**Mammography Image Classification Using Naive Bayes and Ensemble Methods**

**Model Guide for naivebayes.py**

1. **Starting with Importing the required libraries**

“import time

…

import numpy as np”

1. **Transforming section**

“transform = transforms.Compose([

transforms.Resize((224, 224)),

transforms.ToTensor(),

])”

* In this section, we are resizing the images. Having a consistent image size is efficient in terms of computing power, since we are handling large data. Size of 224x224 pixels is often used in this area as we researched.

1. **Dataset class section**

class MammographyDataset(Dataset):

def \_init\_(self, data\_dir, data\_transform=None):

…

def \_len\_(self):

…

def transform(self, image\_files):

…

def labels(self):

…”

* In this section, we are listing all the files we have in our dataset. Following that, we return the number of images using the “\_len\_” section. “transform” function is used to load and transform the images, fitting them in arrays. In the last section, we are getting an image, executing the work required and we get the label from its name. Extra information can be found in code comments.

1. **Loading dataset, train and test loads**

“dataset = MammographyDataset(data\_dir='D:/jpeg', data\_transform=transform)

print(f'Calculations started...')

counter\_start = time.time()

cpu\_start = psutil.cpu\_percent(interval=1)

memory\_start = psutil.virtual\_memory()

X = dataset.transform(dataset.image\_files)

y = dataset.labels()

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

* Firstly, we are assigning our dataset variable to the actual dataset we have typing its directory. Then printing out that our main calculations are started using a print function. The start time also is recorded to measure the runtime, as well as initial CPU and memory usage.
* We are using %20 of the data for testing. As for the “random\_state” section, we decided to use a fixed number to ensure that we get healthy results comparing the outcomes from different tests we run, not because we had a different split of test and train.

1. **Naive Bayes section**

“naive\_bayes = GaussianNB()

naive\_bayes.fit(X\_train, y\_train)

y\_pred\_nb = naive\_bayes.predict(X\_test)

accuracy\_nb = accuracy\_score(y\_test, y\_pred\_nb) \* 100

print(f'Accuracy on test set (Naive Bayes): {accuracy\_nb:.2f}%')

f1\_score\_nb = f1\_score(y\_test, y\_pred\_nb, average='macro') \* 100

print(f'F1 Score result: {f1\_score\_nb:.2f}%')

recall\_nb = recall\_score(y\_test, y\_pred\_nb, average='macro') \* 100

print(f'Recall result: {recall\_nb:.2f}%')”

* In the starting section, we are defining the required layers to maintain a NB model.
* The Naive Bayes classifier is trained on the training set.
* Predictions are made on the test set, and the accuracy, F1 score, and recall are calculated and printed.

1. **Harmonic mean calculation section**

**“**harmonic\_mean\_value = harmonic\_mean([accuracy\_nb, f1\_score\_nb, recall\_nb])

print(f'Harmonic Mean of Accuracy, F1 Score, and Recall: {harmonic\_mean\_value:.2f}%')”

* In this section, we are calculating the harmonic mean of accuracy, F1 score, and recall to achieve a metric for them.

1. **Cross validation section**

“cv\_scores = cross\_val\_score(naive\_bayes, X, y, cv=5)

print("Cross-Validation Scores:", cv\_scores)

print("Mean Accuracy:", np.mean(cv\_scores))”

* In this section, we are evaluating the model's performance by splitting the data into multiple folds, training on some of the folds, and doing testing on the not used, remaining fold for couple of times, for 5 times as fort his case. Following these calculations, we are printing the required metrics.

1. **Performance section**

“counter\_end = time.time()

cpu\_end = psutil.cpu\_percent(interval=1)

memory\_end = psutil.virtual\_memory()

print(f"Runtime of the program is {counter\_end - counter\_start:.2f} seconds.")

print(f'Initial CPU usage: {cpu\_start}%, Final CPU usage: {cpu\_end}%')

print(f'Initial Memory usage: {memory\_start.percent}%, Final Memory usage: {memory\_end.percent}%')”

* In this final section, we are printing the total runtime of our program. We are printing end time and calculating the total runtime, along with final CPU and memory usage, to monitor the efficiency of the entire process.