Ain Shams University, Faculty of Engineering

Milestone 1 Report

27 December 2022

Prepared by: Abdulrahman Ayman, Ahmed Sameh, ElSaeed Ahmed ElSaeed

Contents

1)Function Implementation:

Naïve Bayes Classifier:

```
def NaiveBayes_Train_Test(self, train_sizes, test_size):
    output_text = ""
    sample = self.datasamplecombo.currentText().lower().replace(" ", "")
    if sample == "digitdata":
        items_tes = loadDataFile("./digitdata/testimages", test_size, 28, 28)
        Y_test = loadLabelsFile("./digitdata/testlabels", test_size)
        X_test = [[] for i in range(28) for j in range(28)]
    else:
        items_tes = loadDataFile("facedata/facedatatest", test_size, 60, 70)
        Y_test = loadLabelsFile("facedata/facedatatestlabels", test_size)
        X_test = [[] for i in range(60) for j in range(70)]
    for item_index in range(len(items_tes)):...
    X_test = np.transpose(X_test)
    all_outs = []
    all_percent = []
    for sample_size in train_sizes:
        if sample == "digitdata":
            items = loadDataFile("./digitdata/trainingimages", sample_size, 28, 28)
            Y = loadLabelsFile("./digitdata/traininglabels", sample_size)
            X = [[] for i in range(28) for j in range(28)]
        else:
            items = loadDataFile("facedata/facedatatrain", sample_size, 60, 70)
            Y = loadLabelsFile("facedata/facedatatrainlabels", sample_size)
            X = [[] for i in range(60) for j in range(70)]
        for item_index in range(len(items)):...
        X = np.transpose(X)
        clf = GaussianNB(var_smoothing=self.vsmooth)
        clf.fit(X, Y)
        Y_pred = clf.predict(X_test)
        output_text = output_text + "\n" + "..."
        all_outs.append((sample_size, round(accuracy_score(Y_test, Y_pred) * 100, 2)))
        all_percent.append(round(accuracy_score(Y_test, Y_pred) * 100, 2))
        self.temp_text = output_text
    best_pair = all_outs[all_percent.index(max(all_percent))]
    output_text = output_text + "\n" + f"Best Size is {best_pair[0]} with accuracy: {best_pair[1]}"
    self.output_text = output_text
    self.pairs = all_outs
```

Steps:

1. Read either digitdata or facedata to get X_test and Y_test using loadDataFile & LoadLabelsFile provided in the samples.py file

2. Cycle through all sample sizes that are given each time solving classification
3. Once inside for loop load training data either digitdata or facedata to get X and Y using loadDataFile & LoadLabelsFile provided in the samples.py file
4. Create input equal to size of rows * columns and process the loaded data through it
5. Initialize a Gaussian Naïve Bayes Using The implemented version from the SkLearn Library
6. Fit X and Y to the GaussianNB and predict Y_Pred using X_test
7. Check it against Y_test to get both accuracy and precision using function provided by the SkLearn Library
8. Append all outputs to array to get the best one in the end

```
if graph_type == "line":
    x, y = [i[0] for i in pairs], [i[1] for i in pairs]
    plt.xlabel(xlabel)
    plt.ylabel("Accuracy %")
    plt.plot(x, y)
else:
    x, y = [i[0].split(" ")[0] for i in pairs], [i[1] for i in pairs]
    plt.xlabel("Distribution Type")
    plt.ylabel("Accuracy %")
    plt.bar(x, y)
```

9. Graph the result using matplotlib Library

Naïve Bayes Tuning:

```
if flag1 and (not flag2) and (not flag3) and (not flag4):
    output = ""
    all_outs = []
    all_percent = []
    for idx, v in enumerate(np.logspace(int(self.startvar.text()), int(self.endvar.text()), num=17)):
        gnb = GaussianNB(var_smoothing=v)
        gnb.fit(X, Y)
        Y_pred = gnb.predict(X_test)
        output = output + "\n" + "..."
        all_outs.append((v, round(accuracy_score(Y_test, Y_pred) * 100, 2)))
        all_percent.append(round(accuracy_score(Y_test, Y_pred) * 100, 2))
    best_pair = all_outs[all_percent.index(max(all_percent))]
    output = output + "\n" + f"Best Var Smoothing is {best_pair[0]} with accuracy: {best_pair[1]}"
    self.vsmooth = best_pair[0]
    self.output.setText(output)
    self.pairs = all_outs
```

```
else:
    outs = []
    all_percent = []
    if flag1:
        a = GaussianNB()
        a.fit(X, Y)
        Y_pred = a.predict(X_test)
        outs.append(("Gaussian Naive Bayes", round(accuracy_score(Y_test, Y_pred) * 100, 2)))
        all_percent.append(round(accuracy_score(Y_test, Y_pred) * 100, 2))
    if flag2:
        a = BernoulliNB()
        a.fit(X, Y)
        Y_pred = a.predict(X_test)
        outs.append(("Bernoulli Naive Bayes", round(accuracy_score(Y_test, Y_pred) * 100, 2)))
        all_percent.append(round(accuracy_score(Y_test, Y_pred) * 100, 2))
    if flag3:
        a = ComplementNB()
        a.fit(X, Y)
        Y_pred = a.predict(X_test)
        outs.append(("Complement Naive Bayes", round(accuracy_score(Y_test, Y_pred) * 100, 2)))
        all_percent.append(round(accuracy_score(Y_test, Y_pred) * 100, 2))
    if flag4:
        a = MultinomialNB()
        a.fit(X, Y)
        Y_pred = a.predict(X_test)
        outs.append(("Multinomial Naive Bayes", round(accuracy_score(Y_test, Y_pred) * 100, 2)))
        all_percent.append(round(accuracy_score(Y_test, Y_pred) * 100, 2))
    text = ""
    for line in outs:
        text += f"{line[0]} with accuracy: {line[1]}%\n"
    best_pair = outs[all_percent.index(max(all_percent))]
    text = text + "\n" + f"Best Distribution is {best_pair[0]} with accuracy: {best_pair[1]}%"
    self.output.setText(text)
    self.pairs = outs
```

```
self.pairs = []
if self.sample == "digitdata":...
else:...

for item_index in range(len(items_tes)):...
X_test = np.transpose(X_test)

if self.sample == "digitdata":...
else:...
for item_index in range(len(items)):...
X = np.transpose(X)
```

Steps:

1. Read either digitdata or facedata to get X_test and Y_test using loadDataFile & LoadLabelsFile provided in the samples.py file (same as all other functions)

2. load training data either digitdata or facedata to get X and Y using loadDataFile & LoadLabelsFile provided in the samples.py file (same as all other functions)
3. If we are tuning only gaussian distribution, then we cycle through all v smoothing values that are given each time solving classification
    a. Initialize a GaussianNB Using The implemented version from the SkLearn Library with var smoothing the current var smoothing
    b. Fit X and Y to the GaussianNB and predict Y_Pred using X_test
    c. Check it against Y_test to get both accuracy and precision using function provided by the SkLearn Library
    d. Append all outputs to array to get the best one in the end with the best one being the new var smoothing value
4. If we are comparing distributions
    a. we Initialize Distributions for each one checked Using The implemented version from the SkLearn Library
    b. Fit X and Y for all to the distributions and predict Y_Pred using X_test
    c. Check it against Y_test to get both accuracy and precision for each one using function provided by the SkLearn Library
    d. Append all outputs to array to get the best one in the end with the best one being the best distribution

```
if graph_type == "line":
    x, y = [i[0] for i in pairs], [i[1] for i in pairs]
    plt.xlabel(xlabel)
    plt.ylabel("Accuracy %")
    plt.plot(x, y)
else:
    x, y = [i[0].split(" ")[0] for i in pairs], [i[1] for i in pairs]
    plt.xlabel("Distribution Type")
    plt.ylabel("Accuracy %")
    plt.bar(x, y)
```

5. Graph the result using matplotlib Library

KNN Classifier:

```
def KNN_Train_Test(self, train_sizes, test_size):
    k = self.knnK
    output_text = ""
    sample = self.datasamplecombo.currentText().lower().replace(" ", "")
    if sample == "digitdata":
        items_tes = loadDataFile("./digitdata/testimages", test_size, 28, 28)
        Y_test = loadLabelsFile("./digitdata/testlabels", test_size)
        X_test = [[] for i in range(28) for j in range(28)]
    else:
        items_tes = loadDataFile("facedata/facedatatest", test_size, 60, 70)
        Y_test = loadLabelsFile("facedata/facedatatestlabels", test_size)
        X_test = [[] for i in range(60) for j in range(70)]
    for item_index in range(len(items_tes)):...
    X_test = np.transpose(X_test)
    all_outs = []
    all_percent = []
    for sample_size in train_sizes:
        if sample == "digitdata":
            items = loadDataFile("./digitdata/trainingimages", sample_size, 28, 28)
            Y = loadLabelsFile("./digitdata/traininglabels", sample_size)
            X = [[] for i in range(28) for j in range(28)]
        else:
            items = loadDataFile("facedata/facedatatrain", sample_size, 60, 70)
            Y = loadLabelsFile("facedata/facedatatrainlabels", sample_size)
            X = [[] for i in range(60) for j in range(70)]
        for item_index in range(len(items)):...
        X = np.transpose(X)
        neigh = KNeighborsClassifier(n_neighbors=k, metric="euclidean")
        neigh.fit(X, Y)
        Y_pred = neigh.predict(X_test)
        output_text = output_text + "\n" + "..."
        all_outs.append((sample_size, round(accuracy_score(Y_test, Y_pred) * 100, 2)))
        all_percent.append(round(accuracy_score(Y_test, Y_pred) * 100, 2))
        self.temp_text = output_text
    best_pair = all_outs[all_percent.index(max(all_percent))]
    output_text = output_text + "\n" + f"Best Size is {best_pair[0]} with accuracy: {best_pair[1]}"
    self.output_text = output_text
    self.pairs = all_outs
```

Steps:

1. Read either digitdata or facedata to get X_test and Y_test using loadDataFile & LoadLabelsFile provided in the samples.py file

2. Cycle through all sample sizes that are given each time solving classification
3. Once inside for loop load training data either digitdata or facedata to get X and Y using loadDataFile & LoadLabelsFile provided in the samples.py file
4. Create input equal to size of rows * columns and process the loaded data through it
5. Initialize a KNeighborsClassifier Using The implemented version from the SkLearn Library with default K = 2 & Euclidean metric
6. Fit X and Y to the KNeighborsClassifier and predict Y_Pred using X_test
7. Check it against Y_test to get both accuracy and precision using function provided by the SkLearn Library
8. Append all outputs to array to get the best one in the end

```
if graph_type == "line":
    x, y = [i[0] for i in pairs], [i[1] for i in pairs]
    plt.xlabel(xlabel)
    plt.ylabel("Accuracy %")
    plt.plot(x, y)
else:
    x, y = [i[0].split(" ")[0] for i in pairs], [i[1] for i in pairs]
    plt.xlabel("Distribution Type")
    plt.ylabel("Accuracy %")
    plt.bar(x, y)
```

Graph the result using matplotlib Library

KNN Tuning:

```
def go_Press(self):
    try:
        self.pairs = []
        if self.sample == "digitdata":...
        else:...
        for item_index in range(len(items_tes)):...
        X_test = np.transpose(X_test)

        if self.sample == "digitdata":...
        else:...
        for item_index in range(len(items)):...
        X = np.transpose(X)

        output = ""
        all_outs = []
        all_percent = []
        for k in range(int(self.startk.text()), int(self.endk.text())+1):...
        best_pair = all_outs[all_percent.index(max(all_percent))]
        output = output + "\n" + f"Best k is {best_pair[0]} with accuracy: {best_pair[1]}"
        self.k = best_pair[0]
        self.output.setText(output)
        self.pairs = all_outs
    except Exception:...
```

Steps:

6. Read either digitdata or facedata to get X_test and Y_test using loadDataFile & LoadLabelsFile provided in the samples.py file
7. load training data either digitdata or facedata to get X and Y using loadDataFile & LoadLabelsFile provided in the samples.py file
8. Cycle through all K neighbours that are given each time solving classification
9. Initialize a KNeighborsClassifier Using The implemented version from the SkLearn Library with K the current K & Euclidean metric

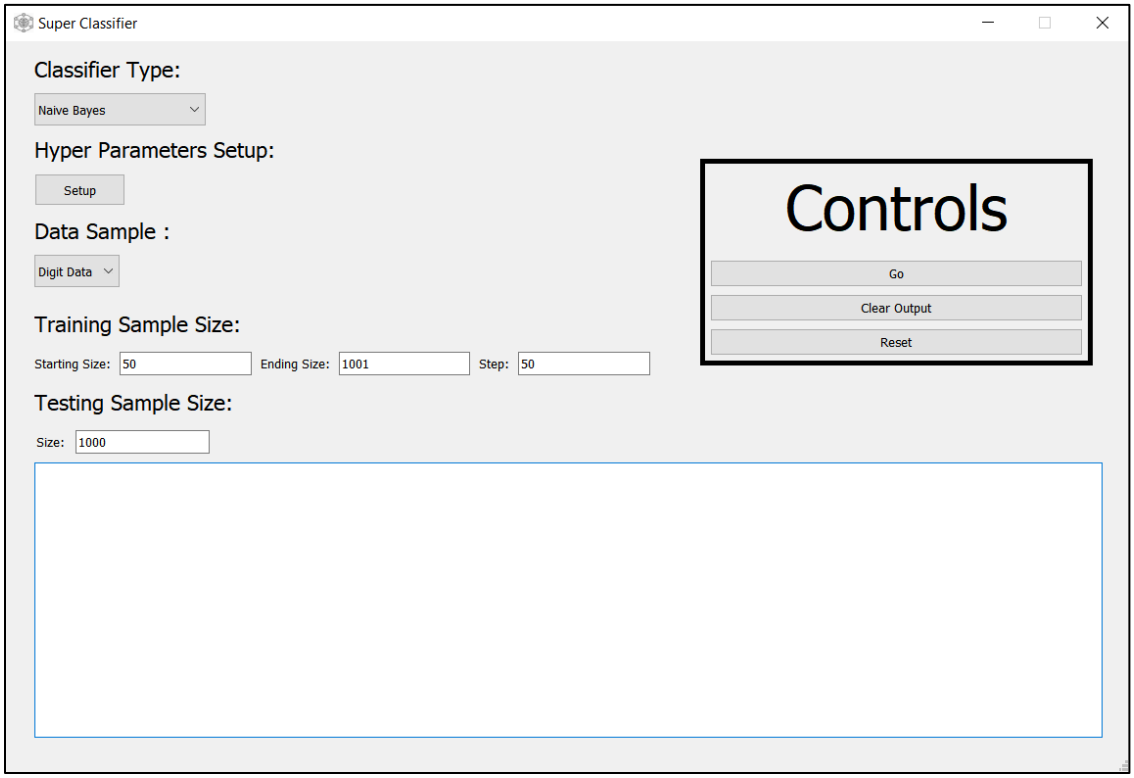
10. Fit X and Y to the KNeighborsClassifier and predict Y_Pred using X_test
11. Check it against Y_test to get both accuracy and precision using function provided by the SkLearn Library
12. Append all outputs to array to get the best one in the end with the best one being the new K value

```
if graph_type == "line":
    x, y = [i[0] for i in pairs], [i[1] for i in pairs]
    plt.xlabel(xlabel)
    plt.ylabel("Accuracy %")
    plt.plot(x, y)
else:
    x, y = [i[0].split(" ")[0] for i in pairs], [i[1] for i in pairs]
    plt.xlabel("Distribution Type")
    plt.ylabel("Accuracy %")
    plt.bar(x, y)
```

Graph the result using matplotlib Library

2)Running Project:

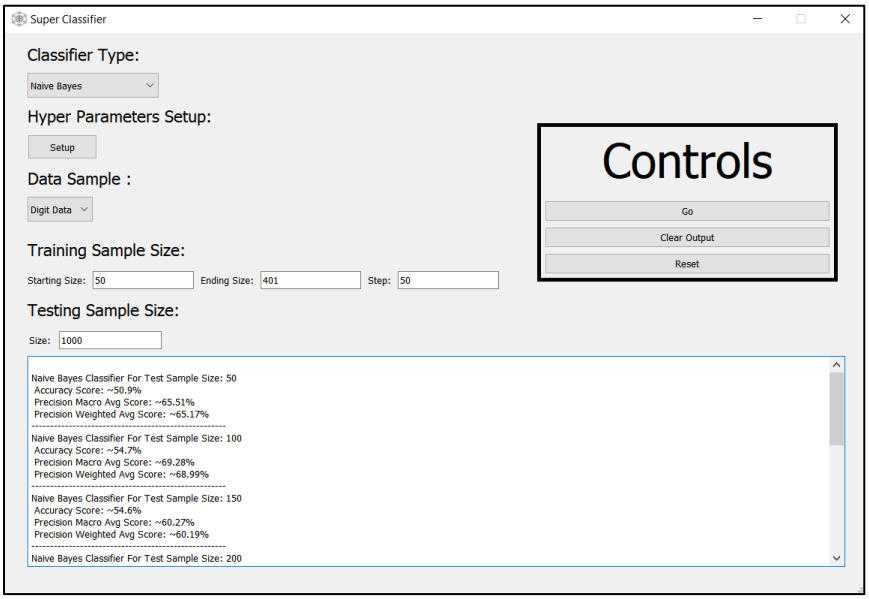
Main Screen:



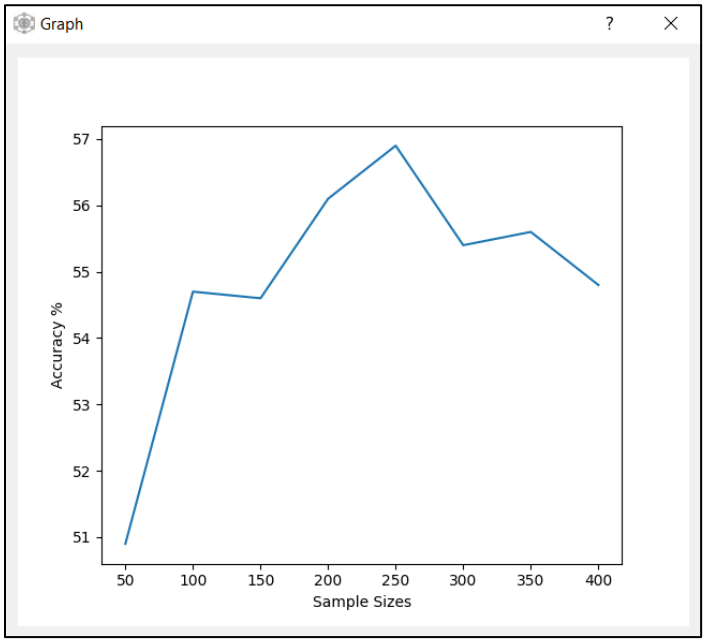
Here we have the main screen:

- we can choose classifier type from comb box
- we can setup hyperparameter from setup button
- we can choose which sample to use from data sample
- we can set variable training size by giving start, end and step in the text boxes which will be transferred to format range(start, end, step) in code
- we can set variable testing size in the text box

Naïve Bayes Functions:



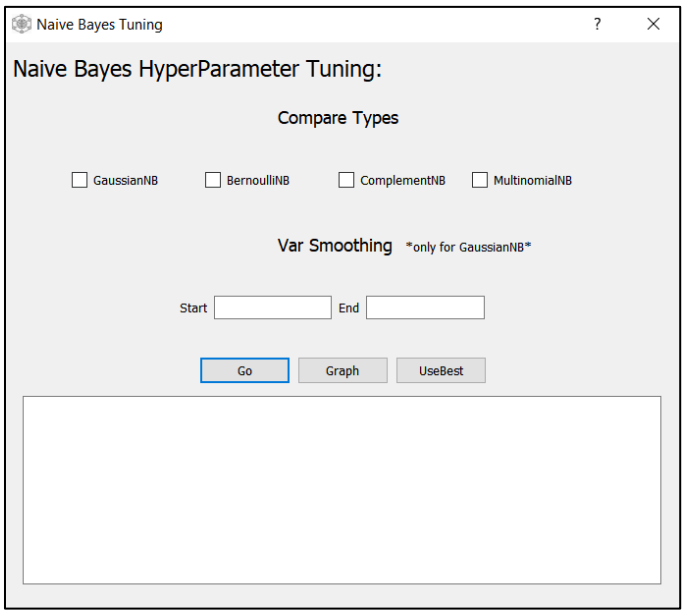
- By pressing Go we get output of Gaussian Naïve bayes for all training sizes we have set:




```
Naive Bayes Classifier For Test Sample Size: 50
 Accuracy Score: ~50.9%
 Precision Macro Avg Score: ~65.51%
 Precision Weighted Avg Score: ~65.17%
----------------------------------------------------
Naive Bayes Classifier For Test Sample Size: 100
 Accuracy Score: ~54.7%
 Precision Macro Avg Score: ~69.28%
 Precision Weighted Avg Score: ~68.99%
----------------------------------------------------
Naive Bayes Classifier For Test Sample Size: 150
 Accuracy Score: ~54.6%
 Precision Macro Avg Score: ~60.27%
 Precision Weighted Avg Score: ~60.19%
----------------------------------------------------
Naive Bayes Classifier For Test Sample Size: 200
 Accuracy Score: ~56.1%
 Precision Macro Avg Score: ~60.79%
 Precision Weighted Avg Score: ~60.49%
----------------------------------------------------
Naive Bayes Classifier For Test Sample Size: 250
 Accuracy Score: ~56.9%
 Precision Macro Avg Score: ~59.08%
 Precision Weighted Avg Score: ~58.81%
----------------------------------------------------
Naive Bayes Classifier For Test Sample Size: 300
 Accuracy Score: ~55.4%
 Precision Macro Avg Score: ~58.62%
 Precision Weighted Avg Score: ~58.24%
----------------------------------------------------
Naive Bayes Classifier For Test Sample Size: 350
 Accuracy Score: ~55.6%
 Precision Macro Avg Score: ~59.81%
 Precision Weighted Avg Score: ~59.41%
----------------------------------------------------
Naive Bayes Classifier For Test Sample Size: 400
 Accuracy Score: ~54.8%
 Precision Macro Avg Score: ~59.8%
 Precision Weighted Avg Score: ~59.34%
----------------------------------------------------
Best Size is 250 with accuracy: 56.9
```

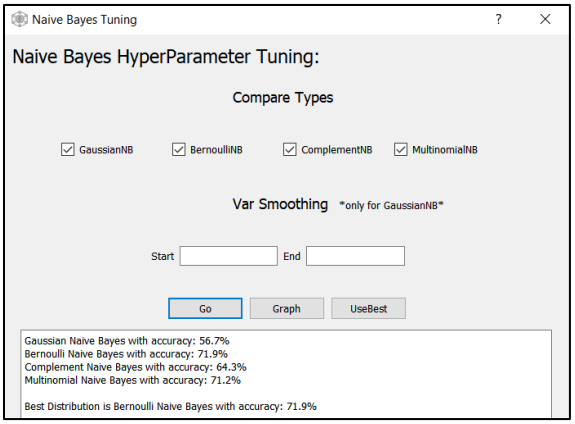
- By pressing Clear we clear output & by pressing reset will reset all variables and all textboxes

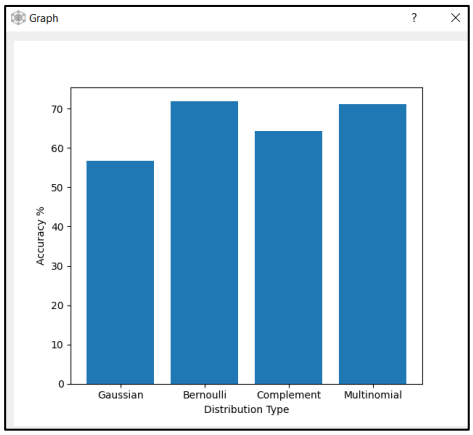
- By pressing Hyper parameter Setup Button:



We get the following screen where we can choose different distribution types to compare their accuracy you can either:

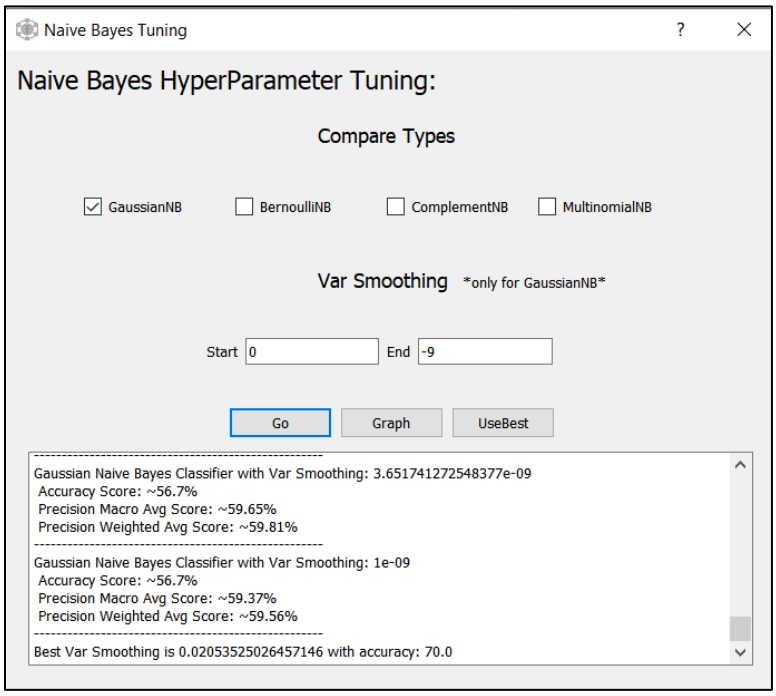
1. Compare by checking multiple distributions

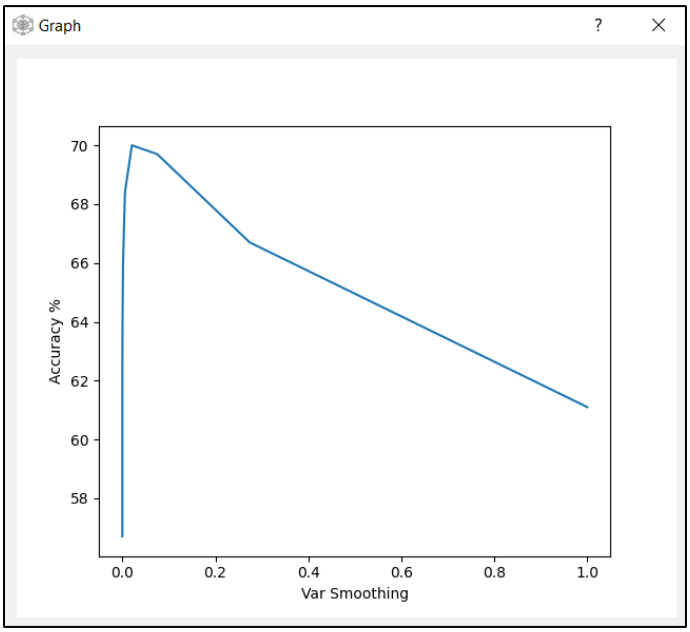




Note: if you want to graph press go first then press graph

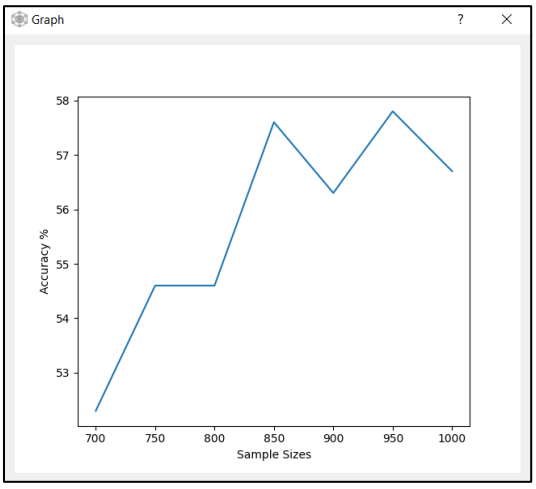
2. Tune the var smoothing parameter by choosing GaussianNB and entering start and end





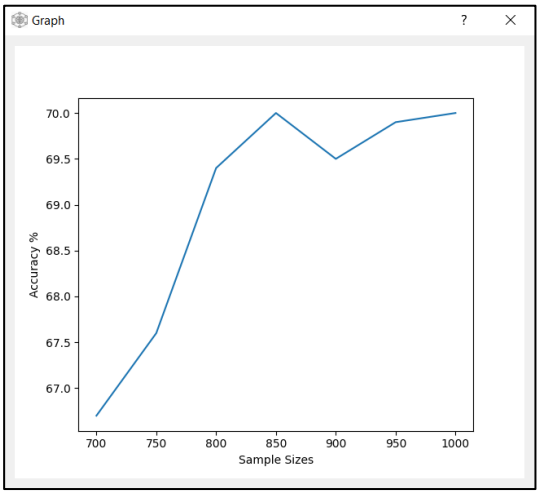
Note: By pressing UseBest Button you will return to main screen and the best var smoothing will be applied

Before:



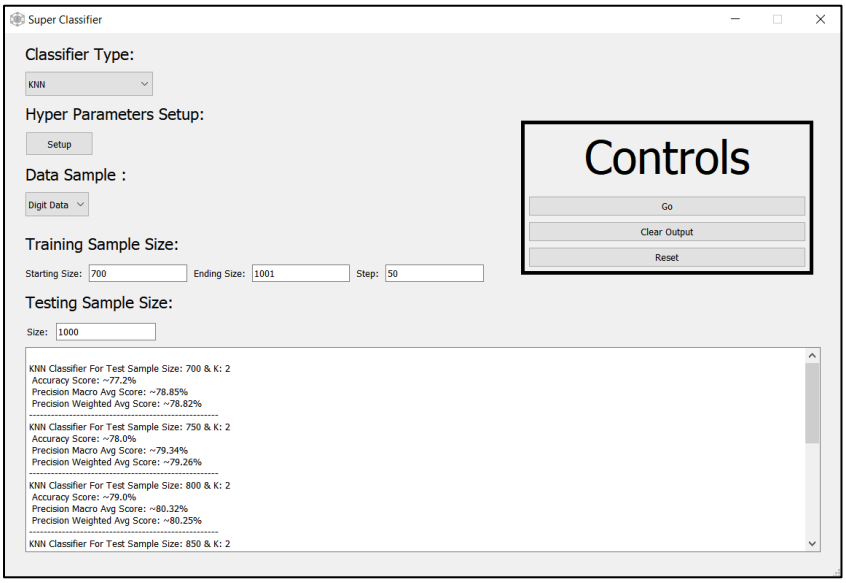

```
Naive Bayes Classifier For Test Sample Size: 700
 Accuracy Score: ~52.3%
 Precision Macro Avg Score: ~55.82%
 Precision Weighted Avg Score: ~55.75%
--------------------------------------------------
Naive Bayes Classifier For Test Sample Size: 750
 Accuracy Score: ~54.6%
 Precision Macro Avg Score: ~58.14%
 Precision Weighted Avg Score: ~58.13%
--------------------------------------------------
Naive Bayes Classifier For Test Sample Size: 800
 Accuracy Score: ~54.6%
 Precision Macro Avg Score: ~58.1%
 Precision Weighted Avg Score: ~58.11%
--------------------------------------------------
Naive Bayes Classifier For Test Sample Size: 850
 Accuracy Score: ~57.6%
 Precision Macro Avg Score: ~59.84%
 Precision Weighted Avg Score: ~59.88%
--------------------------------------------------
Naive Bayes Classifier For Test Sample Size: 900
 Accuracy Score: ~56.3%
 Precision Macro Avg Score: ~59.87%
 Precision Weighted Avg Score: ~59.97%
--------------------------------------------------
Naive Bayes Classifier For Test Sample Size: 950
 Accuracy Score: ~57.8%
 Precision Macro Avg Score: ~59.86%
 Precision Weighted Avg Score: ~60.01%
--------------------------------------------------
Naive Bayes Classifier For Test Sample Size: 1000
 Accuracy Score: ~56.7%
 Precision Macro Avg Score: ~59.37%
 Precision Weighted Avg Score: ~59.56%
--------------------------------------------------
Best Size is 950 with accuracy: 57.8
```

After:

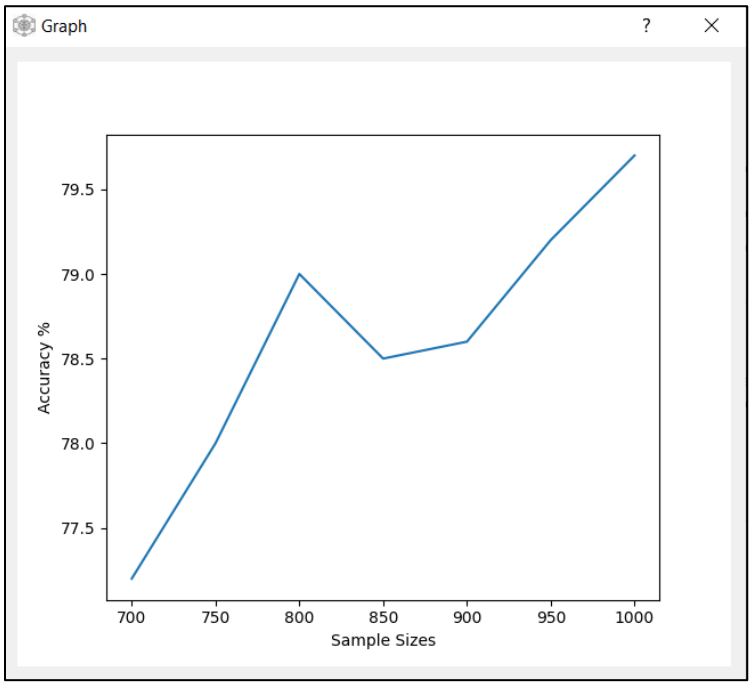



```
Naive Bayes Classifier For Test Sample Size: 700
 Accuracy Score: ~66.7%
 Precision Macro Avg Score: ~70.04%
 Precision Weighted Avg Score: ~70.1%
--------------------------------------------------
Naive Bayes Classifier For Test Sample Size: 750
 Accuracy Score: ~67.6%
 Precision Macro Avg Score: ~70.8%
 Precision Weighted Avg Score: ~70.83%
--------------------------------------------------
Naive Bayes Classifier For Test Sample Size: 800
 Accuracy Score: ~69.4%
 Precision Macro Avg Score: ~72.16%
 Precision Weighted Avg Score: ~72.19%
--------------------------------------------------
Naive Bayes Classifier For Test Sample Size: 850
 Accuracy Score: ~70.0%
 Precision Macro Avg Score: ~72.57%
 Precision Weighted Avg Score: ~72.57%
--------------------------------------------------
Naive Bayes Classifier For Test Sample Size: 900
 Accuracy Score: ~69.5%
 Precision Macro Avg Score: ~72.37%
 Precision Weighted Avg Score: ~72.33%
--------------------------------------------------
Naive Bayes Classifier For Test Sample Size: 950
 Accuracy Score: ~69.9%
 Precision Macro Avg Score: ~72.77%
 Precision Weighted Avg Score: ~72.73%
--------------------------------------------------
Naive Bayes Classifier For Test Sample Size: 1000
 Accuracy Score: ~70.0%
 Precision Macro Avg Score: ~72.67%
 Precision Weighted Avg Score: ~72.65%
--------------------------------------------------
Best Size is 850 with accuracy: 70.0
```

KNN Functions:



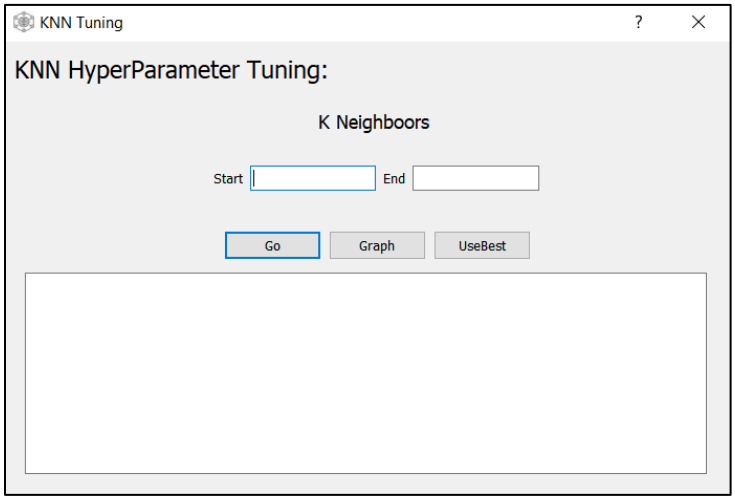
- By pressing Go we get output of KNN Classification results for all training sizes we have set and a default K value of 2:



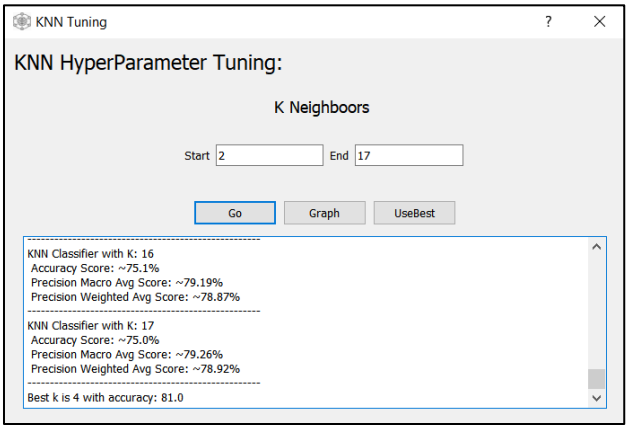

```
KNN Classifier For Test Sample Size: 700 & K: 2
 Accuracy Score: ~77.2%
 Precision Macro Avg Score: ~78.85%
 Precision Weighted Avg Score: ~78.82%
--------------------------------------------------
KNN Classifier For Test Sample Size: 750 & K: 2
 Accuracy Score: ~78.0%
 Precision Macro Avg Score: ~79.34%
 Precision Weighted Avg Score: ~79.26%
--------------------------------------------------
KNN Classifier For Test Sample Size: 800 & K: 2
 Accuracy Score: ~79.0%
 Precision Macro Avg Score: ~80.32%
 Precision Weighted Avg Score: ~80.25%
--------------------------------------------------
KNN Classifier For Test Sample Size: 850 & K: 2
 Accuracy Score: ~78.5%
 Precision Macro Avg Score: ~79.82%
 Precision Weighted Avg Score: ~79.73%
--------------------------------------------------
KNN Classifier For Test Sample Size: 900 & K: 2
 Accuracy Score: ~78.6%
 Precision Macro Avg Score: ~79.87%
 Precision Weighted Avg Score: ~79.78%
--------------------------------------------------
KNN Classifier For Test Sample Size: 950 & K: 2
 Accuracy Score: ~79.2%
 Precision Macro Avg Score: ~80.28%
 Precision Weighted Avg Score: ~80.21%
--------------------------------------------------
KNN Classifier For Test Sample Size: 1000 & K: 2
 Accuracy Score: ~79.7%
 Precision Macro Avg Score: ~80.82%
 Precision Weighted Avg Score: ~80.77%
--------------------------------------------------
Best Size is 1000 with accuracy: 79.7
```

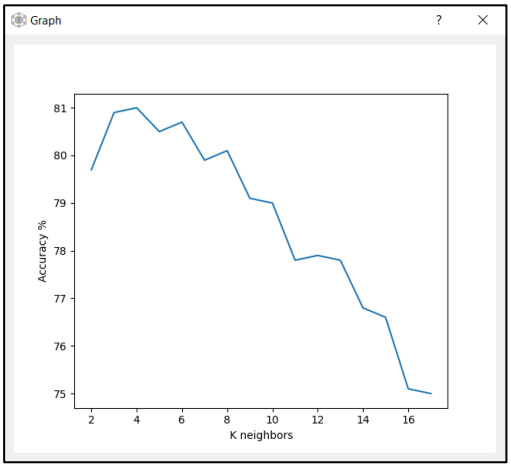
- By pressing Clear we clear output & by pressing reset will reset all variables and all textboxes

- By pressing Hyper parameter Setup Button:



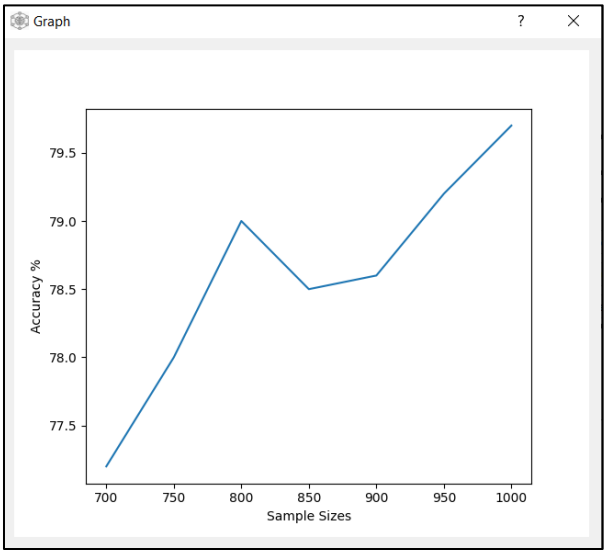
We get the following screen where we test a range of values for the nearest neighbour hyper parameter by entering start and end then pressing





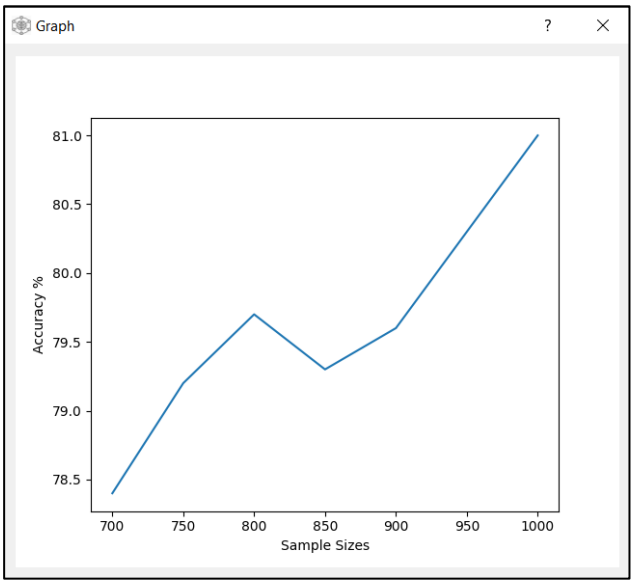
Note: By pressing UseBest Button you will return to main screen and the best K will be applied

Before:




```
KNN Classifier For Test Sample Size: 700 & K: 2
 Accuracy Score: ~77.2%
 Precision Macro Avg Score: ~78.85%
 Precision Weighted Avg Score: ~78.82%
-------------------------------------------------------
KNN Classifier For Test Sample Size: 750 & K: 2
 Accuracy Score: ~78.0%
 Precision Macro Avg Score: ~79.34%
 Precision Weighted Avg Score: ~79.26%
-------------------------------------------------------
KNN Classifier For Test Sample Size: 800 & K: 2
 Accuracy Score: ~79.0%
 Precision Macro Avg Score: ~80.32%
 Precision Weighted Avg Score: ~80.25%
-------------------------------------------------------
KNN Classifier For Test Sample Size: 850 & K: 2
 Accuracy Score: ~78.5%
 Precision Macro Avg Score: ~79.82%
 Precision Weighted Avg Score: ~79.73%
-------------------------------------------------------
KNN Classifier For Test Sample Size: 900 & K: 2
 Accuracy Score: ~78.6%
 Precision Macro Avg Score: ~79.87%
 Precision Weighted Avg Score: ~79.78%
-------------------------------------------------------
KNN Classifier For Test Sample Size: 950 & K: 2
 Accuracy Score: ~79.2%
 Precision Macro Avg Score: ~80.28%
 Precision Weighted Avg Score: ~80.21%
-------------------------------------------------------
KNN Classifier For Test Sample Size: 1000 & K: 2
 Accuracy Score: ~79.7%
 Precision Macro Avg Score: ~80.82%
 Precision Weighted Avg Score: ~80.77%
-------------------------------------------------------
Best Size is 1000 with accuracy: 79.7
```

After:




```
KNN Classifier For Test Sample Size: 700 & K: 4
 Accuracy Score: ~78.4%
 Precision Macro Avg Score: ~80.6%
 Precision Weighted Avg Score: ~80.44%
-------------------------------------------------------
KNN Classifier For Test Sample Size: 750 & K: 4
 Accuracy Score: ~79.2%
 Precision Macro Avg Score: ~81.04%
 Precision Weighted Avg Score: ~80.89%
-------------------------------------------------------
KNN Classifier For Test Sample Size: 800 & K: 4
 Accuracy Score: ~79.7%
 Precision Macro Avg Score: ~81.3%
 Precision Weighted Avg Score: ~81.15%
-------------------------------------------------------
KNN Classifier For Test Sample Size: 850 & K: 4
 Accuracy Score: ~79.3%
 Precision Macro Avg Score: ~81.0%
 Precision Weighted Avg Score: ~80.84%
-------------------------------------------------------
KNN Classifier For Test Sample Size: 900 & K: 4
 Accuracy Score: ~79.6%
 Precision Macro Avg Score: ~81.19%
 Precision Weighted Avg Score: ~81.02%
-------------------------------------------------------
KNN Classifier For Test Sample Size: 950 & K: 4
 Accuracy Score: ~80.3%
 Precision Macro Avg Score: ~81.59%
 Precision Weighted Avg Score: ~81.45%
-------------------------------------------------------
KNN Classifier For Test Sample Size: 1000 & K: 4
 Accuracy Score: ~81.0%
 Precision Macro Avg Score: ~82.29%
 Precision Weighted Avg Score: ~82.12%
-------------------------------------------------------
Best Size is 1000 with accuracy: 81.0
```