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

**Model Guide for naivebayeswithbaggingboosting.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()

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.
* 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 with Bagging Boosting section**

“base\_naive\_bayes = GaussianNB()

bagging\_naive\_bayes = BaggingClassifier(base\_naive\_bayes, n\_estimators=10, random\_state=42)

bagging\_naive\_bayes.fit(X\_train, y\_train):

…

boosting\_naive\_bayes = AdaBoostClassifier(GaussianNB(), n\_estimators=10, algorithm='SAMME', random\_state=42)

boosting\_naive\_bayes.fit(X\_train, y\_train)

…”

* In the starting section, we are defining the required layers to maintain a NB model.
* for the “random\_state” parameter, 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.
* As for “bagging” section, bagging is used to combine results of multiple models to improve accuracy and robustness. Following calculations, we are printing accuracy, F1 score, and recall.
* As for “boosting” section, boosting is used to applying the model to correct errors from previous iterations for selected times, aiming for higher accuracy. Following calculations, we are printing accuracy, F1 score, and recall.

1. **Harmonic mean calculation section**

**“**harmonic\_mean\_bagging = harmonic\_mean([accuracy\_bagging, f1\_bagging, recall\_bagging])

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

harmonic\_mean\_boosting = harmonic\_mean([accuracy\_boosting, f1\_boosting, recall\_boosting])

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

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

1. **Cross validation section**

“bagging\_cv\_scores = cross\_val\_score(bagging\_naive\_bayes, X, y, cv=5)

print("Bagging Cross-Validation Scores:", bagging\_cv\_scores)

print("Bagging Mean Accuracy:", np.mean(bagging\_cv\_scores))

boosting\_cv\_scores = cross\_val\_score(boosting\_naive\_bayes, X, y, cv=5)

print("Boosting Cross-Validation Scores:", boosting\_cv\_scores)

print("Boosting Mean Accuracy:", np.mean(boosting\_cv\_scores))”

* In this section, we are evaluating the model's performance by splitting the data into multiple folds, training on some folds, and testing on the remaining fold. This is repeated multiple times. Following these calculations, we are printing the required metrics.

1. **Performance section**

“counter\_end = time.time()

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

* In this final section, we are printing the total runtime of our program.