**Python Code (Traffic sign detection):**

## Importing

# Import the necessary modules for information control and visual portrayal

import pandas as pd

import seaborn as sns

from sklearn.preprocessing import LabelEncoder

import matplotlib as matplot

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

import cv2

import os

import matplotlib.pyplot as plt

from PIL import Image

import numpy as np

df\_meta = pd.read\_csv('Meta.csv', index\_col=None)

df\_data=pd.read\_csv('Train.csv', index\_col=None)

# We are bringing in the two information and its meta to do a superior information examination

# Data Inspection

# Check to check for any missing qualities in our informational index

df\_data.isnull().any()

print(df\_meta.head())

# The information is by all accounts neatly designed so no requirement for control. All the

shapes and hues are given numbers.

#But the way isn't required for information examination so we will evacuate it

df\_meta= df\_meta.drop(columns="Path")

print(df\_data.head())

# The information is by all accounts neatly designed so no requirement for control

#But the way isn't required for information examination so we will evacuate it

df\_data= df\_data.drop(columns="Path")

print(df\_meta.shape)

print(df\_meta.dtypes)

print(df\_data.shape)

print(df\_data.dtypes)

# Data Analysis

corr1 = df\_meta.corr()

corr1 = (corr1)

sns.heatmap(corr1,

xticklabels=corr1.columns.values,

yticklabels=corr1.columns.values)

corr1

# Drawing distribution graphs between classid. We can see how all the classes are distributed

in the data.

fig,ax = plt.subplots(figsize=(15,5))

ax = sns.countplot(df\_data['ClassId'])

plt.show()

# From the figure we can tell what number of various signs are there in database and what

number of signs are a similar sign however with various classes.Sign ID (by Ukrainian Traffic

Rule)

fig,ax = plt.subplots(figsize=(15,5))

hatchet = sns.countplot(df\_meta['SignId'])

plt.show()

f,ax = plt.subplots(figsize=(15,6))

hatchet = sns.scatterplot(x='SignId',y='ClassId',data=df\_meta)

plt.show()

# We can perceive how tallness and width shifts between classes

hatchet = sns.lineplot(x="ClassId", y="Height", data=df\_data)

hatchet = sns.lineplot(x="ClassId", y="Width", data=df\_data)

sns.pairplot(df\_data)

df\_meta.describe()

df\_data.describe()

# Data Preprocessing and KNN Implementation

#reading the quantity of classes from meta record

meta\_data = pd.read\_csv('Meta.csv')

meta\_shape = meta\_data.shape

no\_classes = meta\_shape[0]

# resizing picture to 20x20 size. likewise grayscaling the picture

# changing over all pictures to np exhibit and including marks

# took some assistance from web on the most proficient method to change over picture

informational collection into numpy cluster. As it was something new for me

# Take in note that you need to give the way in which your preparation information lies on way

factor underneath

# My information was in AML venture record in work area that is the reason I have given that

way

import cv2 # going to utilize cv2 as its quicker to resize and dim scale picture with it.

import os

train\_data=[]

train\_labels=[]

side = 20

for c in range(no\_classes) :

way = "/Users/client/Desktop/AML Project/train/{0}/".format(c) # ensure you give the right

root way

documents = os.listdir(path)

for document in records:

train\_image = cv2.imread(path+file)

image\_resized = cv2.resize(train\_image, (side, side), introduction = cv2.INTER\_AREA) #resizing

picture to 20x20 here

dim = cv2.cvtColor(image\_resized, cv2.COLOR\_BGR2GRAY) #Grayscaling picture

train\_data.append(np.array(gray))

train\_labels.append(c)

information = np.array(train\_data)

information = data.reshape((data.shape[0], 20\*20))

data\_scaled = data.astype(float)/255 # Normalizing information by partitioning with 255. so

information is somewhere in the range of 0 and 1

names = np.array(train\_labels)

le = LabelEncoder()

names = le.fit\_transform(labels) #adding marks to information

X\_train, X\_val, y\_train, y\_val = train\_test\_split(data\_scaled, names, test\_size=0.25,

random\_state=42) # spliting information into test and preparing. 25% test information

from sklearn.neighbors import KNeighborsClassifier

model = KNeighborsClassifier(n\_neighbors=3) #just default settings

model.fit(X\_train, y\_train)

# Since we need to process a huge number of pictures, this progression takes some time.

Around 4-5 minutes

y\_pred = model.predict(X\_val)

# Same for this step,it requires a significant stretch of time to finish because of huge dataset.

Around 4-5 mins.

model.score(X\_val, y\_val)

# we can see we got 87% accuracy,atleast in my PC.

#I tried it with n\_neighbors=5 and the exactness appeared to drop to 83%

# ### All models were tested Comparision between Models

# Accuracies are as follows:

# 1. Decition trees:

# entropy:0.7870039783739672

# gini: 0.7749668468836071

# with max depth=20:0.7428338263796797

# 2. Logistic Regression:

# Accuracy: 0.8894783377541998

# 3. Support Vector Machines

# Mean SVM Cross-validation Score: 0.9304895438545735

# 4. Random Forest:

# Mean RF Cross-validation Score: 0.9072640638631736

# 5. Ensemble Learning:

# KNearest Neighbours : 0.839

# Classification Tree : 0.775

# Voting Classifier: 0.911

# Logistic Regression : 0.865

# 6. Bagging Method:

# Accuracy of Bagging Classifier: 0.137

# 7. Boosting Method:

# Test set accuracy: 0.130

# OOB accuracy: 0.130

# 8. KNN:

# Accuracy score:0.8695297357951648

# From the above results it is evident that SVM has the most accurate results on this data set

on second ensembling on the third random forest and so on. The least accurate result with a

lot of error is boosting and bagging methods.