```python
import os
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import preprocessing,svm
from sklearn.model_selection import train_test_split
from sklearn.model_selection import StratifiedKFold ,
cross_val_score,KFold
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score,f1_score,confusion_matrix
from sklearn.svm import SVC  # For Support Vector Classifier
from sklearn.svm import LinearSVC  # For Linear Support Vector
Classifier
from sklearn.tree import DecisionTreeClassifier  # For Decision Tree
Classifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import KFold
from sklearn.neighbors import KNeighborsClassifier

!pip install seaborn
```

```
Requirement already satisfied: seaborn in c:\users\bhanu\appdata\
local\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\
localcache\local-packages\python311\site-packages (0.13.2)
Requirement already satisfied: numpy!=1.24.0,>=1.20 in c:\users\bhanu\
appdata\local\packages\
pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\local-
packages\python311\site-packages (from seaborn) (2.1.1)
Requirement already satisfied: pandas>=1.2 in c:\users\bhanu\appdata\
local\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\
localcache\local-packages\python311\site-packages (from seaborn)
(2.2.3)
Requirement already satisfied: matplotlib!=3.6.1,>=3.4 in c:\users\
bhanu\appdata\local\packages\
pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\local-
packages\python311\site-packages (from seaborn) (3.9.2)
Requirement already satisfied: contourpy>=1.0.1 in c:\users\bhanu\
appdata\local\packages\
pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\local-
packages\python311\site-packages (from matplotlib!=3.6.1,>=3.4-
>seaborn) (1.3.0)
Requirement already satisfied: cycler>=0.10 in c:\users\bhanu\appdata\
local\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\
localcache\local-packages\python311\site-packages (from matplotlib!
=3.6.1,>=3.4->seaborn) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\bhanu\
appdata\local\packages\
pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\local-
packages\python311\site-packages (from matplotlib!=3.6.1,>=3.4-
>seaborn) (4.54.1)
Requirement already satisfied: kiwisolver>=1.3.1 in c:\users\bhanu\
```

```
appdata\local\packages\
pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\local-
packages\python311\site-packages (from matplotlib!=3.6.1,>=3.4-
>seaborn) (1.4.7)
Requirement already satisfied: packaging>=20.0 in c:\users\bhanu\
appdata\local\packages\
pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\local-
packages\python311\site-packages (from matplotlib!=3.6.1,>=3.4-
>seaborn) (24.1)
Requirement already satisfied: pillow>=8 in c:\users\bhanu\appdata\
local\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\
localcache\local-packages\python311\site-packages (from matplotlib!
=3.6.1,>=3.4->seaborn) (10.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in c:\users\bhanu\
appdata\local\packages\
pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\local-
packages\python311\site-packages (from matplotlib!=3.6.1,>=3.4-
>seaborn) (3.1.4)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\bhanu\
appdata\local\packages\
pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\local-
packages\python311\site-packages (from matplotlib!=3.6.1,>=3.4-
>seaborn) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in c:\users\bhanu\appdata\
local\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\
localcache\local-packages\python311\site-packages (from pandas>=1.2-
>seaborn) (2024.2)
Requirement already satisfied: tzdata>=2022.7 in c:\users\bhanu\
appdata\local\packages\
pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\local-
packages\python311\site-packages (from pandas>=1.2->seaborn) (2024.2)
Requirement already satisfied: six>=1.5 in c:\users\bhanu\appdata\
local\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\
localcache\local-packages\python311\site-packages (from python-
dateutil>=2.7->matplotlib!=3.6.1,>=3.4->seaborn) (1.16.0)


[notice] A new release of pip is available: 24.2 -> 24.3.1
[notice] To update, run: C:\Users\Bhanu\AppData\Local\Microsoft\
WindowsApps\PythonSoftwareFoundation.Python.3.11_qbz5n2kfra8p0\
python.exe -m pip install --upgrade pip
```

```python
directory = r"D:\data_mining\cropped"
```

```python
from skimage import filters
from skimage import data, exposure, img_as_float
from skimage.color import rgb2gray
from skimage.io import imread
import numpy as np
def angle(dx, dy):
```

```
    return np.mod(np.arctan2(dy, dx), np.pi)
hist=[]
label=[]
for index,name in enumerate(os.listdir(directory)):
    for image in os.listdir(os.path.join(directory,name)):
        img = imread(os.path.join(directory,name,image.strip()))
        gray_img = rgb2gray(img)
        angle_sobel =
angle(filters.sobel_h(gray_img),filters.sobel_v(gray_img))
        Hist,_=exposure.histogram(angle_sobel, nbins=36)
        hist.append(Hist)
        label.append(index)

X = np.array(hist)
Y = np.array(label)

X_train, X_test, y_train, y_test = train_test_split(X, Y,
test_size=0.2, stratify=Y,random_state=42)

scaler = preprocessing.StandardScaler().fit(X_train)

X_scaled = scaler.transform(X_train)

X_scaled.mean(axis=0)
```

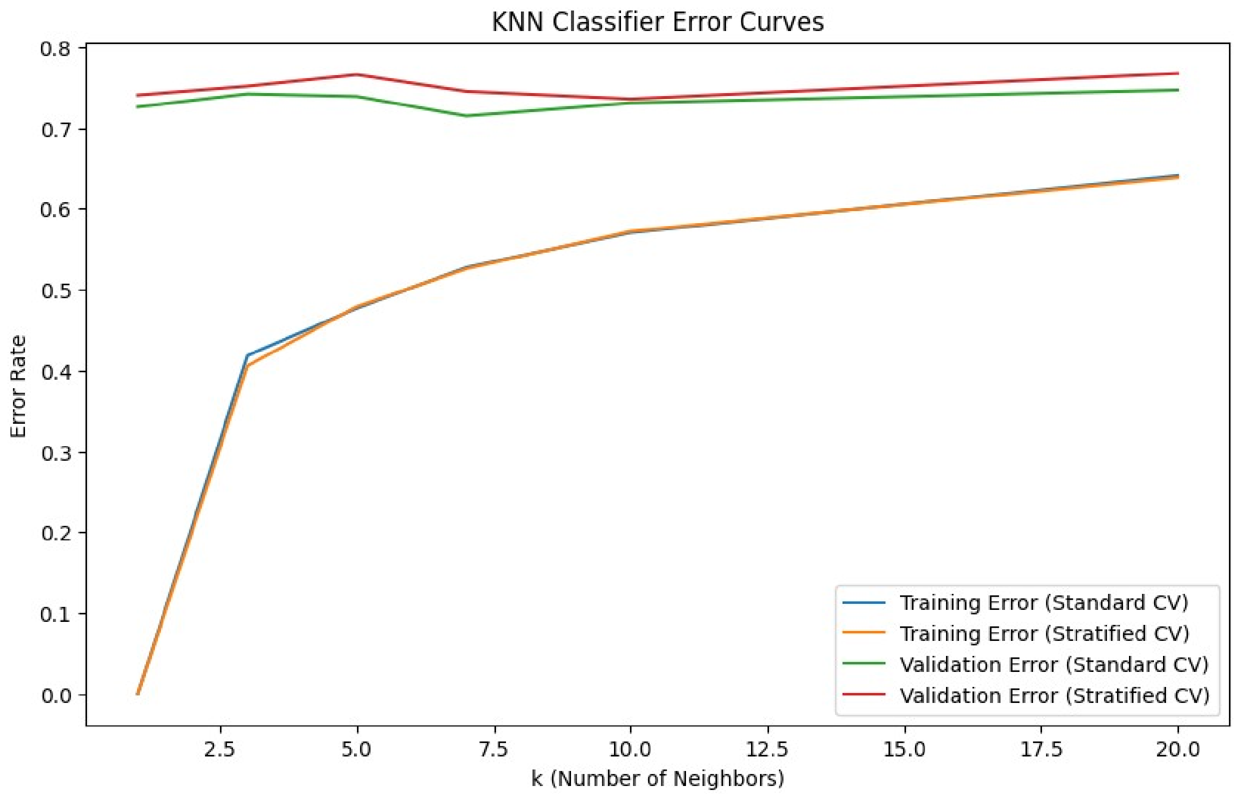
```
array([-3.53574212e-18, -9.13105402e-17,  1.15795554e-16, -
1.63925844e-16,
       -3.76556535e-16,  8.87471271e-17,  4.09262150e-17,
3.80976213e-16,
       -5.65188377e-16,  3.28779820e-16,  3.49331321e-16,
1.48324382e-16,
       -3.54988509e-16,  2.59877046e-16, -2.04012320e-16,
1.09961580e-16,
       -1.52744059e-16,  1.14204470e-16, -2.38309019e-16,
4.22521183e-17,
        2.72252143e-16,  1.06602625e-16, -3.64181438e-17,
2.85643766e-16,
       -4.05240243e-16, -1.04481180e-16, -3.58259070e-16,
2.11437379e-16,
       -1.88720235e-16,  3.63474290e-16, -3.45353611e-16,
5.09146865e-17,
        1.23397400e-16,  1.29054587e-16,  1.17033064e-16, -
6.14335193e-17])
```

```
X_scaled.std(axis=0)
```

```
array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
1.,
       1., 1.])
```

```
X_test_scaled = scaler.transform(X_test)

train_errors=[]
val_errors=[]
kf = KFold(n_splits=5, shuffle=True, random_state=42)
k_val=[1,3,5,7,10,20]
for k in k_val:
    knn = KNeighborsClassifier(n_neighbors=k)
    train = []
    val= []
    for train_idx, val_idx in kf.split(X_scaled):
        x_train, x_val = X_scaled[train_idx], X_scaled[val_idx]
        Y_train, Y_val = y_train[train_idx], y_train[val_idx]
        knn.fit(x_train, Y_train)
        train_pred = knn.predict(x_train)
        val_pred = knn.predict(x_val)
        train_accuracy = accuracy_score(Y_train, train_pred)
        val_accuracy = accuracy_score(Y_val, val_pred)
        train.append(1 - train_accuracy)
        val.append(1 - val_accuracy)
    train_errors.append(np.mean(train))
    val_errors.append(np.mean(val))

stratified_train_e=[]
stratified_val_e=[]
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

for k in k_val:
    knn = KNeighborsClassifier(n_neighbors=k)
    train = []
    val= []
    for train_idx, val_idx in skf.split(X_scaled,y_train):
        x_train, x_val = X_scaled[train_idx], X_scaled[val_idx]
        Y_train, Y_val = y_train[train_idx], y_train[val_idx]
        knn.fit(x_train, Y_train)
        train_pred = knn.predict(x_train)
        val_pred = knn.predict(x_val)
        train_accuracy = accuracy_score(Y_train, train_pred)
        val_accuracy = accuracy_score(Y_val, val_pred)
        train.append(1 - train_accuracy)
        val.append(1 - val_accuracy)
    stratified_train_e.append(np.mean(train))
    stratified_val_e.append(np.mean(val))

plt.figure(figsize=(10, 6))
plt.plot(k_val, train_errors, label='Training Error (Standard CV)')
plt.plot(k_val, stratified_train_e, label='Training Error (Stratified
CV)')
plt.plot(k_val, val_errors, label='Validation Error (Standard CV)')
plt.plot(k_val, stratified_val_e, label='Validation Error (Stratified
```

```
CV)')

plt.title('KNN Classifier Error Curves')
plt.xlabel('k (Number of Neighbors)')
plt.ylabel('Error Rate')
plt.legend()

plt.show()
```




```
# k=3
Model = KNeighborsClassifier(n_neighbors = 3)
Model.fit(X_scaled, y_train)
p = Model.predict(X_test_scaled)
print("Test error when k=3 :" + str(1-(accuracy_score(y_test,p))))
```

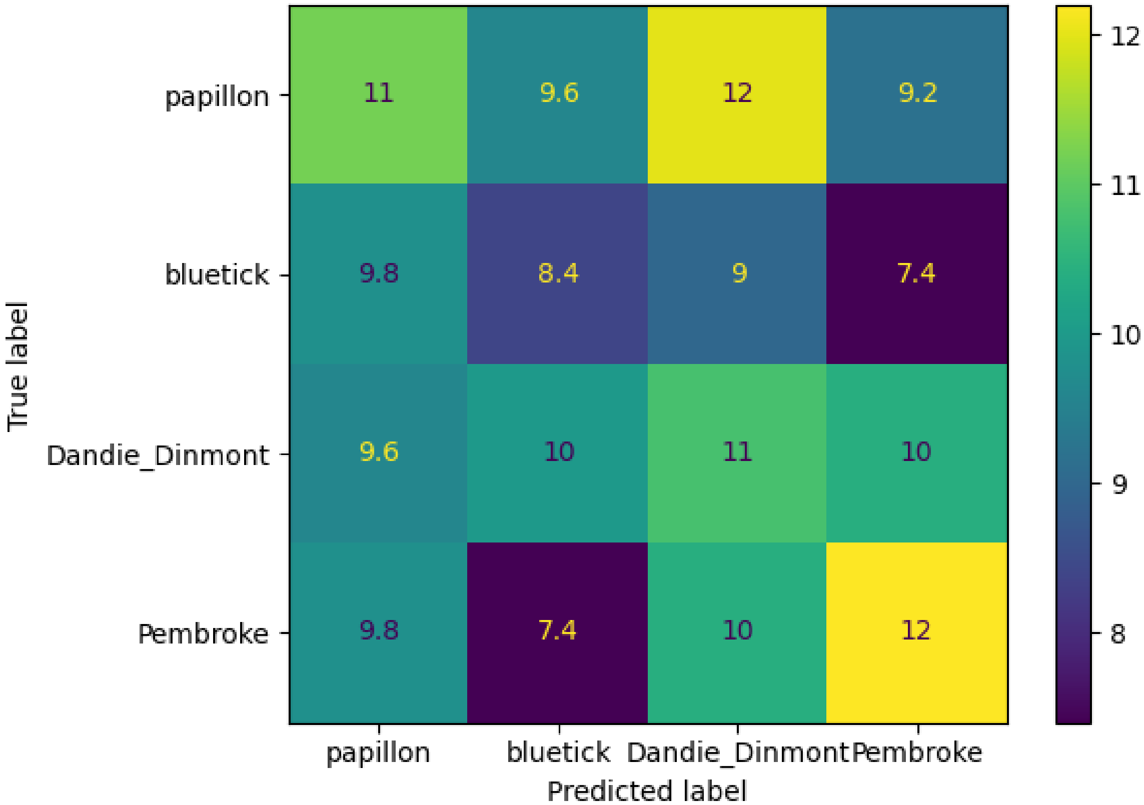
```
Test error when k=3 :0.6835443037974683
```

```
from sklearn import metrics
from sklearn.ensemble import AdaBoostClassifier
from sklearn.neural_network import MLPClassifier
dt_model=DecisionTreeClassifier()
neural_network=MLPClassifier(hidden_layer_sizes=(10,10,10))
Ada_boost= AdaBoostClassifier()
for clf in [dt_model,neural_network,Ada_boost]:
    print(str(clf)+"\n\n")
```

```
    clf.fit(X_scaled,y_train)
    predictions=clf.predict(X_test_scaled)
    confusion_matrix = metrics.confusion_matrix(y_test, predictions)
    report=metrics.classification_report(y_test,predictions)
    print(report)
    truelabels,predictlabels,cm,val_a=[],[],[],[]
    for traini,testi in skf.split(X,Y):
        xtrain,xtest=X[traini],X[testi]
        ytrain,ytest=Y[traini],Y[testi]

        clf.fit(xtrain,ytrain)
        p=clf.predict(xtest)

        truelabels.extend(ytest)
        predictlabels.extend(p)
        val_a.append(metrics.accuracy_score(ytest,p))
        cm.append(metrics.confusion_matrix(ytest,p))
    print("Mean validation acc: "+str(np.mean(val_a)))
    cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix =
sum(cm)/len(cm), display_labels = ['papillon', 'bluetick',
'Dandie_Dinmont','Pembroke'] )
    cm_display.plot()
    plt.show()
```

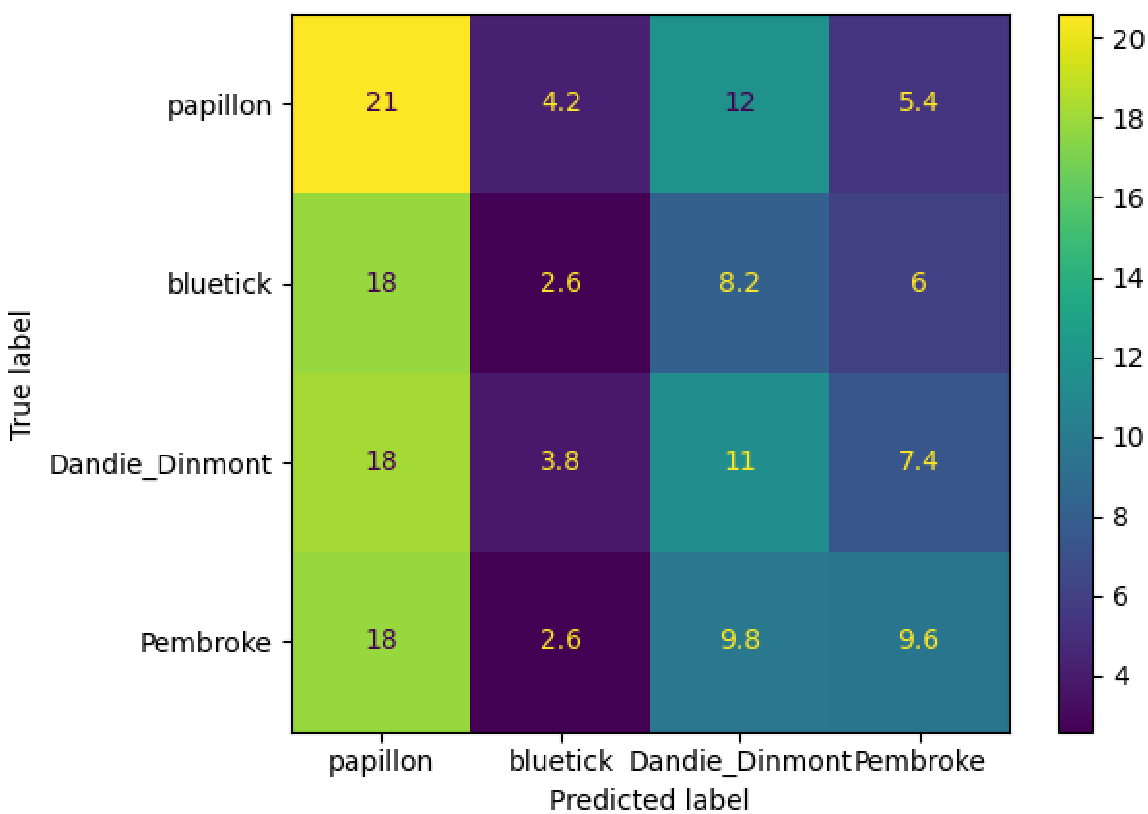
```
DecisionTreeClassifier()
```

```
              precision    recall  f1-score   support

           0       0.16      0.17      0.16        42
           1       0.24      0.23      0.24        35
           2       0.27      0.27      0.27        41
           3       0.28      0.28      0.28        40

    accuracy                           0.23       158
   macro avg       0.24      0.23      0.24       158
weighted avg       0.24      0.23      0.23       158

Mean validation acc: 0.2709505764734338
```




```
MLPClassifier(hidden_layer_sizes=(10, 10, 10))


C:\Users\Bhanu\AppData\Local\Packages\
PythonSoftwareFoundation.Python.3.11_qbz5n2kfra8p0\LocalCache\local-
packages\Python311\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:690: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (200) reached and the optimization
hasn't converged yet.
  warnings.warn(

              precision    recall  f1-score   support

           0       0.31      0.36      0.33        42
           1       0.20      0.23      0.21        35
           2       0.41      0.34      0.37        41
           3       0.39      0.35      0.37        40

    accuracy                           0.32       158
   macro avg       0.33      0.32      0.32       158
weighted avg       0.33      0.32      0.33       158
```

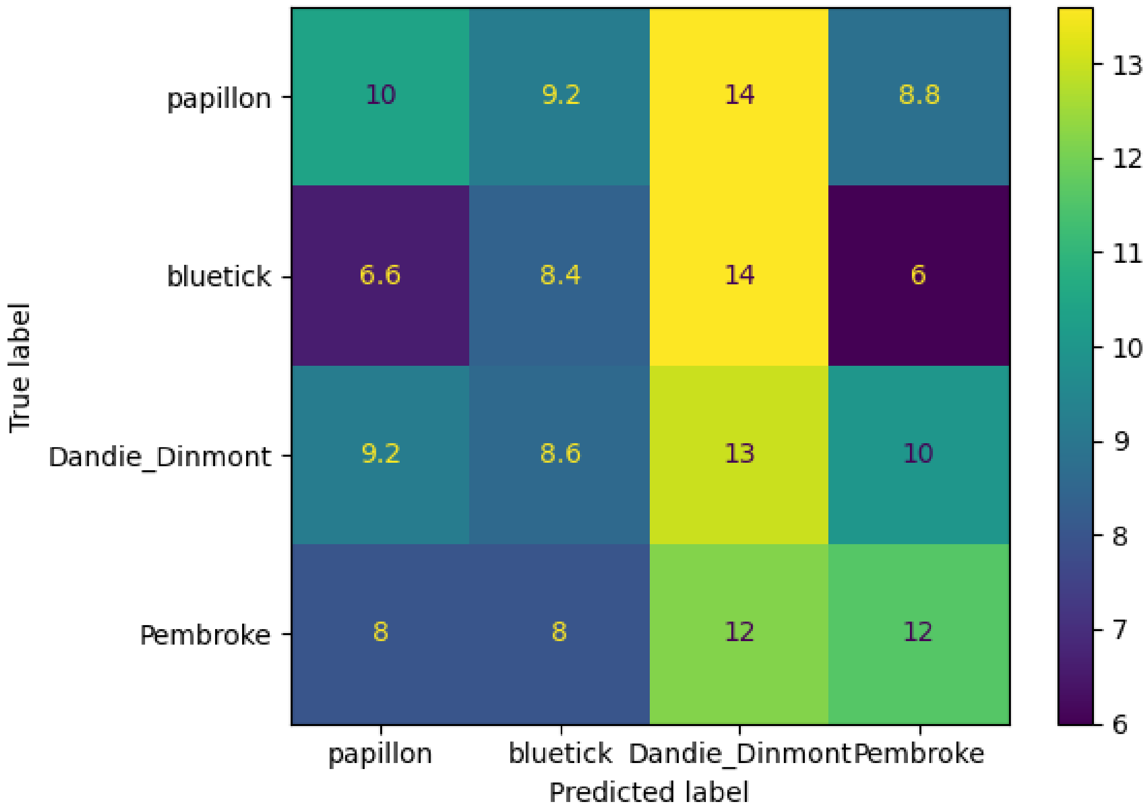
```
C:\Users\Bhanu\AppData\Local\Packages\
PythonSoftwareFoundation.Python.3.11_qbz5n2kfra8p0\LocalCache\local-
packages\Python311\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:690: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (200) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\Bhanu\AppData\Local\Packages\
PythonSoftwareFoundation.Python.3.11_qbz5n2kfra8p0\LocalCache\local-
packages\Python311\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:690: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (200) reached and the optimization
hasn't converged yet.
  warnings.warn(

Mean validation acc: 0.2812061598000483
```




```
AdaBoostClassifier()



C:\Users\Bhanu\AppData\Local\Packages\
PythonSoftwareFoundation.Python.3.11_qbz5n2kfra8p0\LocalCache\local-
packages\Python311\site-packages\sklearn\ensemble\
```

```
_weight_boosting.py:527: FutureWarning: The SAMME.R algorithm (the
default) is deprecated and will be removed in 1.6. Use the SAMME
algorithm to circumvent this warning.
  warnings.warn(

              precision    recall  f1-score   support

           0       0.25      0.36      0.29        42
           1       0.23      0.17      0.20        35
           2       0.23      0.24      0.24        41
           3       0.32      0.23      0.26        40

    accuracy                           0.25       158
   macro avg       0.26      0.25      0.25       158
weighted avg       0.26      0.25      0.25       158


C:\Users\Bhanu\AppData\Local\Packages\
PythonSoftwareFoundation.Python.3.11_qbz5n2kfra8p0\LocalCache\local-
packages\Python311\site-packages\sklearn\ensemble\
_weight_boosting.py:527: FutureWarning: The SAMME.R algorithm (the
default) is deprecated and will be removed in 1.6. Use the SAMME
algorithm to circumvent this warning.
  warnings.warn(
C:\Users\Bhanu\AppData\Local\Packages\
PythonSoftwareFoundation.Python.3.11_qbz5n2kfra8p0\LocalCache\local-
packages\Python311\site-packages\sklearn\ensemble\
_weight_boosting.py:527: FutureWarning: The SAMME.R algorithm (the
default) is deprecated and will be removed in 1.6. Use the SAMME
algorithm to circumvent this warning.
  warnings.warn(
C:\Users\Bhanu\AppData\Local\Packages\
PythonSoftwareFoundation.Python.3.11_qbz5n2kfra8p0\LocalCache\local-
packages\Python311\site-packages\sklearn\ensemble\
_weight_boosting.py:527: FutureWarning: The SAMME.R algorithm (the
default) is deprecated and will be removed in 1.6. Use the SAMME
algorithm to circumvent this warning.
  warnings.warn(
C:\Users\Bhanu\AppData\Local\Packages\
PythonSoftwareFoundation.Python.3.11_qbz5n2kfra8p0\LocalCache\local-
packages\Python311\site-packages\sklearn\ensemble\
_weight_boosting.py:527: FutureWarning: The SAMME.R algorithm (the
default) is deprecated and will be removed in 1.6. Use the SAMME
algorithm to circumvent this warning.
  warnings.warn(
C:\Users\Bhanu\AppData\Local\Packages\
PythonSoftwareFoundation.Python.3.11_qbz5n2kfra8p0\LocalCache\local-
packages\Python311\site-packages\sklearn\ensemble\
_weight_boosting.py:527: FutureWarning: The SAMME.R algorithm (the
default) is deprecated and will be removed in 1.6. Use the SAMME
```

```
algorithm to circumvent this warning.
  warnings.warn(

Mean validation acc: 0.27602193017818266
```




```
By visually comparing the three confusion matrices (on the test set),
which do you think is the best method? Why?
MLP shows better intensity along the diagonal in the confusion matrix,
indicating a higher correct classification rate for each class.

  Cell In[41], line 2
    MLP shows better intensity along the diagonal in the confusion
matrix, indicating a higher correct classification rate for each
class.
        ^
SyntaxError: invalid syntax


Based on the mean validation accuracies (from the 5-fold cross-
validation) for the three methods, which is the best method?
Decision Tree achieved the highest mean validation accuracy,
approximately 28.5% across the 5-fold cross-validation.
```

```
Compute the accuracies for the three methods on the test set. Which is
the best method?
Best Method by Test Accuracy: MLP achieved the highest test accuracy
of 32%, making it the best performer on the test set.

Compute the F-measure for the three methods on the test set. Which is
the best method?
MLP has the highest F-measure on the test set, approximately 0.36,
indicating it maintains a better balance between precision and recall
for this classification task.

from sklearn.svm import LinearSVC
from sklearn.model_selection import StratifiedKFold, KFold
from sklearn.metrics import accuracy_score
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split

class_1, class_2 = 0, 1

# Filter out samples that belong to the two selected classes
mask = (y_train == class_1) | (y_train == class_2)
X_train_two_classes = X_train[mask]
y_train_two_classes = y_train[mask]

# Similarly filter the test set if needed
mask_test = (y_test == class_1) | (y_test == class_2)
X_test_two_classes = X_test[mask_test]
y_test_two_classes = y_test[mask_test]


C_values = [0.1, 1, 10, 100]
standard_kf = KFold(n_splits=5, shuffle=True, random_state=42)
stratified_kf = StratifiedKFold(n_splits=5, shuffle=True,
random_state=42)

# Store errors
standard_train_errors = []
standard_val_errors = []
stratified_train_errors = []
stratified_val_errors = []

# Loop over each C value for both standard and stratified CV
for C in C_values:
    svc = LinearSVC(C=C, random_state=42, max_iter=10000)

    # Standard 5-fold CV
    train_errors_standard = []
    val_errors_standard = []
    for train_index, val_index in
```

```
standard_kf.split(X_train_two_classes):
        X_train_fold, X_val_fold = X_train_two_classes[train_index],
X_train_two_classes[val_index]
        y_train_fold, y_val_fold = y_train_two_classes[train_index],
y_train_two_classes[val_index]

        svc.fit(X_train_fold, y_train_fold)
        train_pred = svc.predict(X_train_fold)
        val_pred = svc.predict(X_val_fold)

        train_errors_standard.append(1 - accuracy_score(y_train_fold,
train_pred))
        val_errors_standard.append(1 - accuracy_score(y_val_fold,
val_pred))

    standard_train_errors.append(np.mean(train_errors_standard) * 100)
    standard_val_errors.append(np.mean(val_errors_standard) * 100)

    # Stratified 5-fold CV
    train_errors_stratified = []
    val_errors_stratified = []
    for train_index, val_index in
stratified_kf.split(X_train_two_classes, y_train_two_classes):
        X_train_fold, X_val_fold = X_train_two_classes[train_index],
X_train_two_classes[val_index]
        y_train_fold, y_val_fold = y_train_two_classes[train_index],
y_train_two_classes[val_index]

        svc.fit(X_train_fold, y_train_fold)
        train_pred = svc.predict(X_train_fold)
        val_pred = svc.predict(X_val_fold)

        train_errors_stratified.append(1 -
accuracy_score(y_train_fold, train_pred))
        val_errors_stratified.append(1 - accuracy_score(y_val_fold,
val_pred))

    stratified_train_errors.append(np.mean(train_errors_stratified) *
100)
    stratified_val_errors.append(np.mean(val_errors_stratified) * 100)

# Plot the error curves
plt.figure(figsize=(10, 6))
plt.plot(C_values, standard_train_errors, label='Standard CV -
Training Error', marker='o')
plt.plot(C_values, standard_val_errors, label='Standard CV -
Validation Error', marker='o')
plt.plot(C_values, stratified_train_errors, label='Stratified CV -
Training Error', marker='o')
plt.plot(C_values, stratified_val_errors, label='Stratified CV -
```

```
Validation Error', marker='o')
plt.xscale('log')
plt.xlabel('C (Regularization Parameter)')
plt.ylabel('Mean Error (%)')
plt.title('Training and Validation Errors for Different C Values')
plt.legend()
plt.show()

Which C has/have the lowest mean error for each curve?
all the four curves have similar mean errors across different values
of C (0.1, 1, 10, 100).
Comments on Model Complexity in relation to C?
In SVM, C controls the strength of regularization. When C values are
low, the model tends to be overly regularized and may end up in
underfitting. When the C values are high, the model would fit training
data closely but may tend to overfit.

There seems to be neither overfitting nor underfitting, as the train
and cross-validation errors dont change appreciably for the whole
range of C.

from sklearn.svm import LinearSVC
from sklearn.metrics import accuracy_score

best_C = 1
best_svc = LinearSVC(C=best_C, random_state=42, max_iter=10000)

best_svc.fit(X_train_two_classes, y_train_two_classes)

test_predictions = best_svc.predict(X_test_two_classes)

test_error = 1 - accuracy_score(y_test_two_classes, test_predictions)
print(f"Test Error with best C ({best_C}): {test_error * 100:.2f}%")
```