

print("first image in images array is:"+ str(labels[0]))

iamge1 = images[0]

print("image size:"+str(iamge1.size))

#print("image type:"+ str(type(iamge1)))

print("image shape:"+ str(iamge1.shape))

plt.imshow(iamge1)

plt.savefig("firstImage.png")

plt.show()

images array size:339968

images array type:<class 'numpy.ndarray'>

images array shape:(83, 32, 128)

images array, holding 83 images, each with a size of 32x128 pixels.

labels type:<class 'list'>

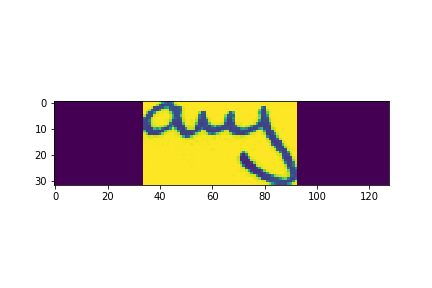
labels .length:83

first label in list:any

first image in images array is:any

image size:4096

image shape:(32, 128)



#### Split data into training and testing sets

# Split data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(images, labels, test\_size=0.2, random\_state=42)

print("X\_train.size:"+str(X\_train.size))

print("X\_train.type:"+ str(type(X\_train)))

print("X\_train.shape:"+ str(X\_train.shape))

print("y\_train.type:"+ str(type(y\_train)))

print("y\_train.size/len:"+str(len(y\_train)))

print("X\_test.size:"+str(X\_test.size))

print("X\_test.type:"+ str(type(X\_test)))

print("X\_test.shape:"+ str(X\_test.shape))

print("y\_test.type:"+ str(type(y\_test)))

print("y\_test.size/len:"+str(len(y\_test)))

X\_train.size:270336

X\_train.type:<class 'numpy.ndarray'>

X\_train.shape:(66, 32, 128)

y\_train.type:<class 'list'>

y\_train.size/len:66

X\_test.size:69632

X\_test.type:<class 'numpy.ndarray'>

X\_test.shape:(17, 32, 128)

y\_test.type:<class 'list'>

y\_test.size/len:17

#### To use image sequences as input for a Hidden Markov Model (HMM), you need to transform the multi-dimensional image data into a 1D array or a sequence of vectors, which can be understood as a time series by the HMM.

# Set parameters

n\_samples = images.shape[0] # total images

seq\_length = images.shape[1] # rows of image

n\_features = images.shape[2] # Assume flattened image patches/features, col of image

n\_states = 10

print("total images " + str(n\_samples)+" images, rows "+str(seq\_length)+", col/patches/features "+str(n\_features))

# Flatten the training and testing image sequences

X\_train\_flat = flatten\_image\_sequence(X\_train)

print("size of training, flatten image sequences:"+str(X\_train\_flat.size))

print("X\_train\_flat.type:"+ str(type(X\_train\_flat)))

print("X\_train\_flat.shape:"+ str(X\_train\_flat.shape))

X\_test\_flat = flatten\_image\_sequence(X\_test)

print("size of testing, flatten image sequences:"+str(X\_test\_flat.size))

print("X\_test\_flat.type:"+ str(type(X\_test\_flat)))

print("X\_test\_flat.shape:"+ str(X\_test\_flat.shape))

# Repeat labels to match flattened data

# same label for a rows of a single image

y\_train\_repeated = np.repeat(y\_train, seq\_length)

y\_test\_repeated = np.repeat(y\_test, seq\_length)

print("size of repeated labes to match flatten training data:"+str(y\_train\_repeated.size))

print("y\_train\_repeated.type:"+ str(type(y\_train\_repeated)))

print("y\_train\_repeated.shape:"+ str(y\_train\_repeated.shape))

print("size of repeated labes to match flatten testing data:"+str(y\_test\_repeated.size))

print("y\_train\_repeated.type:"+ str(type(y\_test\_repeated)))

print("y\_train\_repeated.shape:"+ str(y\_test\_repeated.shape))

total images 83 images, rows 32, col/patches/features 128

size of training, flatten image sequences:270336

X\_train\_flat.type:<class 'numpy.ndarray'>

X\_train\_flat.shape:(2112, 128)

size of testing, flatten image sequences:69632

X\_test\_flat.type:<class 'numpy.ndarray'>

X\_test\_flat.shape:(544, 128)

size of repeated labes to match flatten training data:2112

y\_train\_repeated.type:<class 'numpy.ndarray'>

y\_train\_repeated.shape:(2112,)

size of repeated labes to match flatten testing data:544

y\_train\_repeated.type:<class 'numpy.ndarray'>

y\_train\_repeated.shape:(544,)

#### Train the HMM for each class(words)

# Train the HMM for each class

models = {}

n\_states=10

for class\_label in np.unique(labels):

#print(class\_label)

#model = hmm.GaussianHMM(n\_components=n\_states, covariance\_type="diag", n\_iter=500)

#model = hmm.GaussianHMM(n\_components=n\_states, covariance\_type="diag", n\_iter=seq\_length)

model = hmm.GaussianHMM(n\_components=n\_states, covariance\_type="full", n\_iter=500)

# Filter training data for the current class

X\_train\_class = X\_train\_flat[y\_train\_repeated == class\_label]

if X\_train\_class.shape[0]!=0:

#print("X\_train\_class.type:"+ str(type(X\_train\_class)))

#print("X\_train\_class.size:"+str(X\_train\_class.size))

#print("X\_train\_class.shape:"+ str(X\_train\_class.shape))

#print("class\_label:"+ class\_label)

sample1 = X\_train\_class[0:seq\_length]

#plt.imshow(sample1)

#plt.show()

model.fit(X\_train\_class)

models[class\_label] = model

#### check accuracy of trained model on training data.

# Predict labels for the train set

predict\_images = []

predicted\_labels = []

for i in range(0, len(X\_train\_flat), seq\_length):

sample = X\_train\_flat[i:i+seq\_length]

#plt.imshow(sample)

#plt.show()

predict\_images.append(sample)

scores = {label: model.score(sample) for label, model in models.items()}

predicted\_label = max(scores, key=scores.get)

predicted\_labels.append(predicted\_label)

#print("predicted label:" + predicted\_label)

# Evaluate the model

accuracy = accuracy\_score(y\_train, predicted\_labels)

print("images:"+ str(len(predict\_images)))

print("predicted labels:"+ str(len(predicted\_labels)))

print("Lables:"+ str(len(y\_train)))

print(f"Accuracy: {accuracy}")

images:66

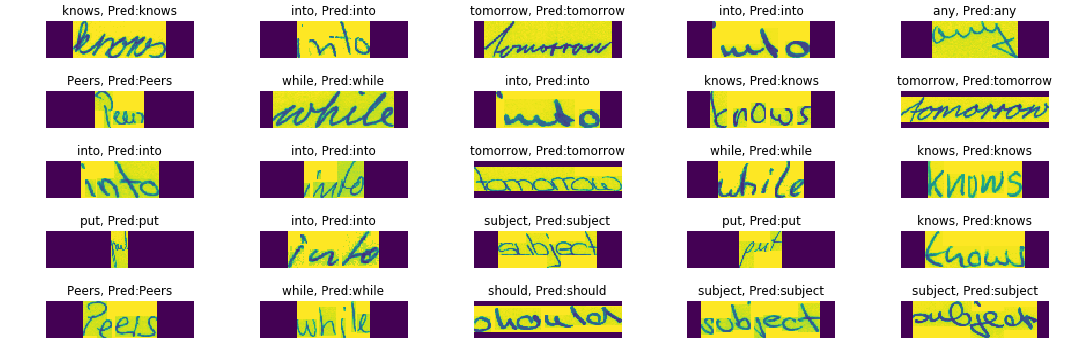
predicted labels:66

Lables:66

Accuracy: 0.9696969696969697

#### few predicted images from training data

display\_grid\_of\_images(predict\_images, predicted\_labels,y\_train, col=5, row=5, figsize=(15,5),save=True,filename="train.png")



#### check accuracy of trained model on testing data

# Predict labels for the test set

predict\_images = []

predicted\_labels = []

for i in range(0, len(X\_test\_flat), seq\_length):

sample = X\_test\_flat[i:i+seq\_length]

#plt.imshow(sample)

#plt.show()

predict\_images.append(sample)

scores = {label: model.score(sample) for label, model in models.items()}

predicted\_label = max(scores, key=scores.get)

predicted\_labels.append(predicted\_label)

#print("predicted label:" + predicted\_label)

# Evaluate the model

accuracy = accuracy\_score(y\_test, predicted\_labels)

print("images:"+ str(len(predict\_images)))

print("predicted labels:"+ str(len(predicted\_labels)))

print("labels:"+ str(len(y\_test)))

print(f"Accuracy: {accuracy}")

images:17

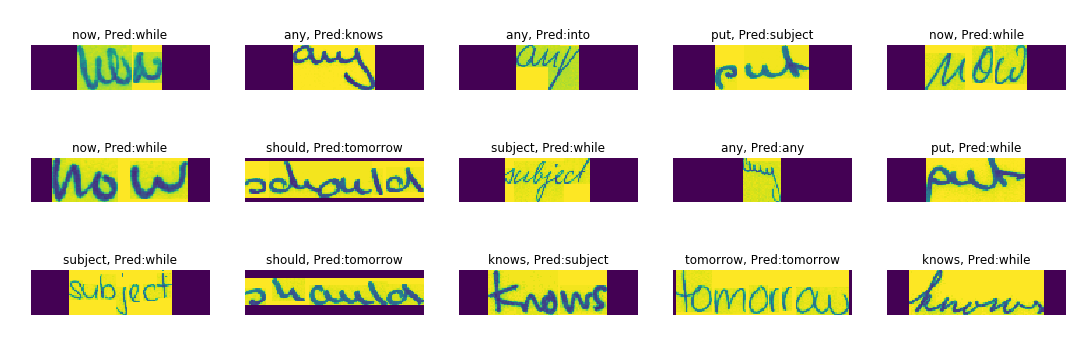
predicted labels:17

labels:17

Accuracy: 0.11764705882352941

#### few predicted images from testing data

display\_grid\_of\_images(predict\_images, predicted\_labels,y\_test, col=5, row=3, figsize=(15,5),save=True,filename="test.png")



### Conclusion:

#### Accuracy of trained model on training data is 97%, but it falls to 10% for test data. This can be improve by increasing number of copies of same word by same writer or different writer.