All model practice

May 16, 2025

[107]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
import tensorflow as tf
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import CategoricalNB, BernoulliNB, GaussianNB,␣
↪ComplementNB, MultinomialNB
from sklearn.preprocessing import LabelEncoder
from sklearn.datasets import load_iris
from sklearn.metrics import accuracy_score, classification_report,␣
↪mean_squared_error, confusion_matrix, ConfusionMatrixDisplay
```

[54]:
```
iris = load_iris()
```

[55]:
```
data = iris.data
```

[ ]:
```

```

[56]:
```
iris.feature_names
```

[56]:
```
['sepal length (cm)',
 'sepal width (cm)',
 'petal length (cm)',
 'petal width (cm)']
```

[57]:
```
df = pd.DataFrame(data, columns=iris.feature_names)
```

[58]:
```
df
```

[58]:
```
     sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)
0                  5.1               3.5                1.4               0.2
1                  4.9               3.0                1.4               0.2
2                  4.7               3.2                1.3               0.2
3                  4.6               3.1                1.5               0.2
4                  5.0               3.6                1.4               0.2
```

```
..                 …                …                 …                 …
145               6.7              3.0               5.2               2.3
146               6.3              2.5               5.0               1.9
147               6.5              3.0               5.2               2.0
148               6.2              3.4               5.4               2.3
149               5.9              3.0               5.1               1.8

[150 rows x 4 columns]
```

[ ]:

[59]:
```python
df['target'] = iris.target
```

[60]:
```python
df
```

[60]:
```
     sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)  \
0                  5.1               3.5                1.4               0.2
1                  4.9               3.0                1.4               0.2
2                  4.7               3.2                1.3               0.2
3                  4.6               3.1                1.5               0.2
4                  5.0               3.6                1.4               0.2
..                 …                 …                  …                 …
145                6.7               3.0                5.2               2.3
146                6.3               2.5                5.0               1.9
147                6.5               3.0                5.2               2.0
148                6.2               3.4                5.4               2.3
149                5.9               3.0                5.1               1.8

     target
0         0
1         0
2         0
3         0
4         0
..      …
145       2
146       2
147       2
148       2
149       2

[150 rows x 5 columns]
```

[61]:
```python
X= df.drop(columns={'target'})
y= df.target
```

[62]:
```python
X_train, X_test, y_train, y_test= train_test_split(X,y, random_state=42,␣
↪shuffle=False, test_size=0.3)
```

[63]:
```python
X_train.shape
```

[63]:
```
(105, 4)
```

[64]:
```python
X_test.shape
```

[64]:
```
(45, 4)
```

[65]:
```python
y_train.shape
```

[65]:
```
(105,)
```

[66]:
```python
y_test.shape
```

[66]:
```
(45,)
```

1 logistic Regression

Question. Use the Breast Cancer Wisconsin dataset from sklearn.datasets to perform binary classification using Logistic Regression. Your task is to:

1) Import the dataset and perform exploratory analysis.

2) Split the data into training and test sets (80-20 split).

3) Train a logistic regression model.

4) Evaluate the model using accuracy, confusion matrix, and classification report.

5) Write the complete Python code to accomplish this task.

[67]:
```python
from sklearn.datasets import load_breast_cancer
```

[68]:
```python
bc = load_breast_cancer()
```

[69]:
```python
bv = bc.data
```

[70]:
```python
data = pd.DataFrame(bv, columns=bc.feature_names)
```

[71]:
```python
data['target'] = bc.target
```

[72]:
```python
data
```

[72]:
```
     mean radius  mean texture  mean perimeter  mean area  mean smoothness  \
0          17.99         10.38          122.80     1001.0          0.11840
```

1	20.57	17.77	132.90	1326.0	0.08474
2	19.69	21.25	130.00	1203.0	0.10960
3	11.42	20.38	77.58	386.1	0.14250
4	20.29	14.34	135.10	1297.0	0.10030
..	…	…	…	…	…
564	21.56	22.39	142.00	1479.0	0.11100
565	20.13	28.25	131.20	1261.0	0.09780
566	16.60	28.08	108.30	858.1	0.08455
567	20.60	29.33	140.10	1265.0	0.11780
568	7.76	24.54	47.92	181.0	0.05263

	mean compactness	mean concavity	mean concave points	mean symmetry	\
0	0.27760	0.30010	0.14710	0.2419	
1	0.07864	0.08690	0.07017	0.1812	
2	0.15990	0.19740	0.12790	0.2069	
3	0.28390	0.24140	0.10520	0.2597	
4	0.13280	0.19800	0.10430	0.1809	
..	…	…	…	…	
564	0.11590	0.24390	0.13890	0.1726	
565	0.10340	0.14400	0.09791	0.1752	
566	0.10230	0.09251	0.05302	0.1590	
567	0.27700	0.35140	0.15200	0.2397	
568	0.04362	0.00000	0.00000	0.1587	

	mean fractal dimension	…	worst texture	worst perimeter	worst area	\
0	0.07871	…	17.33	184.60	2019.0	
1	0.05667	…	23.41	158.80	1956.0	
2	0.05999	…	25.53	152.50	1709.0	
3	0.09744	…	26.50	98.87	567.7	
4	0.05883	…	16.67	152.20	1575.0	
..	…	…	…	…	…	
564	0.05623	…	26.40	166.10	2027.0	
565	0.05533	…	38.25	155.00	1731.0	
566	0.05648	…	34.12	126.70	1124.0	
567	0.07016	…	39.42	184.60	1821.0	
568	0.05884	…	30.37	59.16	268.6	

	worst smoothness	worst compactness	worst concavity	\
0	0.16220	0.66560	0.7119	
1	0.12380	0.18660	0.2416	
2	0.14440	0.42450	0.4504	
3	0.20980	0.86630	0.6869	
4	0.13740	0.20500	0.4000	
..	…	…	…	
564	0.14100	0.21130	0.4107	
565	0.11660	0.19220	0.3215	
566	0.11390	0.30940	0.3403	

```
567           0.16500           0.86810          0.9387
568           0.08996           0.06444          0.0000

     worst concave points  worst symmetry  worst fractal dimension  target
0                  0.2654          0.4601                  0.11890       0
1                  0.1860          0.2750                  0.08902       0
2                  0.2430          0.3613                  0.08758       0
3                  0.2575          0.6638                  0.17300       0
4                  0.1625          0.2364                  0.07678       0
..                    …              …                       …     …
564                0.2216          0.2060                  0.07115       0
565                0.1628          0.2572                  0.06637       0
566                0.1418          0.2218                  0.07820       0
567                0.2650          0.4087                  0.12400       0
568                0.0000          0.2871                  0.07039       1

[569 rows x 31 columns]
```

[73]:
```
X = data.drop(columns=['target'])
```

[74]:
```
y = data['target']
```

[75]:
```
X_train, X_test, y_train, y_test = train_test_split(X,y, random_state=42,␣
↪shuffle=False, test_size=0.2)
```

[76]:
```
X_train.shape
```

[76]:
```
(455, 30)
```

[77]:
```
y_train.shape
```

[77]:
```
(455,)
```

[78]:
```
log = LogisticRegression(max_iter=10000)
```

[79]:
```
log.fit(X_train, y_train)
```

[79]:
```
LogisticRegression(max_iter=10000)
```

[80]:
```
log.score(X_test, y_test)
```

[80]:
```
0.9298245614035088
```

[81]:
```
y_pred_l = log.predict(X_test)
```

[82]:
```
accuracy_score(y_test, y_pred_l)
```

[82]:
```
0.9298245614035088
```

[83]:
```
print(classification_report(y_test,y_pred_l))
```

```
              precision    recall  f1-score   support

           0       0.78      0.96      0.86        26
           1       0.99      0.92      0.95        88

    accuracy                           0.93       114
   macro avg       0.88      0.94      0.91       114
weighted avg       0.94      0.93      0.93       114
```

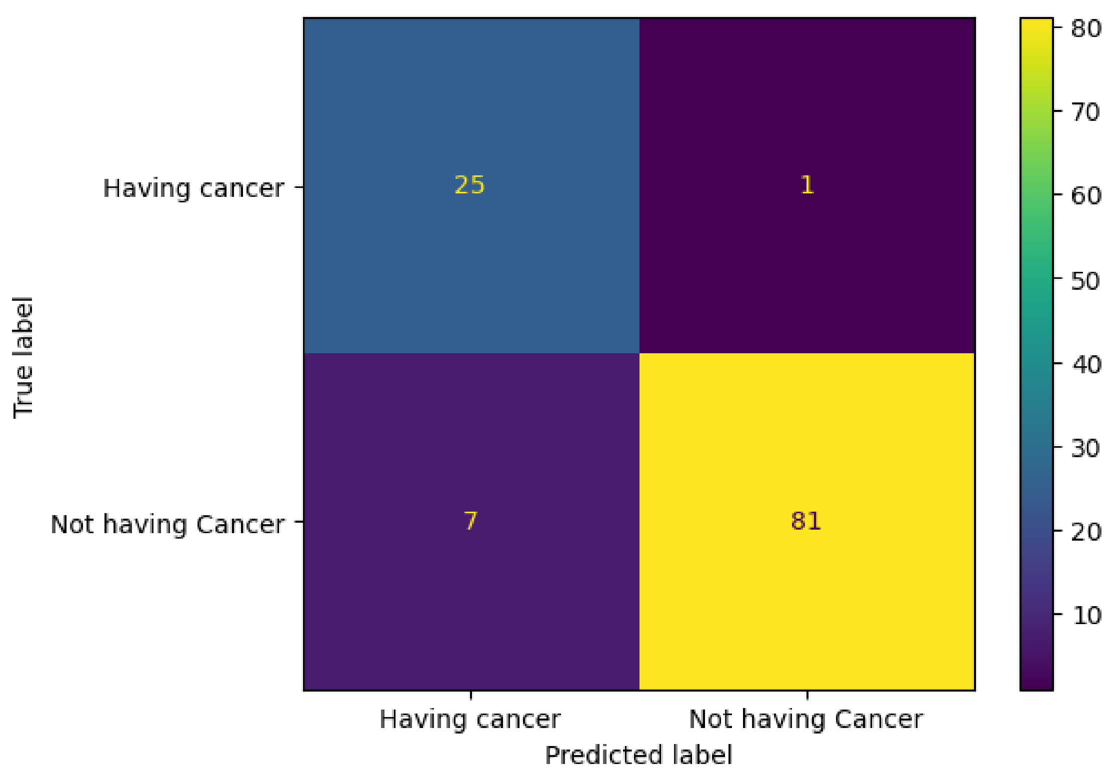
[90]:
```
mse_l = mean_squared_error(y_test,y_pred_l)
mse_l
```

[90]: 0.07017543859649122

[85]:
```
cm = confusion_matrix(y_test, y_pred_l)
```

[89]:
```
label = ['Having cancer', 'Not having Cancer']
cmd = ConfusionMatrixDisplay(cm, display_labels=label)
cmd.plot()
```

[89]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x22528016540>



2 SVM

Use the Iris dataset from sklearn.datasets to classify flower species using a Support Vector Machine (SVM) classifier. Your task is to:

1)Load and explore the dataset.

2)Split the data into training and test sets (70-30 split).

3)Train an SVM classifier with an RBF kernel.

4)Predict the test set results.

5)Evaluate the model using accuracy score and confusion matrix.

6)(Optional) Visualize the decision boundaries using two selected features.

[92]:
```
from sklearn.datasets import load_iris
import pandas as pd
import numpy as np
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, confusion_matrix,␣
    ↪ConfusionMatrixDisplay, mean_squared_error
from sklearn.model_selection import train_test_split
```

[93]:
```
iris = load_iris()
```

[94]:
```
ir = iris.data
```

[95]:
```
data = pd.DataFrame(ir, columns=iris.feature_names)
```

[96]:
```
data
```

[96]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2
..	…	…	…	…
145	6.7	3.0	5.2	2.3
146	6.3	2.5	5.0	1.9
147	6.5	3.0	5.2	2.0
148	6.2	3.4	5.4	2.3
149	5.9	3.0	5.1	1.8

```
[150 rows x 4 columns]
```

[97]:
```python
data['target'] = iris.target
```

[98]:
```python
data
```

[98]:
```
     sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)  \
0                  5.1               3.5                1.4               0.2
1                  4.9               3.0                1.4               0.2
2                  4.7               3.2                1.3               0.2
3                  4.6               3.1                1.5               0.2
4                  5.0               3.6                1.4               0.2
..                 …                 …                  …                 …
145                6.7               3.0                5.2               2.3
146                6.3               2.5                5.0               1.9
147                6.5               3.0                5.2               2.0
148                6.2               3.4                5.4               2.3
149                5.9               3.0                5.1               1.8

     target
0         0
1         0
2         0
3         0
4         0
..      …
145       2
146       2
147       2
148       2
149       2

[150 rows x 5 columns]
```

[99]:
```python
X = data.drop(columns={'target'})
```

[ ]:
```python

```

[100]:
```python
y = data['target']
```

[101]:
```python
X_train, X_test, y_train, y_test = train_test_split(X, y,␣
↪test_size=30,random_state=42)
```

[110]:
```python
svm = SVC(kernel='rbf')
```

[111]:
```python
svm.fit(X_train, y_train)
```

[111]:
```
SVC()
```

[104]:
```
y_pred_s = svm.predict(X_test)
```

[105]:
```
svm.score(X_test, y_test)
```

[105]: 1.0

[106]:
```
accuracy_score(y_test, y_pred_s)
```

[106]: 1.0

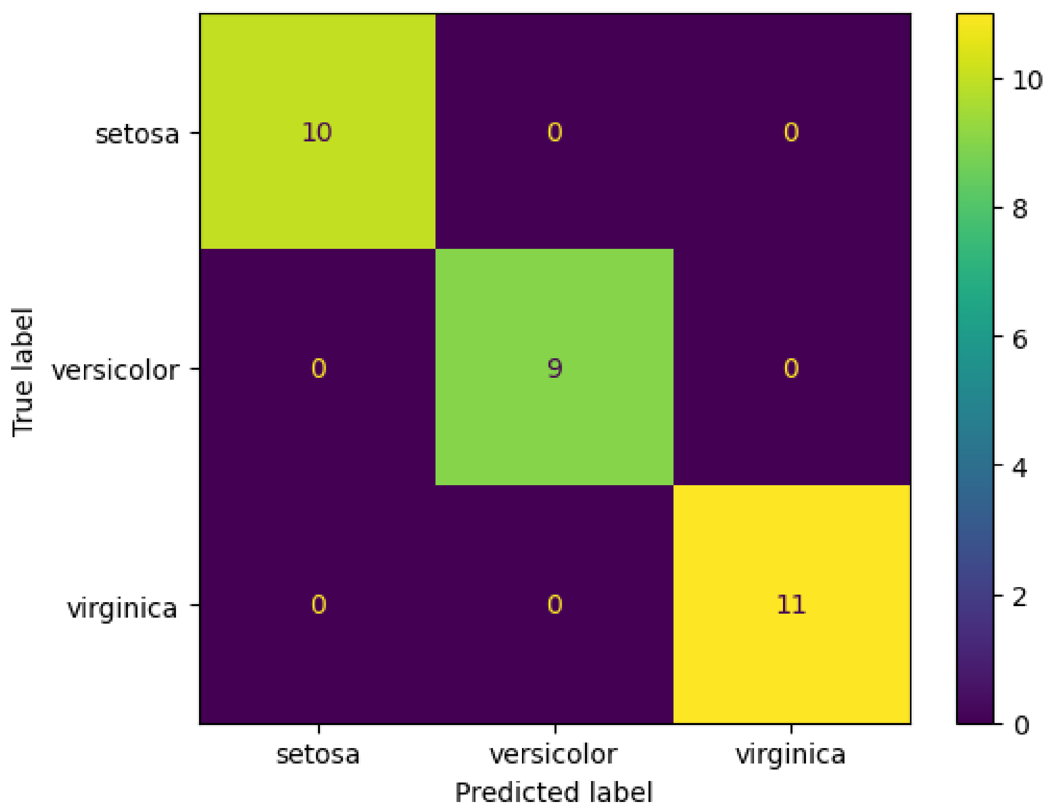
[107]:
```
cm = confusion_matrix(y_test, y_pred_s)
```

[108]:
```
iris.target_names
```

[108]:
```
array(['setosa', 'versicolor', 'virginica'], dtype='<U10')
```

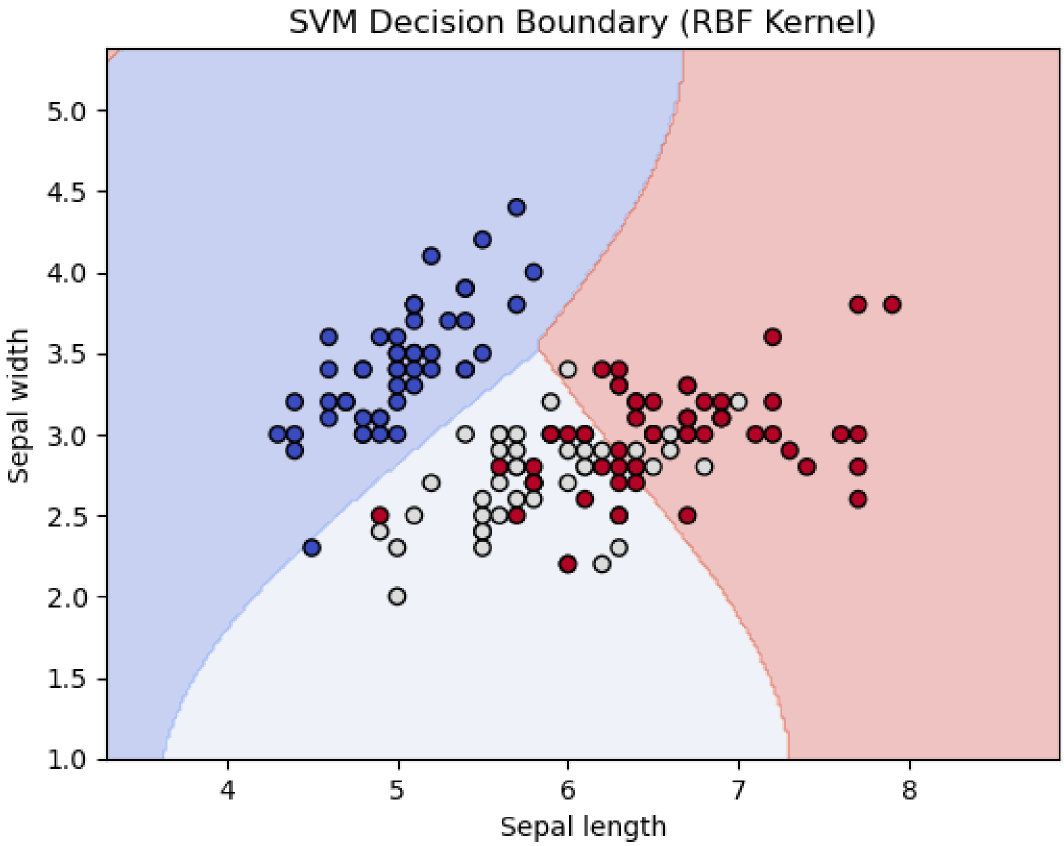
[109]:
```
cmd = ConfusionMatrixDisplay(cm, display_labels=iris.target_names)
cmd.plot()
```

[109]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x23c81dd88c0>
```



[112]:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split

# Load dataset and select two features for visualization
iris = datasets.load_iris()
X = iris.data[:, :2]  # use only sepal length and sepal width
y = iris.target

# Train/test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
↪random_state=42)

# Train SVM
model = SVC(kernel='rbf', gamma='auto')
model.fit(X_train, y_train)

# Create meshgrid
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.02),
                     np.arange(y_min, y_max, 0.02))

# Predict over the grid
Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

# Plotting
plt.contourf(xx, yy, Z, alpha=0.3, cmap=plt.cm.coolwarm)
plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.coolwarm, edgecolors='k')
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')
plt.title('SVM Decision Boundary (RBF Kernel)')
plt.show()
```



3 ANN

Question: Use the Digits dataset from sklearn.datasets to build and evaluate a Neural Network (ANN) for digit classification. Your tasks are:

1) Load and explore the dataset (e.g., image shapes, target labels).

2) Preprocess the data (e.g., scaling the pixel values).

3) Split the data into training and test sets (80-20 split).

4) Create and train a Multi-layer Perceptron (MLP) classifier using MLPClassifier from sklearn.neural_network.

5) Evaluate the model using accuracy, confusion matrix, and classification report.

6) (Optional) Visualize some predicted vs actual digits using matplotlib.

[63]:
```
from sklearn.datasets import load_digits
from sklearn.neural_network import MLPClassifier
```

[64]:
```
from sklearn.model_selection import train_test_split
```

[65]:
```
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, accuracy_score,␣
↪confusion_matrix, ConfusionMatrixDisplay
from sklearn.preprocessing import StandardScaler
import pandas as pd
import numpy as np
```

[66]:
```
ld = load_digits()
```

[67]:
```
ld_v = ld.data
```

[68]:
```
ld_v
```

[68]:
```
array([[ 0.,  0.,  5., …,  0.,  0.,  0.],
       [ 0.,  0.,  0., …, 10.,  0.,  0.],
       [ 0.,  0.,  0., …, 16.,  9.,  0.],
       …,
       [ 0.,  0.,  1., …,  6.,  0.,  0.],
       [ 0.,  0.,  2., …, 12.,  0.,  0.],
       [ 0.,  0., 10., …, 12.,  1.,  0.]])
```

[69]:
```
ss = StandardScaler()
ld_v = ss.fit_transform(ld_v)
```

[70]:
```
ld_v
```

[70]:
```
array([[ 0.        , -0.33501649, -0.04308102, …, -1.14664746,
        -0.5056698 , -0.19600752],
       [ 0.        , -0.33501649, -1.09493684, …,  0.54856067,
        -0.5056698 , -0.19600752],
       [ 0.        , -0.33501649, -1.09493684, …,  1.56568555,
         1.6951369 , -0.19600752],
       …,
       [ 0.        , -0.33501649, -0.88456568, …, -0.12952258,
        -0.5056698 , -0.19600752],
       [ 0.        , -0.33501649, -0.67419451, …,  0.8876023 ,
        -0.5056698 , -0.19600752],
       [ 0.        , -0.33501649,  1.00877481, …,  0.8876023 ,
        -0.26113572, -0.19600752]])
```

[71]:
```
data = pd.DataFrame(ld_v, columns=ld.feature_names)
```

[72]:
```
data['target'] = ld.target
```

[73]:
```
data
```

[73]:
```
      pixel_0_0  pixel_0_1  pixel_0_2  pixel_0_3  pixel_0_4  pixel_0_5  \
0           0.0  -0.335016  -0.043081   0.274072  -0.664478  -0.844129
1           0.0  -0.335016  -1.094937   0.038648   0.268751  -0.138020
2           0.0  -0.335016  -1.094937  -1.844742   0.735366   1.097673
3           0.0  -0.335016   0.377661   0.744919   0.268751  -0.844129
4           0.0  -0.335016  -1.094937  -2.551014  -0.197863  -1.020657
…           …         …         …         …         …         …
1792        0.0  -0.335016  -0.253452  -0.432200   0.268751   0.038508
1793        0.0  -0.335016   0.167290   0.980343   0.268751   0.921145
1794        0.0  -0.335016  -0.884566  -0.196776   0.735366  -0.844129
1795        0.0  -0.335016  -0.674195  -0.432200  -1.131092  -1.020657
1796        0.0  -0.335016   1.008775   0.509495  -0.897785  -0.844129

      pixel_0_6  pixel_0_7  pixel_1_0  pixel_1_1  …  pixel_6_7  pixel_7_0  \
0     -0.409724  -0.125023  -0.059078  -0.624009  …  -0.209785  -0.023596
1     -0.409724  -0.125023  -0.059078  -0.624009  …  -0.209785  -0.023596
2     -0.409724  -0.125023  -0.059078  -0.624009  …  -0.209785  -0.023596
3     -0.409724  -0.125023  -0.059078   1.879691  …  -0.209785  -0.023596
4     -0.409724  -0.125023  -0.059078  -0.624009  …  -0.209785  -0.023596
…           …         …         …         …  …         …         …
1792  -0.409724  -0.125023  -0.059078  -0.311047  …  -0.209785  -0.023596
1793  -0.108958  -0.125023  -0.059078  -0.624009  …  -0.209785  -0.023596
1794  -0.409724  -0.125023  -0.059078  -0.624009  …  -0.209785  -0.023596
1795  -0.409724  -0.125023  -0.059078  -0.624009  …  -0.209785  -0.023596
1796  -0.409724  -0.125023  -0.059078   0.001916  …  -0.209785  -0.023596

      pixel_7_1  pixel_7_2  pixel_7_3  pixel_7_4  pixel_7_5  pixel_7_6  \
0     -0.299081   0.086719   0.208293  -0.366771  -1.146647  -0.505670
1     -0.299081  -1.089383  -0.249010   0.849632   0.548561  -0.505670
2     -0.299081  -1.089383  -2.078218  -0.164037   1.565686   1.695137
3     -0.299081   0.282736   0.208293   0.241430   0.379040  -0.505670
4     -0.299081  -1.089383  -2.306869   0.849632  -0.468564  -0.505670
…           …         …         …         …         …         …
1792  -0.299081  -0.697349   0.436944   0.646898   0.379040  -0.505670
1793  -0.299081   0.086719   0.894246   0.444164  -0.129523  -0.505670
1794  -0.299081  -0.697349  -0.706312   0.241430  -0.129523  -0.505670
1795  -0.299081  -0.109298  -0.020358   0.849632   0.887602  -0.505670
1796   0.771535   0.478753  -0.020358   0.444164   0.887602  -0.261136

      pixel_7_7  target
0     -0.196008       0
1     -0.196008       1
2     -0.196008       2
3     -0.196008       3
4     -0.196008       4
```

```
…          …      …
1792  -0.196008      9
1793  -0.196008      0
1794  -0.196008      8
1795  -0.196008      9
1796  -0.196008      8

[1797 rows x 65 columns]
```

[74]:

```
X = data.drop(columns={'target'})
y = data['target']
```

[75]:

```
X_train,X_test, y_train, y_test = train_test_split(X,y,random_state=42,␣
↪test_size=0.2)
```

[109]:

```
mlp = MLPClassifier(early_stopping=True,learning_rate_init=0.01,verbose=True)
```

[110]:

```
mlp.fit(X_train, y_train)
```

```
Iteration 1, loss = 1.25186916
Validation score: 0.909722
Iteration 2, loss = 0.26145032
Validation score: 0.965278
Iteration 3, loss = 0.12828842
Validation score: 0.958333
Iteration 4, loss = 0.07893738
Validation score: 0.958333
Iteration 5, loss = 0.04699669
Validation score: 0.937500
Iteration 6, loss = 0.03117986
Validation score: 0.965278
Iteration 7, loss = 0.02187479
Validation score: 0.979167
Iteration 8, loss = 0.01571871
Validation score: 0.979167
Iteration 9, loss = 0.01210044
Validation score: 0.972222
Iteration 10, loss = 0.00954561
Validation score: 0.972222
Iteration 11, loss = 0.00818447
Validation score: 0.972222
Iteration 12, loss = 0.00680873
Validation score: 0.979167
Iteration 13, loss = 0.00588970
Validation score: 0.979167
Iteration 14, loss = 0.00516508
Validation score: 0.979167
Iteration 15, loss = 0.00459044
```

```
Validation score: 0.979167
Iteration 16, loss = 0.00418566
Validation score: 0.972222
Iteration 17, loss = 0.00379488
Validation score: 0.979167
Iteration 18, loss = 0.00343051
Validation score: 0.972222
Validation score did not improve more than tol=0.000100 for 10 consecutive
epochs. Stopping.
```

[110]: MLPClassifier(early_stopping=True, learning_rate_init=0.01, verbose=True)

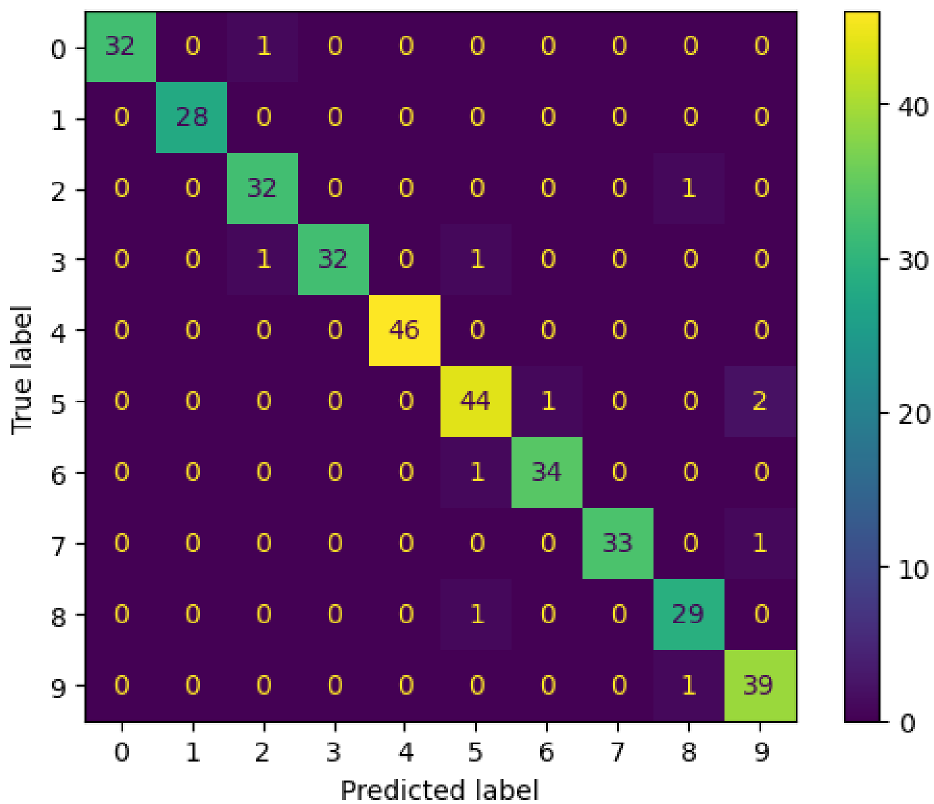
[111]:
```
y_pred_m = mlp.predict(X_test)
```

[112]:
```
print(accuracy_score(y_test, y_pred_m))
```

```
0.9694444444444444
```

[113]:
```
cm = confusion_matrix(y_test, y_pred_m)
cmd = ConfusionMatrixDisplay(cm,display_labels=ld.target_names)
cmd.plot()
```

[113]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1ed0dc008f0>



[114]:
```
a_V = pd.DataFrame({'actual':y_test,
                    'predicted':y_pred_m})
```

[115]:
```
a_V
```

[115]:

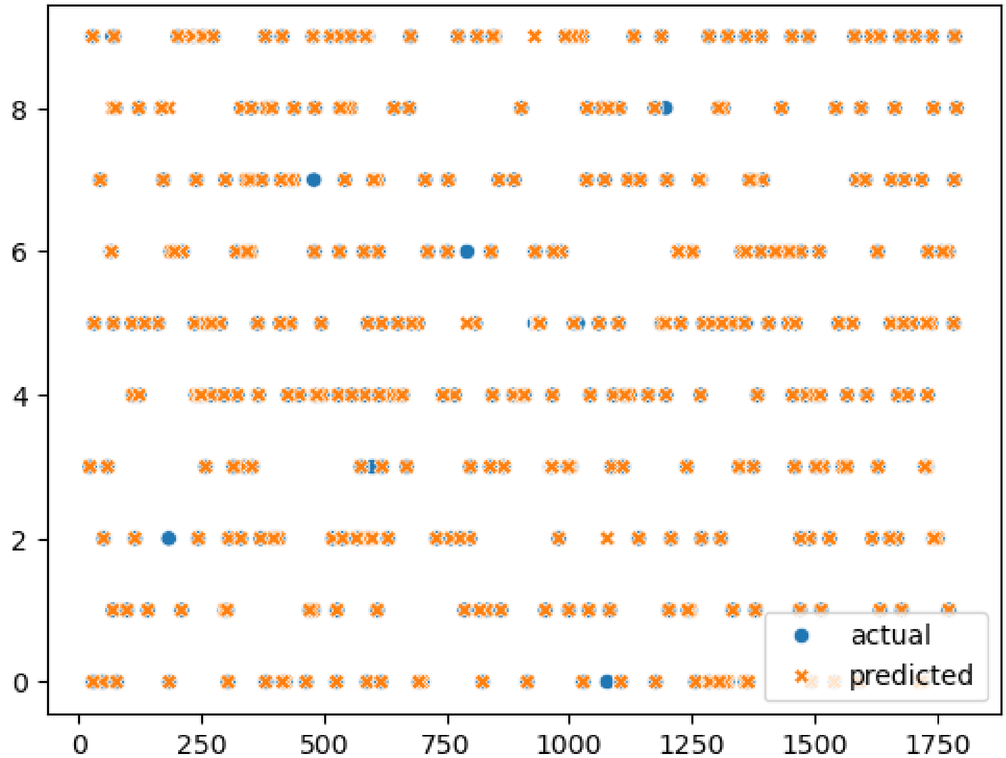
	actual	predicted
1245	6	6
220	9	9
1518	3	3
438	7	7
1270	2	2
…	…	…
1731	4	4
1630	3	3
1037	8	8
965	3	3
1461	5	5

[360 rows x 2 columns]

[116]:
```
import matplotlib.pyplot as plt
import seaborn as sns
```

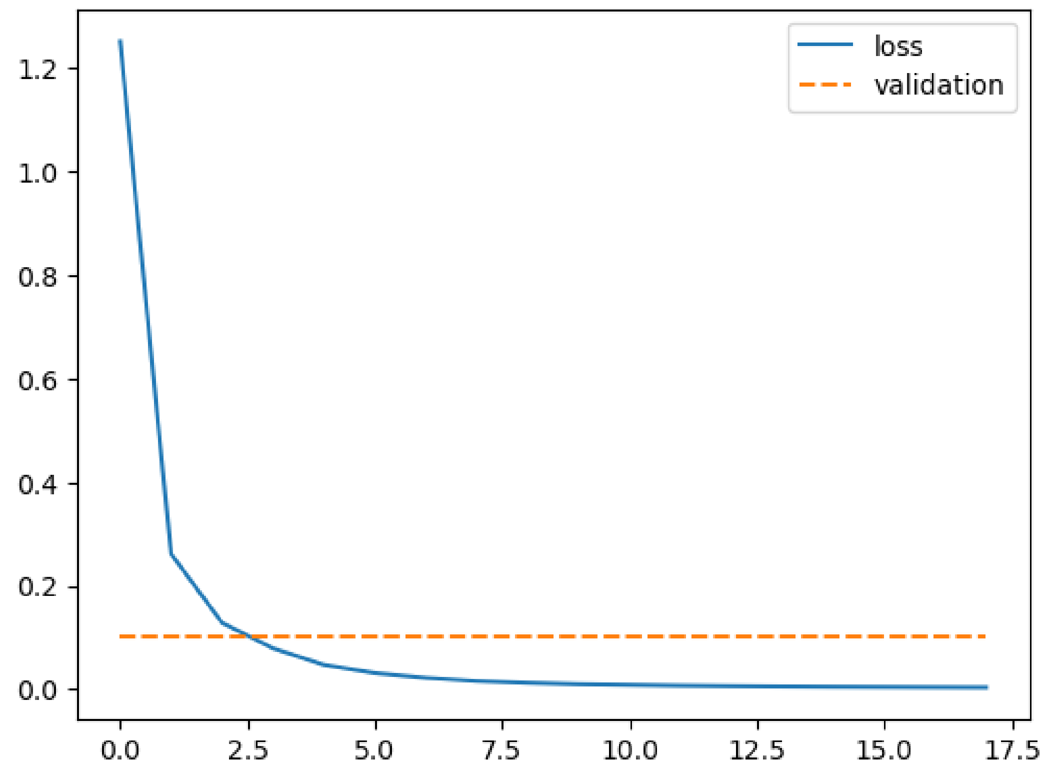
[117]:
```
sns.scatterplot(a_V)
```

[117]: <Axes: >



[118]:
```
loss = mlp.loss_curve_
validation = mlp.validation_fraction
l_v = pd.DataFrame({'loss' : loss,
                    'validation' : validation})
sns.lineplot(l_v)
```

[118]: <Axes: >



4 Naive Bayes classifier

Question: Use the 20 Newsgroups dataset from sklearn.datasets to build a text classification model using a Naive Bayes classifier. Your tasks are:

1) Load the 20newsgroups dataset with a subset of categories (e.g., 'sci.space', 'rec.sport.baseball', 'comp.graphics').

2) Preprocess the text data using TF-IDF vectorization.

3) Split the data into training and test sets (80-20 split).

4) Train a Multinomial Naive Bayes classifier.

5) Evaluate the model using accuracy, confusion matrix, and classification report.

6) (Optional) Print a few sample predictions with their actual categories.

[1]:
```
import pandas as pd
import numpy as np
from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import TfidfVectorizer
```

```python
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, confusion_matrix,␣
↪ConfusionMatrixDisplay, classification_report
import matplotlib.pyplot as plt
import seaborn as sns
import re
```

[2]:
```python
categories = ['sci.space', 'rec.sport.baseball', 'comp.graphics']

newsgroups_data = fetch_20newsgroups(subset = 'all' , categories=categories,␣
↪remove=('headers', 'footers', 'quotes'))
```

[3]:
```python
tf = TfidfVectorizer()
```

[4]:
```python
X = tf.fit_transform(newsgroups_data.data)
```

[5]:
```python
y = newsgroups_data.target
```

[6]:
```python
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42,␣
↪test_size=0.2)
```

[7]:
```python
nb = MultinomialNB()
```

[8]:
```python
nb.fit(X_train, y_train)
```

[8]:
```
MultinomialNB()
```

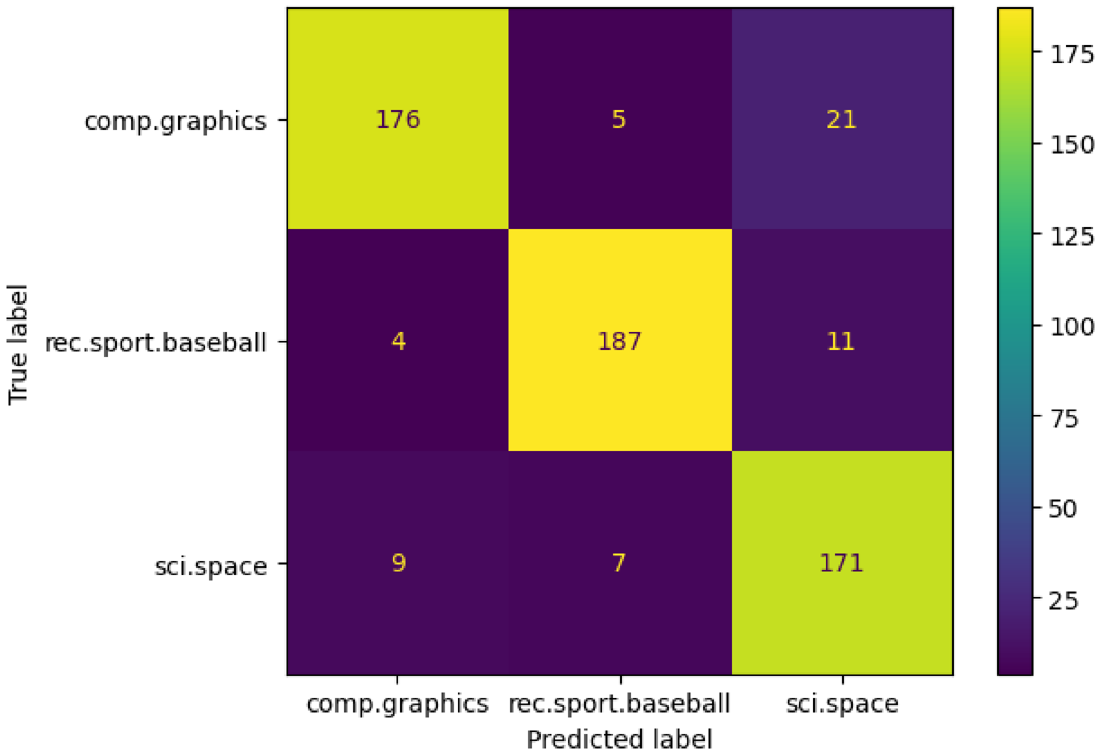
[9]:
```python
y_pred_n = nb.predict(X_test)
```

[10]:
```python
accuracy_score(y_test, y_pred_n)
```

[10]:
```
0.9035532994923858
```

[11]:
```python
cm = confusion_matrix(y_test, y_pred_n)
cmd = ConfusionMatrixDisplay(cm, display_labels=newsgroups_data.target_names)
cmd.plot()
```

[11]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x2d6abbc8710>
```



[13]:
```
print(classification_report(y_test, y_pred_n))
```

```
              precision    recall  f1-score   support

           0       0.93      0.87      0.90       202
           1       0.94      0.93      0.93       202
           2       0.84      0.91      0.88       187

    accuracy                           0.90       591
   macro avg       0.90      0.90      0.90       591
weighted avg       0.91      0.90      0.90       591
```

[16]:
```
for i in range(5):
    print(newsgroups_data.data[i])
    print("Actual:", newsgroups_data.target_names[y_test[i]])
    print("Predicted:", newsgroups_data.target_names[y_pred_n[i]])
```

```
I know it's only wishful thinking, with our current President,
but this is from last fall:

     "Is there life on Mars?  Maybe not now.  But there will be."
        -- Daniel S. Goldin, NASA Administrator, 24 August 1992
```

-- Ken Jenks, NASA/JSC/GM2, Space Shuttle Program Office
      kjenks@gothamcity.jsc.nasa.gov (713) 483-4368
Actual: rec.sport.baseball
Predicted: rec.sport.baseball

Here's one I remember: (sort of)
Yogi's asleep in a hotel room late at night and gets a call from someone.
After he answers the phone the person at the other end asks if he woke Yogi
up. Yogi answered, "No, the phone did."
Actual: comp.graphics
Predicted: comp.graphics
Archive-name: space/acronyms
Edition: 8

Acronym List for sci.astro, sci.space, and sci.space.shuttle:
Edition 8, 1992 Dec 7
Last posted: 1992 Aug 27

This list is offered as a reference for translating commonly appearing acronyms in the space-related newsgroups. If I forgot or botched your favorite acronym, please let me know! Also, if there's an acronym *not* on this list that confuses you, drop me a line, and if I can figure it out, I'll add it to the list.

Note that this is intended to be a reference for *frequently seen* acronyms, and is most emphatically *not* encyclopedic. If I incorporated every acronym I ever saw, I'd soon run out of disk space! :-)

The list will be posted at regular intervals, every 30 days. All comments regarding it are welcome; I'm reachable as bradfrd2@ncar.ucar.edu.

Note that this just tells what the acronyms stand for -- you're on your own for figuring out what they *mean*! Note also that the total number of acronyms in use far exceeds what I can list; special-purpose acronyms that are essentially always explained as they're introduced are omitted. Further, some acronyms stand for more than one thing; as of Edition 3 of the list, these acronyms appear on multiple lines, unless they're simply different ways of referring to the same thing.

Thanks to everybody who's sent suggestions since the first version of the list, and especially to Garrett A. Wollman (wollman@griffin.uvm.edu), who is maintaining an independent list, somewhat more verbose in character than mine, and to Daniel Fischer (dfi@specklec.mpifr-bonn.mpg.de), who is maintaining a truly HUGE list (535 at last count) of acronyms and terms, mostly in German (which I read, fortunately).

Special thanks this time to Ken Hollis at NASA, who sent me a copy of NASA

Reference Publication 1059 Revised: _Space Transportation System and Associated Payloads: Glossary, Acronyms, and Abbreviations_, a truly mammoth tome -- almost 300 pages of TLAs.

Special Bonus! At the end of this posting, you will find a perl program written by none other than Larry Wall, whose purpose is to scramble the acronym list in an entertaining fashion. Thanks, Larry!

A&A: Astronomy and Astrophysics
AAO: Anglo-Australian Observatory
AAS: American Astronomical Society
AAS: American Astronautical Society
AAVSO: American Association of Variable Star Observers
ACE: Advanced Composition Explorer
ACRV: Assured Crew Return Vehicle (or) Astronaut Crew Rescue Vehicle
ADFRF: Ames-Dryden Flight Research Facility (was DFRF) (NASA)
AGN: Active Galactic Nucleus
AGU: American Geophysical Union
AIAA: American Institute of Aeronautics and Astronautics
AIPS: Astronomical Image Processing System
AJ: Astronomical Journal
ALEXIS: Array of Low Energy X-ray Imaging Sensors
ALPO: Association of Lunar and Planetary Observers
ALS: Advanced Launch System
ANSI: American National Standards Institute
AOA: Abort Once Around (Shuttle abort plan)
AOCS: Attitude and Orbit Control System
Ap.J: Astrophysical Journal
APM: Attached Pressurized Module (a.k.a. Columbus)
APU: Auxiliary Power Unit
ARC: Ames Research Center (NASA)
ARTEMIS: Advanced Relay TEchnology MISsion
ASA: Astronomical Society of the Atlantic
ASI: Agenzia Spaziale Italiano
ASRM: Advanced Solid Rocket Motor
ATDRS: Advanced Tracking and Data Relay Satellite
ATLAS: Atmospheric Laboratory for Applications and Science
ATM: Amateur Telescope Maker
ATO: Abort To Orbit (Shuttle abort plan)
AU: Astronomical Unit
AURA: Association of Universities for Research in Astronomy
AW&ST: Aviation Week and Space Technology (a.k.a. AvLeak)
AXAF: Advanced X-ray Astrophysics Facility
BATSE: Burst And Transient Source Experiment (on CGRO)
BBXRT: Broad-Band X-Ray Telescope (ASTRO package)
BEM: Bug-Eyed Monster
BH: Black Hole
BIMA: Berkeley Illinois Maryland Array

BNSC: British National Space Centre
BTW: By The Way
C&T: Communications & Tracking
CCAFS: Cape Canaveral Air Force Station
CCD: Charge-Coupled Device
CCDS: Centers for the Commercial Development of Space
CD-ROM: Compact Disk Read-Only Memory
CFA: Center For Astrophysics
CFC: ChloroFluoroCarbon
CFF: Columbus Free Flyer
CFHT: Canada-France-Hawaii Telescope
CGRO: (Arthur Holley) Compton Gamma Ray Observatory (was GRO)
CHARA: Center for High Angular Resolution Astronomy
CIRRIS: Cryogenic InfraRed Radiance Instrument for Shuttle
CIT: Circumstellar Imaging Telescope
CM: Command Module (Apollo spacecraft)
CMCC: Central Mission Control Centre (ESA)
CNES: Centre National d'Etude Spatiales
CNO: Carbon-Nitrogen-Oxygen
CNSR: Comet Nucleus Sample Return
COBE: COsmic Background Explorer
COMPTEL: COMPton TELescope (on CGRO)
COSTAR: Corrective Optics Space Telescope Axial Replacement
CRAF: Comet Rendezvous / Asteroid Flyby
CRRES: Combined Release / Radiation Effects Satellite
CSM: Command and Service Module (Apollo spacecraft)
CSTC: Consolidated Satellite Test Center (USAF)
CTIO: Cerro Tololo Interamerican Observatory
DCX: Delta Clipper eXperimental
DDCU: DC-to-DC Converter Unit
DFRF: Dryden Flight Research Facility (now ADFRF)
DMSP: Defense Meteorological Satellite Program
DOD: Department Of Defense (sometimes DoD)
DOE: Department Of Energy
DOT: Department Of Transportation
DSCS: Defense Satellite Communications System
DSN: Deep Space Network
DSP: Defense Support Program (USAF/NRO)
EAFB: Edwards Air Force Base
ECS: Environmental Control System
EDO: Extended Duration Orbiter
EGRET: Energetic Gamma Ray Experiment Telescope (on CGRO)
EJASA: Electronic Journal of the Astronomical Society of the Atlantic
ELV: Expendable Launch Vehicle
EMU: Extravehicular Mobility Unit
EOS: Earth Observing System
ERS: Earth Resources Satellite (as in ERS-1)
ESA: European Space Agency

ESO: European Southern Observatory
ET: (Shuttle) External Tank
ETLA: Extended Three Letter Acronym
ETR: Eastern Test Range
EUV: Extreme UltraViolet
EUVE: Extreme UltraViolet Explorer
EVA: ExtraVehicular Activity
FAQ: Frequently Asked Questions
FAST: Fast Auroral SnapshoT explorer
FFT: Fast Fourier Transform
FGS: Fine Guidance Sensors (on HST)
FHST: Fixed Head Star Trackers (on HST)
FIR: Far InfraRed
FITS: Flexible Image Transport System
FOC: Faint Object Camera (on HST)
FOS: Faint Object Spectrograph (on HST)
FRR: Flight-Readiness Review
FTP: File Transfer Protocol
FTS: Flight Telerobotic Servicer
FUSE: Far Ultraviolet Spectroscopic Explorer
FWHM: Full Width at Half Maximum
FYI: For Your Information
GAS: Get-Away Special
GBT: Green Bank Telescope
GCVS: General Catalog of Variable Stars
GEM: Giotto Extended Mission
GEO: Geosynchronous Earth Orbit
GDS: Great Dark Spot
GHRS: Goddard High Resolution Spectrograph (on HST)
GIF: Graphics Interchange Format
GLOMR: Global Low-Orbiting Message Relay
GMC: Giant Molecular Cloud
GMRT: Giant Meter-wave Radio Telescope
GMT: Greenwich Mean Time (also called UT)
GOES: Geostationary Orbiting Environmental Satellite
GOX: Gaseous OXygen
GPC: General Purpose Computer
GPS: Global Positioning System
GRO: Gamma Ray Observatory (now CGRO)
GRS: Gamma Ray Spectrometer (on Mars Observer)
GRS: Great Red Spot
GSC: Guide Star Catalog (for HST)
GSFC: Goddard Space Flight Center (NASA)
GTO: Geostationary Transfer Orbit
HAO: High Altitude Observatory
HD: Henry Draper catalog entry
HEAO: High Energy Astronomical Observatory
HeRA: Hermes Robotic Arm

HF: High Frequency
HGA: High Gain Antenna
HLC: Heavy Lift Capability
HLV: Heavy Lift Vehicle
HMC: Halley Multicolor Camera (on Giotto)
HR: Hertzsprung-Russell (diagram)
HRI: High Resolution Imager (on ROSAT)
HSP: High Speed Photometer (on HST)
HST: Hubble Space Telescope
HUT: Hopkins Ultraviolet Telescope (ASTRO package)
HV: High Voltage
IAPPP: International Amateur/Professional Photoelectric Photometry
IAU: International Astronomical Union
IAUC: IAU Circular
ICE: International Cometary Explorer
IDA: International Dark-sky Association
IDL: Interactive Data Language
IGM: InterGalactic Medium
IGY: International Geophysical Year
IMHO: In My Humble Opinion
IOTA: Infrared-Optical Telescope Array
IOTA: International Occultation Timing Association
IPS: Inertial Pointing System
IR: InfraRed
IRAF: Image Reduction and Analysis Facility
IRAS: InfraRed Astronomical Satellite
ISAS: Institute of Space and Astronautical Science (Japan)
ISM: InterStellar Medium
ISO: Infrared Space Observatory
ISO: International Standards Organization
ISPM: International Solar Polar Mission (now Ulysses)
ISY: International Space Year
IUE: International Ultraviolet Explorer
IUS: Inertial Upper Stage
JEM: Japanese Experiment Module (for SSF)
JGR: Journal of Geophysical Research
JILA: Joint Institute for Laboratory Astrophysics
JPL: Jet Propulsion Laboratory
JSC: Johnson Space Center (NASA)
KAO: Kuiper Airborne Observatory
KPNO: Kitt Peak National Observatory
KSC: Kennedy Space Center (NASA)
KTB: Cretaceous-Tertiary Boundary (from German)
LANL: Los Alamos National Laboratory
LaRC: Langley Research Center (NASA)
LDEF: Long Duration Exposure Facility
LEM: Lunar Excursion Module (a.k.a. LM) (Apollo spacecraft)
LEO: Low Earth Orbit

LeRC: Lewis Research Center (NASA)
LEST: Large Earth-based Solar Telescope
LFSA: List of Frequently Seen Acronyms (!)
LGA: Low Gain Antenna
LGM: Little Green Men
LH: Liquid Hydrogen (also LH2 or LHX)
LLNL: Lawrence-Livermore National Laboratory
LM: Lunar Module (a.k.a. LEM) (Apollo spacecraft)
LMC: Large Magellanic Cloud
LN2: Liquid N2 (Nitrogen)
LOX: Liquid OXygen
LRB: Liquid Rocket Booster
LSR: Local Standard of Rest
LTP: Lunar Transient Phenomenon
MB: Manned Base
MCC: Mission Control Center
MECO: Main Engine CutOff
MMH: MonoMethyl Hydrazine
MMT: Multiple Mirror Telescope
MMU: Manned Maneuvering Unit
MNRAS: Monthly Notices of the Royal Astronomical Society
MOC: Mars Observer Camera (on Mars Observer)
MOL: Manned Orbiting Laboratory
MOLA: Mars Observer Laser Altimeter (on Mars Observer)
MOMV: Manned Orbital Maneuvering Vehicle
MOTV: Manned Orbital Transfer Vehicle
MPC: Minor Planets Circular
MRSR: Mars Rover and Sample Return
MRSRM: Mars Rover and Sample Return Mission
MSFC: (George C.) Marshall Space Flight Center (NASA)
MTC: Man Tended Capability
NACA: National Advisory Committee on Aeronautics (became NASA)
NASA: National Aeronautics and Space Administration
NASDA: NAtional Space Development Agency (Japan)
NASM: National Air and Space Museum
NASP: National AeroSpace Plane
NBS: National Bureau of Standards (now NIST)
NDV: NASP Derived Vehicle
NERVA: Nuclear Engine for Rocket Vehicle Application
NGC: New General Catalog
NICMOS: Near Infrared Camera / Multi Object Spectrometer (HST upgrade)
NIMS: Near-Infrared Mapping Spectrometer (on Galileo)
NIR: Near InfraRed
NIST: National Institute for Standards and Technology (was NBS)
NLDP: National Launch Development Program
NOAA: National Oceanic and Atmospheric Administration
NOAO: National Optical Astronomy Observatories
NRAO: National Radio Astronomy Observatory

NRO: National Reconnaissance Office
NS: Neutron Star
NSA: National Security Agency
NSF: National Science Foundation
NSO: National Solar Observatory
NSSDC: National Space Science Data Center
NTR: Nuclear Thermal Rocket(ry)
NTT: New Technology Telescope
OAO: Orbiting Astronomical Observatory
OCST: Office of Commercial Space Transportation
OMB: Office of Management and Budget
OMS: Orbital Maneuvering System
OPF: Orbiter Processing Facility
ORFEUS: Orbiting and Retrievable Far and Extreme Ultraviolet Spectrometer
OSC: Orbital Sciences Corporation
OSCAR: Orbiting Satellite Carrying Amateur Radio
OSSA: Office of Space Science and Applications
OSSE: Oriented Scintillation Spectrometer Experiment (on CGRO)
OTA: Optical Telescope Assembly (on HST)
OTHB: Over The Horizon Backscatter
OTV: Orbital Transfer Vehicle
OV: Orbital Vehicle
PAM: Payload Assist Module
PAM-D: Payload Assist Module, Delta-class
PI: Principal Investigator
PLSS: Portable Life Support System
PM: Pressurized Module
PMC: Permanently Manned Capability
PMIRR: Pressure Modulated InfraRed Radiometer (on Mars Observer)
PMT: PhotoMultiplier Tube
PSF: Point Spread Function
PSR: PulSaR
PV: Photovoltaic
PVO: Pioneer Venus Orbiter
QSO: Quasi-Stellar Object
RCI: Rodent Cage Interface (for SLS mission)
RCS: Reaction Control System
REM: Rat Enclosure Module (for SLS mission)
RF: Radio Frequency
RFI: Radio Frequency Interference
RIACS: Research Institute for Advanced Computer Science
RMS: Remote Manipulator System
RNGC: Revised New General Catalog
ROSAT: ROentgen SATellite
ROUS: Rodents Of Unusual Size (I don't believe they exist)
RSN: Real Soon Now
RTG: Radioisotope Thermoelectric Generator
RTLS: Return To Launch Site (Shuttle abort plan)

SAA: South Atlantic Anomaly
SAGA: Solar Array Gain Augmentation (for HST)
SAMPEX: Solar Anomalous and Magnetospheric Particle EXplorer
SAO: Smithsonian Astrophysical Observatory
SAR: Search And Rescue
SAR: Synthetic Aperture Radar
SARA: Satellite pour Astronomie Radio Amateur
SAREX: Search and Rescue Exercise
SAREX: Shuttle Amateur Radio Experiment
SAS: Space Activity Suit
SAS: Space Adaptation Syndrome
SAT: Synthetic Aperture Telescope
S/C: SpaceCraft
SCA: Shuttle Carrier Aircraft
SCT: Schmidt-Cassegrain Telescope
SDI: Strategic Defense Initiative
SDIO: Strategic Defense Initiative Organization
SEI: Space Exploration Initiative
SEST: Swedish ESO Submillimeter Telescope
SETI: Search for ExtraTerrestrial Intelligence
SID: Sudden Ionospheric Disturbance
SIR: Shuttle Imaging Radar
SIRTF: Space (formerly Shuttle) InfraRed Telescope Facility
SL: SpaceLab
SLAR: Side-Looking Airborne Radar
SLC: Space Launch Complex
SLS: Space(lab) Life Sciences
SMC: Small Magellanic Cloud
SME: Solar Mesosphere Explorer
SMEX: SMall EXplorers
SMM: Solar Maximum Mission
SN: SuperNova (e.g., SN1987A)
SNR: Signal to Noise Ratio
SNR: SuperNova Remnant
SNU: Solar Neutrino Units
SOFIA: Stratospheric Observatory For Infrared Astronomy
SOHO: SOlar Heliospheric Observatory
SPAN: Space Physics and Analysis Network
SPDM: Special Purpose Dextrous Manipulator
SPOT: Systeme Probatoire pour l'Observation de la Terre
SPS: Solar Power Satellite
SRB: Solid Rocket Booster
SRM: Solid Rocket Motor
SSF: Space Station Fred (er, Freedom)
SSI: Solid-State Imager (on Galileo)
SSI: Space Studies Institut
SSME: Space Shuttle Main Engine
SSPF: Space Station Processing Facility

SSRMS: Space Station Remote Manipulator System
SST: Spectroscopic Survey Telescope
SST: SuperSonic Transport
SSTO: Single Stage To Orbit
STIS: Space Telescope Imaging Spectrometer (to replace FOC and GHRS)
STS: Shuttle Transport System (or) Space Transportation System
STScI: Space Telescope Science Institute
SWAS: Submillimeter Wave Astronomy Satellite
SWF: ShortWave Fading
TAL: Transatlantic Abort Landing (Shuttle abort plan)
TAU: Thousand Astronomical Unit (mission)
TCS: Thermal Control System
TDRS: Tracking and Data Relay Satellite
TDRSS: Tracking and Data Relay Satellite System
TES: Thermal Emission Spectrometer (on Mars Observer)
TIROS: Television InfraRed Observation Satellite
TLA: Three Letter Acronym
TOMS: Total Ozone Mapping Spectrometer
TPS: Thermal Protection System
TSS: Tethered Satellite System
UARS: Upper Atmosphere Research Satellite
UBM: Unpressurized Berthing Mechanism
UDMH: Unsymmetrical DiMethyl Hydrazine
UFO: Unidentified Flying Object
UGC: Uppsala General Catalog
UHF: Ultra High Frequency
UIT: Ultraviolet Imaging Telescope (Astro package)
UKST: United Kingdom Schmidt Telescope
USAF: United States Air Force
USMP: United States Microgravity Payload
UT: Universal Time (a.k.a. GMT, UTC, or Zulu Time)
UTC: Coordinated Universal Time (a.k.a. UT)
UV: UltraViolet
UVS: UltraViolet Spectrometer
VAB: Vehicle Assembly Building (formerly Vertical Assembly Building)
VAFB: Vandenberg Air Force Base
VEEGA: Venus-Earth-Earth Gravity Assist (Galileo flight path)
VHF: Very High Frequency
VLA: Very Large Array
VLBA: Very Long Baseline Array
VLBI: Very Long Baseline Interferometry
VLF: Very Low Frequency
VLT: Very Large Telescope
VMS: Vertical Motion Simulator
VOIR: Venus Orbiting Imaging Radar (superseded by VRM)
VPF: Vertical Processing Facility
VRM: Venus Radar Mapper (now called Magellan)
WD: White Dwarf

WFPC: Wide Field / Planetary Camera (on HST)
WFPCII: Replacement for WFPC
WIYN: Wisconsin / Indiana / Yale / NOAO telescope
WSMR: White Sands Missile Range
WTR: Western Test Range
WUPPE: Wisconsin Ultraviolet PhotoPolarimter Experiment (Astro package)
XMM: X-ray Multi Mirror
XUV: eXtreme UltraViolet
YSO: Young Stellar Object

```
#!/usr/bin/perl
# 'alt', An Acronym Scrambling Program, by Larry Wall

$THRESHOLD = 2;

srand;
while (<>) {
    next unless /^([A-Z]\S+): */;
    $key = $1;
    $acro{$key} = $';
    @words = split(/\W+/,$');
    unshift(@words,$key);
    $off = 0;
    foreach $word (@words) {
        next unless $word =~ /^[A-Z]/;
        *w = $&;
        vec($w{$word}, $off++ % 6, 1) = 1;
    }
}

foreach $letter (A .. Z) {
    *w = $letter;
    @w = keys %w;
    if (@w < $THRESHOLD) {
        @d = `egrep '^$letter' /usr/dict/words`;
        chop @d;
        push(@w, @d);
    }
}

foreach $key (sort keys %acro) {
    $off = 0;
    $acro = $acro{$key};
    $acro =~ s/((([A-Z])[A-Z]*)[a-z]*)/ &pick($3, $2, $1, ++$off) || $& /eg;
    print "$key: $acro";
}
```

```
sub pick {
    local($letter, $prefix, $oldword, $off) = @_;
    $i = 0;
    if (length($prefix) > 1 && index($key,$prefix) < 0) {
        if ($prefix eq $oldword) {
            $prefix = '';
        }
        else {
            $prefix = $letter;
        }
    }
    if (length($prefix) > 1) {
        local(*w) = substr($prefix,0,1);
        do {
            $word = $w[rand @w];
        } until $word ne $oldword && $word =~ /^$prefix/i || ++$i > 30;
        $word =~ s/^$prefix/$prefix/i;
        $word;
    }
    elsif (length($prefix) == 1) {
        local(*w) = $prefix;
        do {
            $word = $w[rand @w];
        } until $word ne $oldword && vec($w{$word}, $off, 1) || ++$i > 10;
        $word = "\u\L$word" if $word =~ tr/a-z/A-Z/;
        $word;
    }
    else {
        local(*w) = substr($oldword,0,1);
        do {
            $word = $w[rand @w];
        } until $word ne $oldword && $word =~ tr/a-z/A-Z/ == 0 || ++$i > 30;
        $word;
    }
}
```

```
Actual: comp.graphics
Predicted: comp.graphics

   >If this idea goes through, it's the thin end of the wedge.  Soon
   >companies will be doing larger, and more permanant, billboards in the
   >sky.  I wouldn't want a world a few decades from now when the sky
   >looks like Las Vegas.  That would _really_ make me sad.

   Think for a moment about the technology required to do that.  By
   the time they could make the Earth's sky look like Las Vegas,
   the people could afford to go backpacking on the Moon. Round
   trip costs for 500 kg to the Moon would be about the same as
```

```
   5000 kg in a Low Earth "advertising" orbit: Very roughly the
   same cost as a smallish billboard, therefore. If such ads were
   to become common place, that would have to be a very low price…

This is nonsense.  Its like saying that by the time commercials
on television become commonplace every citizen will have their own
hour long nationally broadcast TV program.

   There's always been a problem of having to get
   away from civilization before you can really find "natural"
   scenery. 100 years ago, this usually didn't take a trip
   of over 5 miles. Today, most people would have to go 100 miles
   or more. If we ever get to the point where we have billboards
   on orbit, that essentially means that no place on Earth is still
   "wild." While that may or may not be a good thing, the orbital
   billboards aren't the problem: They are just a symptom of
   growing, densely-populated civilization. Banning such ads will
   not save your view of the night sky, because by the time
   such ads could become widespread you will probably have trouble
   finding a place without street lights, where you can _see_
   the stars…

The rest of your post is strange mishmash of "its already really bad"
and "it doesn't really matter if it gets worse."  You should try to
figure out what you are really arguing for.  (Kneejerk anti-environ-
mentalism?)

-david
Actual: comp.graphics
Predicted: comp.graphics
If gamma ray bursters are extragalactic, would absorption from the
galaxy be expected?  How transparent is the galactic core to gamma
rays?

How much energy does a burster put out?  I know energy depends on
distance, which is unknown.  An answer of the form _X_ ergs per
megaparsec^2 is OK.


Actual: comp.graphics
Predicted: sci.space
```

5 Random Forest

Question: Use the Wine dataset from sklearn.datasets to classify wine types using a Random Forest classifier. Your tasks are:

1) Load the Wine dataset and explore its features and classes.

2) Split the dataset into training and test sets (80-20 split).

3) Train a Random Forest Classifier.

4) Evaluate the model using accuracy, confusion matrix, and classification report.

5) (Optional) Show the feature importances and visualize them with a bar plot.

6) (Optional) plot the Decision tree of a random forest.

7) (Optional) Plot MSE of each decision tree of random forest.

8) (Optional) plot How mse is reducing when we increase the decision tree in random-forest

[1]:
```
import pandas as pd
import numpy as np
from sklearn.datasets import load_wine
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import plot_tree
from sklearn.metrics import accuracy_score, confusion_matrix,␣
↪ConfusionMatrixDisplay, mean_squared_error
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

[2]:
```
lw = load_wine()
```

[3]:
```
lv = lw.data
```

[4]:
```
data = pd.DataFrame(lv, columns=lw.feature_names)
```

[5]:
```
data['target'] = lw.target
```

[6]:
```
data
```

[6]:
```
     alcohol  malic_acid   ash  alcalinity_of_ash  magnesium  total_phenols  \
0      14.23        1.71  2.43               15.6      127.0           2.80
1      13.20        1.78  2.14               11.2      100.0           2.65
2      13.16        2.36  2.67               18.6      101.0           2.80
3      14.37        1.95  2.50               16.8      113.0           3.85
4      13.24        2.59  2.87               21.0      118.0           2.80
..       …         …    …                …          …             …
173    13.71        5.65  2.45               20.5       95.0           1.68
```

```
174    13.40        3.91  2.48               23.0      102.0           1.80
175    13.27        4.28  2.26               20.0      120.0           1.59
176    13.17        2.59  2.37               20.0      120.0           1.65
177    14.13        4.10  2.74               24.5       96.0           2.05

     flavanoids  nonflavanoid_phenols  proanthocyanins  color_intensity   hue  \
0          3.06                  0.28             2.29             5.64  1.04
1          2.76                  0.26             1.28             4.38  1.05
2          3.24                  0.30             2.81             5.68  1.03
3          3.49                  0.24             2.18             7.80  0.86
4          2.69                  0.39             1.82             4.32  1.04
..          …                   …                …               …    …
173        0.61                  0.52             1.06             7.70  0.64
174        0.75                  0.43             1.41             7.30  0.70
175        0.69                  0.43             1.35            10.20  0.59
176        0.68                  0.53             1.46             9.30  0.60
177        0.76                  0.56             1.35             9.20  0.61

     od280/od315_of_diluted_wines  proline  target
0                            3.92   1065.0       0
1                            3.40   1050.0       0
2                            3.17   1185.0       0
3                            3.45   1480.0       0
4                            2.93    735.0       0
..                            …        …      …
173                          1.74    740.0       2
174                          1.56    750.0       2
175                          1.56    835.0       2
176                          1.62    840.0       2
177                          1.60    560.0       2

[178 rows x 14 columns]
```

[7]:
```
X = data.drop(columns={'target'})
y = data['target']
```

[8]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42,␣
 ↪test_size=0.2)
```

[9]:
```
rf = RandomForestClassifier()
```

[10]:
```
rf.fit(X_train, y_train)
```

[10]:
```
RandomForestClassifier()
```

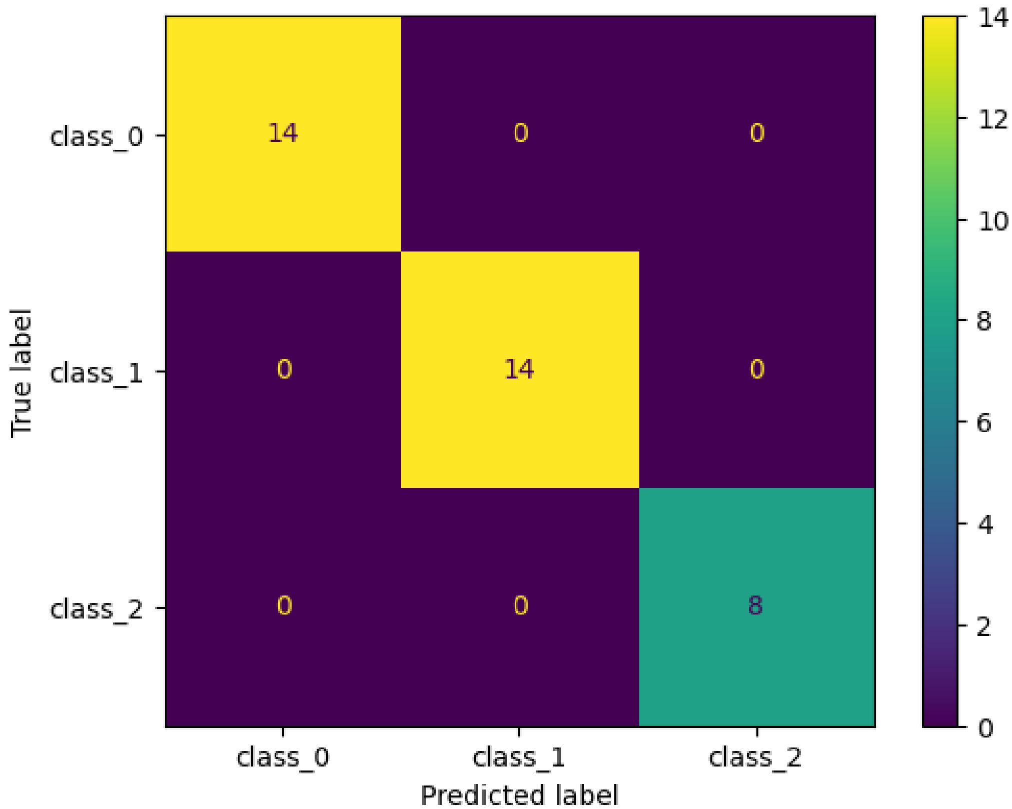
[11]:
```
y_pred_rf = rf.predict(X_test)
```

[12]:
```
accuracy_score(y_test, y_pred_rf)
```

[12]: 1.0

[13]:
```
cm = confusion_matrix(y_test, y_pred_rf)
cmd = ConfusionMatrixDisplay(cm, display_labels=lw.target_names)
cmd.plot()
```

[13]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1e5dee03f80>



[14]:
```
importance = rf.feature_importances_
f_n = lw.feature_names
i_f = pd.DataFrame({'importance' : importance,
                    'feature names' : f_n})
```

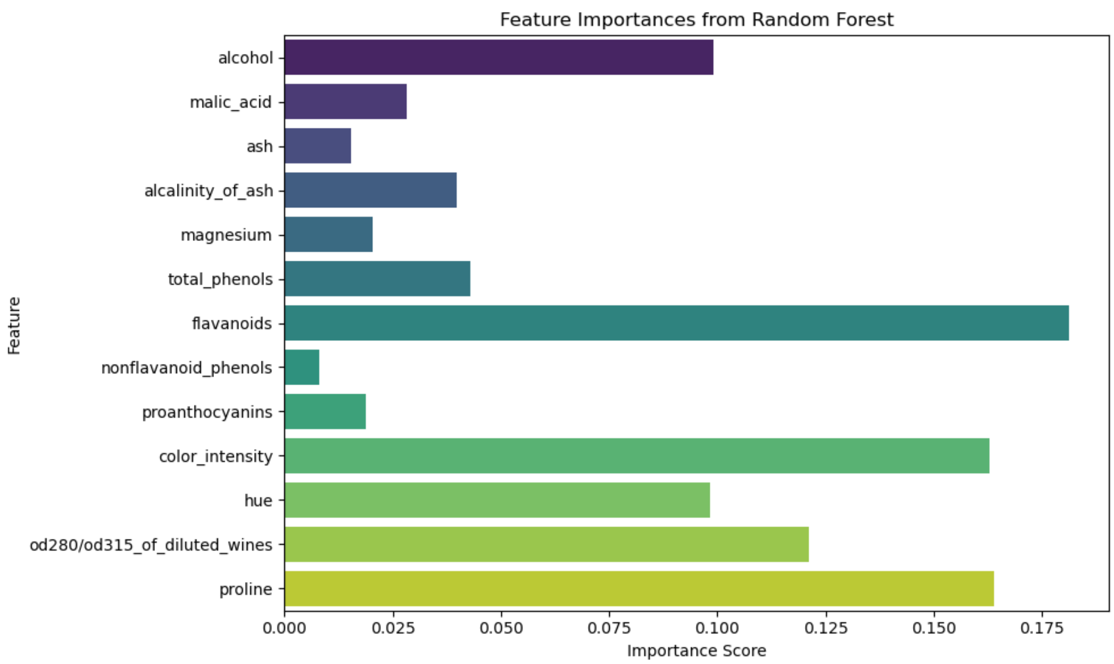
[15]:
```
i_f
```

[15]:

	importance	feature names
0	0.098977	alcohol
1	0.028134	malic_acid
2	0.015517	ash

```
3      0.039902              alcalinity_of_ash
4      0.020421                      magnesium
5      0.042897                  total_phenols
6      0.181336                     flavanoids
7      0.008072           nonflavanoid_phenols
8      0.018844                proanthocyanins
9      0.162747                color_intensity
10     0.098188                            hue
11     0.121076  od280/od315_of_diluted_wines
12     0.163889                        proline
```

[16]:
```
plt.figure(figsize=(10, 6))
sns.barplot(i_f, x='importance', y='feature names', palette='viridis')
plt.title('Feature Importances from Random Forest')
plt.xlabel('Importance Score')
plt.ylabel('Feature')
plt.tight_layout()
plt.show()
```



[ ]:
```

```

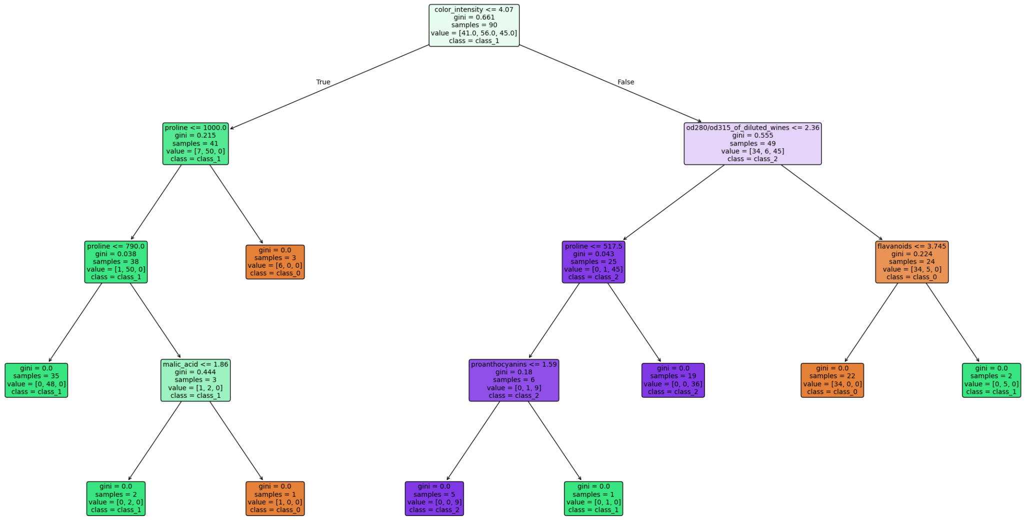
[17]:
```
tree_to_plot = rf.estimators_[99]
```

[19]:
```
tree_to_plot
```

[19]: DecisionTreeClassifier(max_features='sqrt', random_state=945251247)

[18]:
```
plt.figure(figsize=(30,16))
plot_tree(tree_to_plot, feature_names=lw.feature_names, class_names=lw.
 ↪target_names, filled=True, rounded=True, fontsize=10)
```

[18]:
```
[Text(0.4642857142857143, 0.9, 'color_intensity <= 4.07\ngini = 0.661\nsamples =
90\nvalue = [41.0, 56.0, 45.0]\nclass = class_1'),
 Text(0.21428571428571427, 0.7, 'proline <= 1000.0\ngini = 0.215\nsamples =
41\nvalue = [7, 50, 0]\nclass = class_1'),
 Text(0.3392857142857143, 0.8, 'True  '),
 Text(0.14285714285714285, 0.5, 'proline <= 790.0\ngini = 0.038\nsamples =
38\nvalue = [1, 50, 0]\nclass = class_1'),
 Text(0.07142857142857142, 0.3, 'gini = 0.0\nsamples = 35\nvalue = [0, 48,
0]\nclass = class_1'),
 Text(0.21428571428571427, 0.3, 'malic_acid <= 1.86\ngini = 0.444\nsamples =
3\nvalue = [1, 2, 0]\nclass = class_1'),
 Text(0.14285714285714285, 0.1, 'gini = 0.0\nsamples = 2\nvalue = [0, 2,
0]\nclass = class_1'),
 Text(0.2857142857142857, 0.1, 'gini = 0.0\nsamples = 1\nvalue = [1, 0,
0]\nclass = class_0'),
 Text(0.2857142857142857, 0.5, 'gini = 0.0\nsamples = 3\nvalue = [6, 0,
0]\nclass = class_0'),
 Text(0.7142857142857143, 0.7, 'od280/od315_of_diluted_wines <= 2.36\ngini =
0.555\nsamples = 49\nvalue = [34, 6, 45]\nclass = class_2'),
 Text(0.5892857142857143, 0.8, '  False'),
 Text(0.5714285714285714, 0.5, 'proline <= 517.5\ngini = 0.043\nsamples =
25\nvalue = [0, 1, 45]\nclass = class_2'),
 Text(0.5, 0.3, 'proanthocyanins <= 1.59\ngini = 0.18\nsamples = 6\nvalue = [0,
1, 9]\nclass = class_2'),
 Text(0.42857142857142855, 0.1, 'gini = 0.0\nsamples = 5\nvalue = [0, 0,
9]\nclass = class_2'),
 Text(0.5714285714285714, 0.1, 'gini = 0.0\nsamples = 1\nvalue = [0, 1,
0]\nclass = class_1'),
 Text(0.6428571428571429, 0.3, 'gini = 0.0\nsamples = 19\nvalue = [0, 0,
36]\nclass = class_2'),
 Text(0.8571428571428571, 0.5, 'flavanoids <= 3.745\ngini = 0.224\nsamples =
24\nvalue = [34, 5, 0]\nclass = class_0'),
 Text(0.7857142857142857, 0.3, 'gini = 0.0\nsamples = 22\nvalue = [34, 0,
0]\nclass = class_0'),
 Text(0.9285714285714286, 0.3, 'gini = 0.0\nsamples = 2\nvalue = [0, 5,
0]\nclass = class_1')]
```



[52]:
```python
def mse_score(y_test, y_pred):
    mse = mean_squared_error(y_test, y_pred)
    return mse
```

[53]:
```python
tree = []
mse_s = []
for i in range(100):
    dt = rf.estimators_[i]
    y_pred = dt.predict(X_test)
    mse = mse_score(y_test, y_pred)
    tree.append(i)
    mse_s.append(mse)
```

[54]:
```python
t_m = pd.DataFrame({'Tree': tree,
                    'MSE' : mse_s})
```

[62]:
```python
plt.figure(figsize=(30,16))
sns.lineplot(t_m, y = t_m['MSE'], x=t_m['Tree'])
plt.grid(True)
plt.xticks(range(len(mse_s)))
```

[62]:
```
([<matplotlib.axis.XTick at 0x1ed01d0dbb0>,
  <matplotlib.axis.XTick at 0x1ed02437cb0>,
  <matplotlib.axis.XTick at 0x1ed024106b0>,
  <matplotlib.axis.XTick at 0x1ed01c50380>,
  <matplotlib.axis.XTick at 0x1ed01d3b2c0>,
  <matplotlib.axis.XTick at 0x1ed01d3b140>,
```

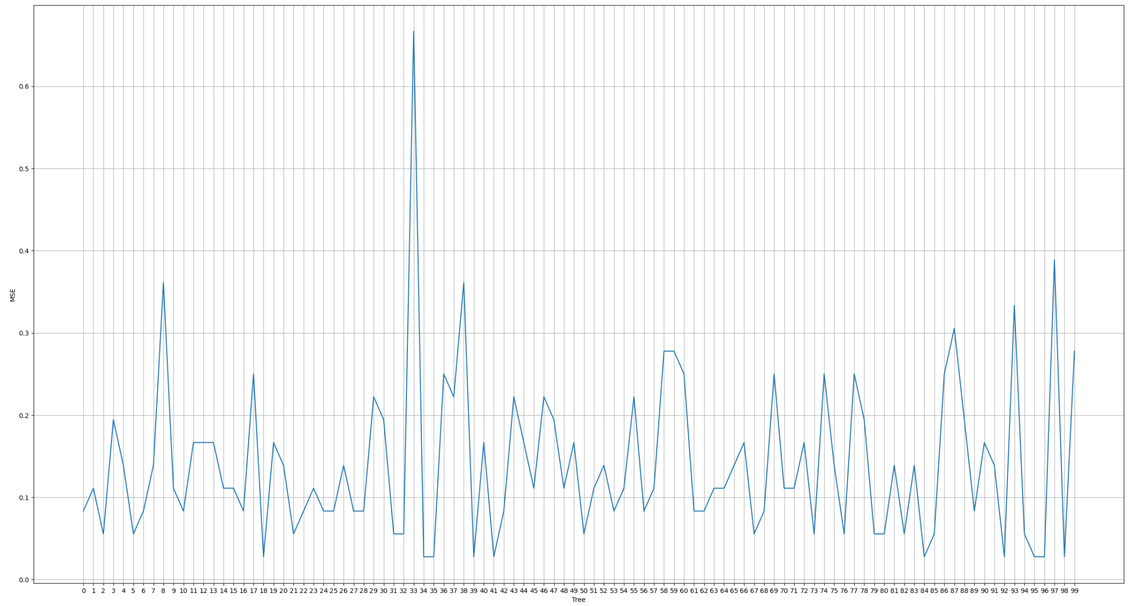
```
<matplotlib.axis.XTick at 0x1ed01cbb050>,
<matplotlib.axis.XTick at 0x1ed01cba540>,
<matplotlib.axis.XTick at 0x1ed02312480>,
<matplotlib.axis.XTick at 0x1ed01d57560>,
<matplotlib.axis.XTick at 0x1ed01cdd160>,
<matplotlib.axis.XTick at 0x1ed01cb9cd0>,
<matplotlib.axis.XTick at 0x1ed01d57e30>,
<matplotlib.axis.XTick at 0x1ed01d786b0>,
<matplotlib.axis.XTick at 0x1ed01d78cb0>,
<matplotlib.axis.XTick at 0x1ed01d795e0>,
<matplotlib.axis.XTick at 0x1ed01d54b60>,
<matplotlib.axis.XTick at 0x1ed01d78320>,
<matplotlib.axis.XTick at 0x1ed01d55280>,
<matplotlib.axis.XTick at 0x1ed01d7a660>,
<matplotlib.axis.XTick at 0x1ed01d7ae40>,
<matplotlib.axis.XTick at 0x1ed01d7b530>,
<matplotlib.axis.XTick at 0x1ed01d7b350>,
<matplotlib.axis.XTick at 0x1ed01d78920>,
<matplotlib.axis.XTick at 0x1ed01d7be00>,
<matplotlib.axis.XTick at 0x1ed01d986e0>,
<matplotlib.axis.XTick at 0x1ed01d98cb0>,
<matplotlib.axis.XTick at 0x1ed01d7bdd0>,
<matplotlib.axis.XTick at 0x1ed01d98f20>,
<matplotlib.axis.XTick at 0x1ed01d7bbf0>,
<matplotlib.axis.XTick at 0x1ed01d99dc0>,
<matplotlib.axis.XTick at 0x1ed01d9a5d0>,
<matplotlib.axis.XTick at 0x1ed01d9ad20>,
<matplotlib.axis.XTick at 0x1ed01c52c00>,
<matplotlib.axis.XTick at 0x1ed01d99940>,
<matplotlib.axis.XTick at 0x1ed01d9b620>,
<matplotlib.axis.XTick at 0x1ed01d9bd10>,
<matplotlib.axis.XTick at 0x1ed01dc05c0>,
<matplotlib.axis.XTick at 0x1ed01dc0d70>,
<matplotlib.axis.XTick at 0x1ed01d9b350>,
<matplotlib.axis.XTick at 0x1ed01dc0200>,
<matplotlib.axis.XTick at 0x1ed01d9b380>,
<matplotlib.axis.XTick at 0x1ed01dc1ca0>,
<matplotlib.axis.XTick at 0x1ed01dc2570>,
<matplotlib.axis.XTick at 0x1ed01d9a240>,
<matplotlib.axis.XTick at 0x1ed01dc1bb0>,
<matplotlib.axis.XTick at 0x1ed01dc2f90>,
<matplotlib.axis.XTick at 0x1ed01dc35c0>,
<matplotlib.axis.XTick at 0x1ed01dc3dd0>,
<matplotlib.axis.XTick at 0x1ed01de8620>,
<matplotlib.axis.XTick at 0x1ed01dc3a70>,
<matplotlib.axis.XTick at 0x1ed01de86e0>,
<matplotlib.axis.XTick at 0x1ed01de90d0>,
```

```
<matplotlib.axis.XTick at 0x1ed01de96d0>,
<matplotlib.axis.XTick at 0x1ed01de9ee0>,
<matplotlib.axis.XTick at 0x1ed01dea6c0>,
<matplotlib.axis.XTick at 0x1ed01dc2b40>,
<matplotlib.axis.XTick at 0x1ed01d55bb0>,
<matplotlib.axis.XTick at 0x1ed01de92e0>,
<matplotlib.axis.XTick at 0x1ed01deaba0>,
<matplotlib.axis.XTick at 0x1ed01deb380>,
<matplotlib.axis.XTick at 0x1ed01debb90>,
<matplotlib.axis.XTick at 0x1ed01deb470>,
<matplotlib.axis.XTick at 0x1ed01de9eb0>,
<matplotlib.axis.XTick at 0x1ed01e104d0>,
<matplotlib.axis.XTick at 0x1ed01e10cb0>,
<matplotlib.axis.XTick at 0x1ed01e113a0>,
<matplotlib.axis.XTick at 0x1ed01dc2360>,
<matplotlib.axis.XTick at 0x1ed01e11700>,
<matplotlib.axis.XTick at 0x1ed01e11df0>,
<matplotlib.axis.XTick at 0x1ed01e123f0>,
<matplotlib.axis.XTick at 0x1ed01e12cc0>,
<matplotlib.axis.XTick at 0x1ed01e13350>,
<matplotlib.axis.XTick at 0x1ed01e11eb0>,
<matplotlib.axis.XTick at 0x1ed01e10380>,
<matplotlib.axis.XTick at 0x1ed01e13d70>,
<matplotlib.axis.XTick at 0x1ed01e10950>,
<matplotlib.axis.XTick at 0x1ed01e2ccb0>,
<matplotlib.axis.XTick at 0x1ed01e2d460>,
<matplotlib.axis.XTick at 0x1ed01e121e0>,
<matplotlib.axis.XTick at 0x1ed01e2c260>,
<matplotlib.axis.XTick at 0x1ed01e2dd60>,
<matplotlib.axis.XTick at 0x1ed01e2e480>,
<matplotlib.axis.XTick at 0x1ed01e2ec30>,
<matplotlib.axis.XTick at 0x1ed01e13230>,
<matplotlib.axis.XTick at 0x1ed01e2e2d0>,
<matplotlib.axis.XTick at 0x1ed01e2f5c0>,
<matplotlib.axis.XTick at 0x1ed01e2fda0>,
<matplotlib.axis.XTick at 0x1ed01e506e0>,
<matplotlib.axis.XTick at 0x1ed01e50e90>,
<matplotlib.axis.XTick at 0x1ed01e2faa0>,
<matplotlib.axis.XTick at 0x1ed01e50920>,
<matplotlib.axis.XTick at 0x1ed01e51850>,
<matplotlib.axis.XTick at 0x1ed01e51eb0>,
<matplotlib.axis.XTick at 0x1ed01e52600>,
<matplotlib.axis.XTick at 0x1ed01e52d50>,
<matplotlib.axis.XTick at 0x1ed01e52d80>,
<matplotlib.axis.XTick at 0x1ed01e50110>,
<matplotlib.axis.XTick at 0x1ed01e53680>,
<matplotlib.axis.XTick at 0x1ed01e53da0>],
```

```
[Text(0, 0, '0'),
 Text(1, 0, '1'),
 Text(2, 0, '2'),
 Text(3, 0, '3'),
 Text(4, 0, '4'),
 Text(5, 0, '5'),
 Text(6, 0, '6'),
 Text(7, 0, '7'),
 Text(8, 0, '8'),
 Text(9, 0, '9'),
 Text(10, 0, '10'),
 Text(11, 0, '11'),
 Text(12, 0, '12'),
 Text(13, 0, '13'),
 Text(14, 0, '14'),
 Text(15, 0, '15'),
 Text(16, 0, '16'),
 Text(17, 0, '17'),
 Text(18, 0, '18'),
 Text(19, 0, '19'),
 Text(20, 0, '20'),
 Text(21, 0, '21'),
 Text(22, 0, '22'),
 Text(23, 0, '23'),
 Text(24, 0, '24'),
 Text(25, 0, '25'),
 Text(26, 0, '26'),
 Text(27, 0, '27'),
 Text(28, 0, '28'),
 Text(29, 0, '29'),
 Text(30, 0, '30'),
 Text(31, 0, '31'),
 Text(32, 0, '32'),
 Text(33, 0, '33'),
 Text(34, 0, '34'),
 Text(35, 0, '35'),
 Text(36, 0, '36'),
 Text(37, 0, '37'),
 Text(38, 0, '38'),
 Text(39, 0, '39'),
 Text(40, 0, '40'),
 Text(41, 0, '41'),
 Text(42, 0, '42'),
 Text(43, 0, '43'),
 Text(44, 0, '44'),
 Text(45, 0, '45'),
 Text(46, 0, '46'),
```

```
Text(47, 0, '47'),
Text(48, 0, '48'),
Text(49, 0, '49'),
Text(50, 0, '50'),
Text(51, 0, '51'),
Text(52, 0, '52'),
Text(53, 0, '53'),
Text(54, 0, '54'),
Text(55, 0, '55'),
Text(56, 0, '56'),
Text(57, 0, '57'),
Text(58, 0, '58'),
Text(59, 0, '59'),
Text(60, 0, '60'),
Text(61, 0, '61'),
Text(62, 0, '62'),
Text(63, 0, '63'),
Text(64, 0, '64'),
Text(65, 0, '65'),
Text(66, 0, '66'),
Text(67, 0, '67'),
Text(68, 0, '68'),
Text(69, 0, '69'),
Text(70, 0, '70'),
Text(71, 0, '71'),
Text(72, 0, '72'),
Text(73, 0, '73'),
Text(74, 0, '74'),
Text(75, 0, '75'),
Text(76, 0, '76'),
Text(77, 0, '77'),
Text(78, 0, '78'),
Text(79, 0, '79'),
Text(80, 0, '80'),
Text(81, 0, '81'),
Text(82, 0, '82'),
Text(83, 0, '83'),
Text(84, 0, '84'),
Text(85, 0, '85'),
Text(86, 0, '86'),
Text(87, 0, '87'),
Text(88, 0, '88'),
Text(89, 0, '89'),
Text(90, 0, '90'),
Text(91, 0, '91'),
Text(92, 0, '92'),
Text(93, 0, '93'),
```

```
 Text(94, 0, '94'),
 Text(95, 0, '95'),
 Text(96, 0, '96'),
 Text(97, 0, '97'),
 Text(98, 0, '98'),
 Text(99, 0, '99')])
```

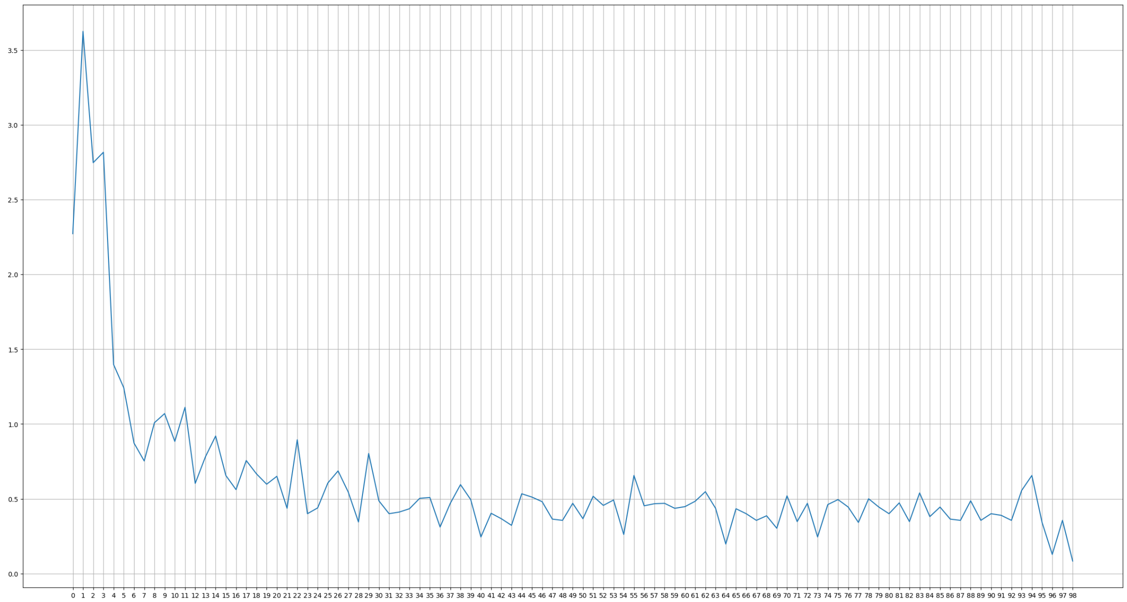


[119]:

```
mse_l = []
for i in range(1,100):
    rf = RandomForestClassifier(n_estimators=i)
    rf.fit(X_train, y_train)
    y_pred = rf.predict(X_test)
    mse = mse_score(y_test,y_pred)
    mse_l.append(mse)
```

[125]:

```
plt.figure(figsize=(30,16))
plt.plot(mse_l)
plt.grid(True)
plt.xticks(range(len(mse_l)))
plt.show()
```



6 KNN

Question: Use the Breast Cancer Wisconsin dataset from sklearn.datasets to classify tumors as malignant or benign using the K-Nearest Neighbors (KNN) algorithm. Your tasks are:

1) Load and explore the dataset (check number of features, classes).

2) Standardize the feature values using StandardScaler.

3) Split the data into training and test sets (75-25 split).

4) Train a KNN classifier (start with n_neighbors=5).

5) Evaluate the model using accuracy, confusion matrix, and classification report.

6) (Optional) Test model performance for different values of k and plot accuracy vs. k.

[17]:
```
import pandas as pd
import numpy as np
from sklearn.datasets import load_breast_cancer
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix,␣
 ↪classification_report,ConfusionMatrixDisplay
import matplotlib.pyplot as plt
```

```python
import seaborn as sns
```

[2]:
```python
lbc = load_breast_cancer()
```

[3]:
```python
bcv = lbc.data
```

[4]:
```python
ss = StandardScaler()
```

[5]:
```python
bcv = ss.fit_transform(bcv)
```

[6]:
```python
data = pd.DataFrame(bcv, columns=lbc.feature_names)
```

[7]:
```python
data
```

[7]:
```
     mean radius  mean texture  mean perimeter  mean area  mean smoothness  \
0       1.097064     -2.073335        1.269934   0.984375         1.568466
1       1.829821     -0.353632        1.685955   1.908708        -0.826962
2       1.579888      0.456187        1.566503   1.558884         0.942210
3      -0.768909      0.253732       -0.592687  -0.764464         3.283553
4       1.750297     -1.151816        1.776573   1.826229         0.280372
..           …             …               …          …               …
564     2.110995      0.721473        2.060786   2.343856         1.041842
565     1.704854      2.085134        1.615931   1.723842         0.102458
566     0.702284      2.045574        0.672676   0.577953        -0.840484
567     1.838341      2.336457        1.982524   1.735218         1.525767
568    -1.808401      1.221792       -1.814389  -1.347789        -3.112085

     mean compactness  mean concavity  mean concave points  mean symmetry  \
0            3.283515        2.652874             2.532475       2.217515
1           -0.487072       -0.023846             0.548144       0.001392
2            1.052926        1.363478             2.037231       0.939685
3            3.402909        1.915897             1.451707       2.867383
4            0.539340        1.371011             1.428493      -0.009560
..                …              …                   …              …
564          0.219060        1.947285             2.320965      -0.312589
565         -0.017833        0.693043             1.263669      -0.217664
566         -0.038680        0.046588             0.105777      -0.809117
567          3.272144        3.296944             2.658866       2.137194
568         -1.150752       -1.114873            -1.261820      -0.820070

     mean fractal dimension  …  worst radius  worst texture  \
0                  2.255747  …      1.886690      -1.359293
1                 -0.868652  …      1.805927      -0.369203
2                 -0.398008  …      1.511870      -0.023974
3                  4.910919  …     -0.281464       0.133984
4                 -0.562450  …      1.298575      -1.466770
..                      …   …           …             …
```

```
564             -0.931027  …      1.901185      0.117700
565             -1.058611  …      1.536720      2.047399
566             -0.895587  …      0.561361      1.374854
567              1.043695  …      1.961239      2.237926
568             -0.561032  …     -1.410893      0.764190

     worst perimeter  worst area  worst smoothness  worst compactness  \
0           2.303601    2.001237          1.307686           2.616665
1           1.535126    1.890489         -0.375612          -0.430444
2           1.347475    1.456285          0.527407           1.082932
3          -0.249939   -0.550021          3.394275           3.893397
4           1.338539    1.220724          0.220556          -0.313395
..               …           …                 …                  …
564         1.752563    2.015301          0.378365          -0.273318
565         1.421940    1.494959         -0.691230          -0.394820
566         0.579001    0.427906         -0.809587           0.350735
567         2.303601    1.653171          1.430427           3.904848
568        -1.432735   -1.075813         -1.859019          -1.207552

     worst concavity  worst concave points  worst symmetry  \
0           2.109526              2.296076        2.750622
1          -0.146749              1.087084       -0.243890
2           0.854974              1.955000        1.152255
3           1.989588              2.175786        6.046041
4           0.613179              0.729259       -0.868353
..               …                     …               …
564         0.664512              1.629151       -1.360158
565         0.236573              0.733827       -0.531855
566         0.326767              0.414069       -1.104549
567         3.197605              2.289985        1.919083
568        -1.305831             -1.745063       -0.048138

     worst fractal dimension
0                   1.937015
1                   0.281190
2                   0.201391
3                   4.935010
4                  -0.397100
..                       …
564                -0.709091
565                -0.973978
566                -0.318409
567                 2.219635
568                -0.751207

[569 rows x 30 columns]
```

[8]:
```python
data['target'] = lbc.target
```

[9]:
```python
data
```

[9]:
```
     mean radius  mean texture  mean perimeter  mean area  mean smoothness  \
0       1.097064     -2.073335        1.269934   0.984375         1.568466
1       1.829821     -0.353632        1.685955   1.908708        -0.826962
2       1.579888      0.456187        1.566503   1.558884         0.942210
3      -0.768909      0.253732       -0.592687  -0.764464         3.283553
4       1.750297     -1.151816        1.776573   1.826229         0.280372
..           …            …              …          …             …
564     2.110995      0.721473        2.060786   2.343856         1.041842
565     1.704854      2.085134        1.615931   1.723842         0.102458
566     0.702284      2.045574        0.672676   0.577953        -0.840484
567     1.838341      2.336457        1.982524   1.735218         1.525767
568    -1.808401      1.221792       -1.814389  -1.347789        -3.112085

     mean compactness  mean concavity  mean concave points  mean symmetry  \
0            3.283515        2.652874             2.532475       2.217515
1           -0.487072       -0.023846             0.548144       0.001392
2            1.052926        1.363478             2.037231       0.939685
3            3.402909        1.915897             1.451707       2.867383
4            0.539340        1.371011             1.428493      -0.009560
..              …              …                   …              …
564          0.219060        1.947285             2.320965      -0.312589
565         -0.017833        0.693043             1.263669      -0.217664
566         -0.038680        0.046588             0.105777      -0.809117
567          3.272144        3.296944             2.658866       2.137194
568         -1.150752       -1.114873            -1.261820      -0.820070

     mean fractal dimension  …  worst texture  worst perimeter  worst area  \
0                  2.255747  …      -1.359293         2.303601    2.001237
1                 -0.868652  …      -0.369203         1.535126    1.890489
2                 -0.398008  …      -0.023974         1.347475    1.456285
3                  4.910919  …       0.133984        -0.249939   -0.550021
4                 -0.562450  …      -1.466770         1.338539    1.220724
..                      …   …           …                …           …
564               -0.931027  …       0.117700         1.752563    2.015301
565               -1.058611  …       2.047399         1.421940    1.494959
566               -0.895587  …       1.374854         0.579001    0.427906
567                1.043695  …       2.237926         2.303601    1.653171
568               -0.561032  …       0.764190        -1.432735   -1.075813

     worst smoothness  worst compactness  worst concavity  \
0            1.307686           2.616665         2.109526
1           -0.375612          -0.430444        -0.146749
2            0.527407           1.082932         0.854974
```

```
3            3.394275           3.893397         1.989588
4            0.220556          -0.313395         0.613179
..                …                  …              …
564          0.378365          -0.273318         0.664512
565         -0.691230          -0.394820         0.236573
566         -0.809587           0.350735         0.326767
567          1.430427           3.904848         3.197605
568         -1.859019          -1.207552        -1.305831

     worst concave points  worst symmetry  worst fractal dimension  target
0                2.296076        2.750622                 1.937015       0
1                1.087084       -0.243890                 0.281190       0
2                1.955000        1.152255                 0.201391       0
3                2.175786        6.046041                 4.935010       0
4                0.729259       -0.868353                -0.397100       0
..                    …               …                       …      …
564              1.629151       -1.360158                -0.709091       0
565              0.733827       -0.531855                -0.973978       0
566              0.414069       -1.104549                -0.318409       0
567              2.289985        1.919083                 2.219635       0
568             -1.745063       -0.048138                -0.751207       1

[569 rows x 31 columns]
```

[10]:
```python
X = data.drop(columns={'target'})
y = data['target']
```

[11]:
```python
X_train, X_test, y_train, y_test = train_test_split(X,y, random_state=42,␣
↪test_size=0.25)
```

[12]:
```python
knn = KNeighborsClassifier()
```

[13]:
```python
knn.fit(X_train, y_train)
```

[13]:
```
KNeighborsClassifier()
```

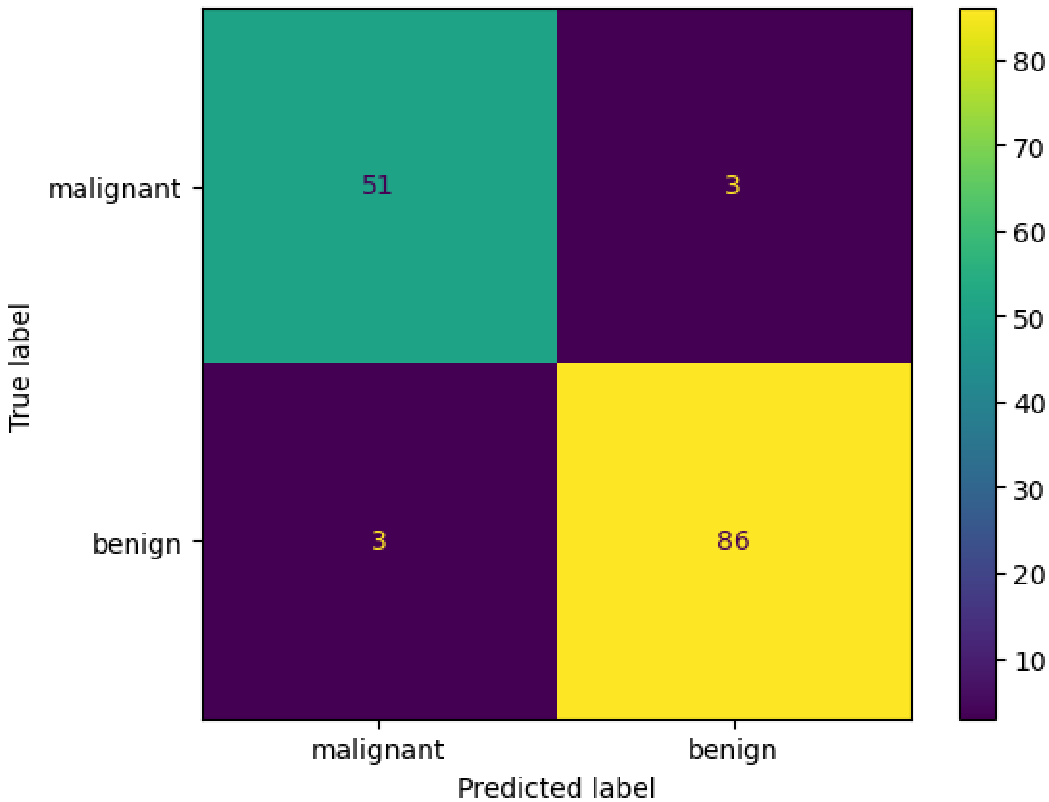
[14]:
```python
y_pred_k = knn.predict(X_test)
```

[15]:
```python
accuracy_score(y_test, y_pred_k)
```

[15]:
```
0.958041958041958
```

[18]:
```python
cm = confusion_matrix(y_test, y_pred_k)
cmd = ConfusionMatrixDisplay(cm, display_labels=lbc.target_names)
cmd.plot()
```

[18]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x2a71aa261e0>
```



[19]:
```
print(classification_report(y_test, y_pred_k))
```

```
              precision    recall  f1-score   support

           0       0.94      0.94      0.94        54
           1       0.97      0.97      0.97        89

    accuracy                           0.96       143
   macro avg       0.96      0.96      0.96       143
weighted avg       0.96      0.96      0.96       143
```

[20]:
```
k = []
acc = []
for i in range(5, 100):
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train,y_train)
    a = knn.score(X_test,y_test)
    acc.append(a)
    k.append(i)
```

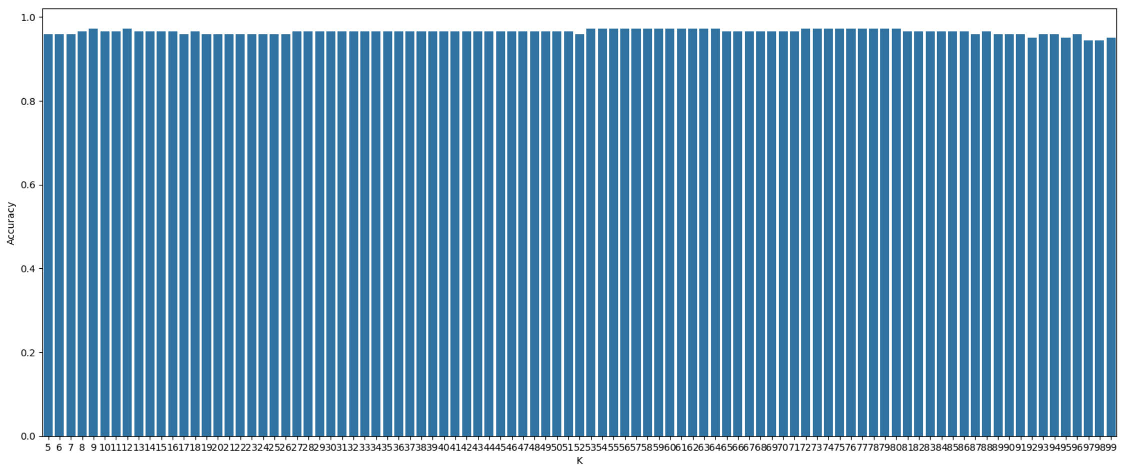
[21]:
```
k_a = pd.DataFrame({'K': k,
                    'Accuracy' : acc})
```

[28]:
```
k_a['Accuracy'][4]
```

[28]: 0.972027972027972

[24]:
```
plt.figure(figsize=(20,8))
sns.barplot(k_a, x='K', y="Accuracy")
```

[24]: <Axes: xlabel='K', ylabel='Accuracy'>



[ ]:
```

```