Homework 2 Part 2

Due: Wednesday, October 5 @ 11:59pm

I strongly recommend you to use HiPerGator to solve this assignment

Open On-Demand: ood.rc.ufl.edu

Import Libraries and magics

In [564…
```
# load libraries and magics
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
```

Question 1

Consider the breast cancer dataset with 30 numerical attributes described below and a total of 569 samples. Each sample is labeled as malignant (class 0) or benign (class 1). This is a **binary classification task**.

In [640…
```
from sklearn.datasets import load_breast_cancer

cancer = load_breast_cancer(return_X_y=False, as_frame=True)

print(cancer.DESCR)
```

```
.. _breast_cancer_dataset:

Breast cancer wisconsin (diagnostic) dataset
--------------------------------------------

**Data Set Characteristics:**

    :Number of Instances: 569

    :Number of Attributes: 30 numeric, predictive attributes and the class

    :Attribute Information:
        - radius (mean of distances from center to points on the perimeter)
        - texture (standard deviation of gray-scale values)
        - perimeter
        - area
        - smoothness (local variation in radius lengths)
        - compactness (perimeter^2 / area - 1.0)
        - concavity (severity of concave portions of the contour)
        - concave points (number of concave portions of the contour)
        - symmetry
        - fractal dimension ("coastline approximation" - 1)

        The mean, standard error, and "worst" or largest (mean of the three
        worst/largest values) of these features were computed for each image,
        resulting in 30 features.  For instance, field 0 is Mean Radius, field
        10 is Radius SE, field 20 is Worst Radius.

        - class:
                - WDBC-Malignant
                - WDBC-Benign

    :Summary Statistics:

    ===================================== ====== ======
                                           Min    Max
    ===================================== ====== ======
    radius (mean):                        6.981  28.11
    texture (mean):                       9.71   39.28
    perimeter (mean):                     43.79  188.5
    area (mean):                          143.5  2501.0
    smoothness (mean):                    0.053  0.163
    compactness (mean):                   0.019  0.345
    concavity (mean):                     0.0    0.427
    concave points (mean):                0.0    0.201
    symmetry (mean):                      0.106  0.304
    fractal dimension (mean):             0.05   0.097
    radius (standard error):              0.112  2.873
    texture (standard error):             0.36   4.885
    perimeter (standard error):           0.757  21.98
    area (standard error):                6.802  542.2
    smoothness (standard error):          0.002  0.031
    compactness (standard error):         0.002  0.135
    concavity (standard error):           0.0    0.396
    concave points (standard error):      0.0    0.053
    symmetry (standard error):            0.008  0.079
    fractal dimension (standard error):   0.001  0.03
    radius (worst):                       7.93   36.04
    texture (worst):                      12.02  49.54
    perimeter (worst):                    50.41  251.2
```

```
    area (worst):                         185.2  4254.0
    smoothness (worst):                   0.071  0.223
    compactness (worst):                  0.027  1.058
    concavity (worst):                    0.0    1.252
    concave points (worst):               0.0    0.291
    symmetry (worst):                     0.156  0.664
    fractal dimension (worst):            0.055  0.208
    ===================================== ====== ======

    :Missing Attribute Values: None

    :Class Distribution: 212 - Malignant, 357 - Benign

    :Creator:  Dr. William H. Wolberg, W. Nick Street, Olvi L. Mangasarian

    :Donor: Nick Street

    :Date: November, 1995

This is a copy of UCI ML Breast Cancer Wisconsin (Diagnostic) datasets.
https://goo.gl/U2Uwz2

Features are computed from a digitized image of a fine needle
aspirate (FNA) of a breast mass.  They describe
characteristics of the cell nuclei present in the image.

Separating plane described above was obtained using
Multisurface Method-Tree (MSM-T) [K. P. Bennett, "Decision Tree
Construction Via Linear Programming." Proceedings of the 4th
Midwest Artificial Intelligence and Cognitive Science Society,
pp. 97-101, 1992], a classification method which uses linear
programming to construct a decision tree.  Relevant features
were selected using an exhaustive search in the space of 1-4
features and 1-3 separating planes.

The actual linear program used to obtain the separating plane
in the 3-dimensional space is that described in:
[K. P. Bennett and O. L. Mangasarian: "Robust Linear
Programming Discrimination of Two Linearly Inseparable Sets",
Optimization Methods and Software 1, 1992, 23-34].

This database is also available through the UW CS ftp server:

ftp ftp.cs.wisc.edu
cd math-prog/cpo-dataset/machine-learn/WDBC/

.. topic:: References

   - W.N. Street, W.H. Wolberg and O.L. Mangasarian. Nuclear feature extractio
n
     for breast tumor diagnosis. IS&T/SPIE 1993 International Symposium on
     Electronic Imaging: Science and Technology, volume 1905, pages 861-870,
     San Jose, CA, 1993.
   - O.L. Mangasarian, W.N. Street and W.H. Wolberg. Breast cancer diagnosis a
nd
     prognosis via linear programming. Operations Research, 43(4), pages 570-5
77,
     July-August 1995.
   - W.H. Wolberg, W.N. Street, and O.L. Mangasarian. Machine learning techniq
ues
```

```
     to diagnose breast cancer from fine-needle aspirates. Cancer Letters 77
(1994)
     163-171.
```

In [641…
```python
cancer.keys()
```

Out[641]:
```
dict_keys(['data', 'target', 'frame', 'target_names', 'DESCR', 'feature_name
s', 'filename', 'data_module'])
```

In [642…
```python
X = cancer.data

X
```

Out[642]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	symı
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010	0.14710	C
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.07017	(
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.12790	0
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.10520	0
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.10430	C
...	...	...	...	...	...	...	...	...	
564	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890	C
565	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791	C
566	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302	C
567	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15200	0
568	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000	C

569 rows × 30 columns

In [643…
```python
t = cancer.target

t
```

Out[643]:
```
0      0
1      0
2      0
3      0
4      0
      ..
564    0
565    0
566    0
567    0
568    1
Name: target, Length: 569, dtype: int64
```

1. Partition the data into training and test sets (80/20 stratified split).
2. Build a `scikit-learn` pipeline to train a Logistic Regression classifier with Lasso regularizer.

3. Carry out hyperparameter tuning and train your final model.

4. Make predictions for training and test sets. Report performance measures using the `classification_report` function.

5. Which features are most informative to make the final prediction?

   - To access the parameters of the Logistic Regression classifier within a pipeline, use the attribute `named_steps` and index it by the name of that step in the pipeline. For example, `final_model.named_steps['logistic_regression'].coef_`.

6. Recall that the Logistic Regression mapper function looks like $\mathbf{y} = \phi(z)$ where $z = \mathbf{w}^T\mathbf{x} + w_0$. Moreover, $y = \begin{cases} 1, & z \geq 0 \\ 0, & z < 0 \end{cases}$. Predict the values for z for the training set. These values can be accessed with the attribute `decision_function`.

7. Now consider $y = \begin{cases} 1, & z \geq \delta \\ 0, & z < \delta \end{cases}$, where $\delta \in \mathbb{R}$ is threshold continuous value. Plot the **precision-recall curve**. Which threshold δ would you use to obtain a **recall of at least 80%**? Justify your answer based on these results.

8. Use the newly found threshold value to make predictions for the test set. Compare the results with those from part 4.

1. Partition the data into training and test sets (80/20 stratified split).

In [644…

```
# Find most predictive attribute
cancer.data['target'] = cancer.target;
corr_matrix = cancer.data.corr(method='pearson');
corr_matrix['target'].sort_values(ascending=False)
```

Out[644]:
```
target                     1.000000
smoothness error           0.067016
mean fractal dimension     0.012838
texture error              0.008303
symmetry error             0.006522
fractal dimension error   -0.077972
concavity error           -0.253730
compactness error         -0.292999
worst fractal dimension   -0.323872
mean symmetry             -0.330499
mean smoothness           -0.358560
concave points error      -0.408042
mean texture              -0.415185
worst symmetry            -0.416294
worst smoothness          -0.421465
worst texture             -0.456903
area error                -0.548236
perimeter error           -0.556141
radius error              -0.567134
worst compactness         -0.590998
mean compactness          -0.596534
worst concavity           -0.659610
mean concavity            -0.696360
mean area                 -0.708984
mean radius               -0.730029
worst area                -0.733825
mean perimeter            -0.742636
worst radius              -0.776454
mean concave points       -0.776614
worst perimeter           -0.782914
worst concave points      -0.793566
Name: target, dtype: float64
```

'smoothness error' has the largest predictive value

In [645…
```
# train-test split with stratification

smoothness_error_cat = pd.cut(cancer.data['smoothness error'],
                              bins=[0.,0.004,0.0053,0.0075,0.01,np.inf],
                              labels=[1, 2, 3, 4, 5]);

train, test, cat_train, cat_test = train_test_split(cancer.data,
                                                    smoothness_error_cat,
                                                    test_size=0.2,
                                                    shuffle=True,
                                                    random_state=42,
                                                   stratify=smoothness_error_ca

num_pipe = Pipeline([('std_scaler', StandardScaler())])
full_pipeline = ColumnTransformer([('num', num_pipe, list(cancer.data.columns))
```

In [646…
```
# final prepared training and test sets
train_set_prepared = full_pipeline.fit_transform(train)
test_set_prepared = full_pipeline.transform(test)

training_set = pd.DataFrame(train_set_prepared,
                            columns=train.columns,
                            index=train.index);
X_train = training_set.drop('target', axis=1).to_numpy();
```

```
t_train = train['target'].to_numpy();

test_set = pd.DataFrame(test_set_prepared,
                        columns=test.columns,
                        index=test.index);
X_test = test_set.drop('target', axis=1).to_numpy();
t_test = test['target'].to_numpy();
```

In [647…
```
X_train.shape, t_train.shape, X_test.shape, t_test.shape
```

Out[647]:
```
((455, 30), (455,), (114, 30), (114,))
```

2. Build a `scikit-learn` pipeline to train a Logistic Regression classifier with Lasso regularizer.

In [648…
```
from sklearn.linear_model import LogisticRegression

log_reg = LogisticRegression(penalty='l1', solver='liblinear', random_state=1)
log_reg.fit(X_train,t_train);
```

3. Carry out hyperparameter tuning and train your final model.

In [649…
```
from sklearn.model_selection import GridSearchCV
# Using GridSearchCV to find the best parameters
param_grid = {'penalty':['l1'], 'C':np.logspace(-20,30,25),
             'max_iter':[50,60,75,80,100,150,200]}
log_reg_gridCV = GridSearchCV(log_reg, param_grid, cv=5);
log_reg_gridCV.fit(X_train, t_train);
```

In [650…
```
log_reg_gridCV.best_params_
```

Out[650]:
```
{'C': 6.81292069057965, 'max_iter': 50, 'penalty': 'l1'}
```

In [651…
```
log_reg_gridCV.best_score_
```

Out[651]:
```
0.9648351648351647
```

In [652…
```
final_model_log_reg = log_reg_gridCV.best_estimator_
final_model_log_reg.fit(X_train, t_train);
```

4. Make predictions for training and test sets. Report performance measures using the `classification_report` function.

In [653…
```
from sklearn.metrics import classification_report

#make predictions using the final model
y_train = final_model_log_reg.predict(X_train)
y_test = final_model_log_reg.predict(X_test)
target_names = ['malignant', 'benign']
print(classification_report(t_train,y_train, target_names=target_names))
```

```
              precision    recall  f1-score   support

   malignant       0.99      0.98      0.98       163
      benign       0.99      1.00      0.99       292

    accuracy                           0.99       455
   macro avg       0.99      0.99      0.99       455
weighted avg       0.99      0.99      0.99       455
```

In [654…

```
print(classification_report(t_test,y_test, target_names=target_names))
```

```
              precision    recall  f1-score   support

   malignant       0.98      0.94      0.96        49
      benign       0.96      0.98      0.97        65

    accuracy                           0.96       114
   macro avg       0.97      0.96      0.96       114
weighted avg       0.97      0.96      0.96       114
```

5. Which features are most informative to make the final prediction?

- To access the parameters of the Logistic Regression classifier within a pipeline, use the attribute `named_steps` and index it by the name of that step in the pipeline. For example, `final_model.named_steps['logistic_regression'].coef_`.

In [655…

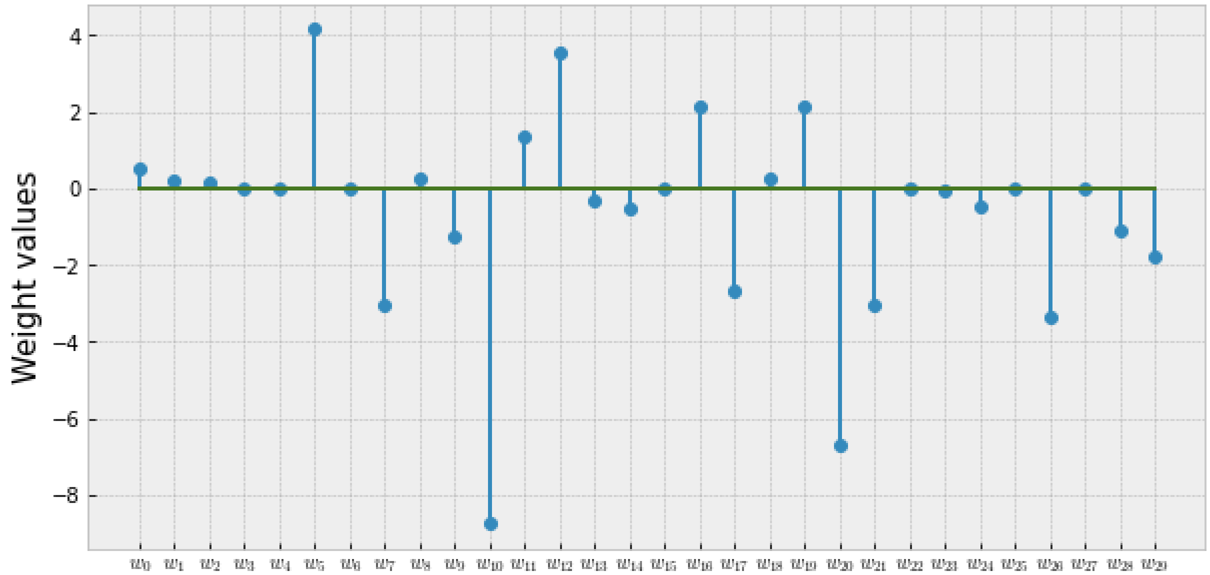
```
w = final_model_log_reg.coef_[0]
w
```

Out[655]:

```
array([ 0.54383662,  0.2215535 ,  0.13461909,  0.        ,  0.        ,
        4.1735879 ,  0.        , -3.04854594,  0.27290557, -1.2720276 ,
       -8.74854189,  1.37149503,  3.54863471, -0.32795844, -0.52363186,
        0.        ,  2.13339901, -2.67994893,  0.25826112,  2.13164993,
       -6.68631971, -3.0556383 ,  0.        , -0.06078568, -0.48252779,
        0.        , -3.33554692,  0.        , -1.12087824, -1.78859429])
```

In [656…

```
plt.figure(figsize=(10,5))
plt.stem(w)
plt.ylabel('Weight values', size=15)
plt.xticks(np.arange(30), ['$w_{'+str(i)+'}$' for i in range(len(w))],rotation=
```



Since features 5, 10, 12, 20, and 26 have the largest magnitudes, they are the most informative to make the final prediction

6. Recall that the Logistic Regression mapper function looks like $\mathbf{y} = \phi(z)$ where $z = \mathbf{w}^T\mathbf{x} + w_0$. Moreover, $y = \begin{cases} 1, & z \geq 0 \\ 0, & z < 0 \end{cases}$. Predict the values for z for the training set. These values can be accessed with the attribute `decision_function`.

In [657…

```
from sklearn.model_selection import cross_val_predict

y_scores = cross_val_predict(final_model_log_reg, X_train, t_train,
                             cv=10, method='decision_function')

y_scores
```

Out[657]:
```
array([ 3.58271977e+00,  2.65113312e-01, -1.84756047e+01,  7.69972691e+00,
        1.73511661e+01,  5.09855281e+00, -2.54301062e+01,  1.56408749e+01,
        1.02467796e+01,  1.83763158e+01,  3.86320410e+00,  1.11160297e+01,
       -1.97539724e+01,  6.83119961e+00,  2.54659999e+00,  6.16227932e+00,
        5.71056129e+00,  4.32361474e+00,  7.30052201e+00, -1.60872173e+01,
        9.41666650e+00,  9.22466210e+00,  6.52427794e+00,  1.03596344e+01,
        1.08130447e+01, -4.49251679e+00,  2.77284838e+00, -2.27897586e+01,
        2.31799276e+01,  1.59623194e+01, -2.00941764e+01,  1.06727181e+01,
        1.22114399e+01, -2.77899519e+01, -2.89891153e+01,  1.31311255e+01,
       -2.62613345e+01,  8.13113035e+00, -7.73806888e+00, -1.32841513e+01,
        3.31964330e+01, -1.45886702e+01, -1.35540005e+01,  7.34148519e+00,
       -2.04685406e+01, -8.62467377e+00,  4.15305017e+00, -1.53129037e+01,
        1.65228831e+01,  1.03703589e+01,  1.20521029e+01,  6.97569829e+00,
       -1.60506835e+00,  5.06652007e+00,  6.69265447e+00,  1.35736419e+01,
       -2.49270275e+01,  2.49220789e+00, -5.77169710e+00,  9.31201009e+00,
        6.05136581e+00,  1.34920088e+01, -2.16554273e+01,  1.57056807e+01,
        9.88281772e-01, -2.91780969e-01,  9.57384105e+00,  3.11779806e+00,
        1.48659342e+01,  1.34581878e+01,  7.67188846e+00,  2.69817318e+00,
        7.74817364e-01, -5.15698069e+01, -1.74670684e+01,  6.99867485e+00,
        3.32010654e+00,  1.64064711e+01,  1.61091919e+00,  6.17062357e+00,
        8.08844757e+00,  1.94113760e+01,  9.59243209e+00, -1.06695498e+01,
        1.71361750e+00,  1.40929322e+01,  1.14681377e+01,  1.60821315e+01,
       -2.89260911e+00,  2.03807166e+01, -1.34329214e+00,  1.52515792e+01,
        1.41764790e+01,  1.62686078e+01,  1.32712278e+01,  2.95568069e+00,
       -1.28580664e+01,  1.78589048e+01,  1.67302721e+01,  1.58709881e+01,
        1.95322415e+01,  1.34976922e+01,  9.07830408e+00, -1.99754745e+01,
        1.21866341e+01,  1.51432812e+00,  3.45278738e+00, -1.34075603e+01,
        1.83794954e+01, -1.52662354e+00,  1.14664288e+01, -4.50438867e+01,
        5.00418057e+00,  8.26010951e+00, -5.70490240e+00, -4.10968818e+01,
       -2.41053901e+01,  4.60503093e+00,  1.78996750e+01, -2.39860952e+01,
       -1.75115856e+01,  1.06810335e+01,  1.66203118e+00,  1.31652917e+01,
        1.75342796e+01,  1.13199672e+01, -4.16960734e+01,  9.11187685e+00,
        1.40804629e-01,  1.62904817e+01, -1.02275032e+01,  1.81604483e+01,
        5.60514369e+00,  6.12496787e+00, -1.78186779e+01,  7.09229031e+00,
        1.60850223e+01,  1.26642720e+01, -2.81861432e+01, -1.09601778e+01,
       -8.70921900e-01,  1.68308333e+01,  1.76722631e+01,  1.11727740e+01,
        1.79162585e+01,  1.47708608e+01,  1.69643454e+01, -3.32121890e+01,
        5.10979189e+00, -2.31505687e+01,  1.06800170e+01,  5.84830012e+00,
        1.27306318e+01,  6.44424911e+00,  1.82846476e+01, -8.36453800e+00,
        1.25238054e+01, -2.47201643e+01, -3.20190321e+01,  9.67581761e+00,
       -3.47783951e+01, -4.92338551e+00, -2.65359794e+01, -2.75565272e+01,
        4.95405778e+00, -2.99085372e+01,  1.26130993e+01, -7.51734756e+00,
        1.65299600e+01,  6.91965860e+00,  1.86048416e+01,  5.29068016e+00,
       -1.83589809e+01,  3.35693262e+00, -1.74906040e+00, -3.64399397e+00,
       -7.98732541e-01, -1.49666695e+01, -1.47112434e+01,  2.04832299e+01,
       -9.84358005e+00, -3.12393574e+01,  1.09074364e+01,  1.07452387e+01,
        1.21846170e+01, -3.08752668e+01,  1.37601876e+01,  1.39203316e+01,
        9.46449266e+00, -2.69766186e+01, -3.18763772e+01,  2.48928633e+00,
       -8.89822510e+00,  4.99884744e+00,  1.36400635e+01, -3.73089744e+01,
       -3.61406491e+01,  7.97716239e+00, -3.28461991e+01,  4.25840763e+00,
        4.74220908e+00, -6.84593157e+00,  3.23658178e+00,  5.08894718e+00,
       -1.74383801e+01,  1.56814535e+00,  3.14310502e+00, -1.82756795e+01,
        1.48892522e+01, -3.66244701e+00,  8.35468868e+00,  2.99836167e+00,
        9.99850444e-01, -2.45820002e+01,  1.29945619e+01,  1.04024678e+01,
        5.55792066e+00,  1.07597826e+01,  1.89437633e+01,  9.90930584e+00,
        7.93711207e+00,  2.06498209e+01,  1.52777288e+01,  6.22966712e+00,
        9.74856530e+00, -1.30821144e+01,  1.28518173e+01,  2.77896672e+00,
        1.26776134e+01,  1.72291772e+01,  1.67316154e+01,  4.71606824e+00,
        1.27796734e+01,  2.37266681e+01,  1.51795414e+01,  5.34664652e+00,
       -1.28791818e+01,  7.43191260e+00, -1.10450453e+01, -8.26786729e+00,
```

```
-3.94427934e+01, -2.28507081e+01,  3.70852429e+00,  1.12765988e+01,
-5.91250334e+00, -4.55408698e+01, -2.32559153e+01, -3.35162586e+01,
 1.51004824e+01,  2.56126352e+00,  9.52990954e+00,  2.13028507e+01,
 2.43178909e+00, -3.41382113e+01,  6.47210298e+00, -6.88972012e+00,
 5.13105495e+00, -2.29190675e+01,  1.58310125e+00,  6.89348827e+00,
-7.08185752e+00,  1.23088110e+01,  1.26828750e+00,  8.14075394e+00,
-3.89228152e-02,  9.50965284e+00,  1.27235591e+01,  1.13187244e+01,
 1.50055326e+01, -3.28112632e+01,  1.68627664e+01, -4.38015762e+01,
-1.22097163e+01, -9.72675198e+00, -7.69940223e+00,  1.53179401e+01,
 1.12300435e+01, -1.74634224e+01, -2.34655722e+01,  1.25053181e+01,
-9.08945780e+00, -4.50603637e+00,  1.48214754e+01,  6.96264439e+00,
 1.19726485e+01,  8.14611531e+00,  6.34271048e+00, -6.94841419e+00,
 4.11960531e+00,  1.19343170e+01, -1.20360604e+01,  1.28687256e+01,
-1.64368201e+01,  1.07522486e+01, -1.92784122e+01,  7.46433863e+00,
 2.33864489e+00,  3.07661614e+00, -1.80739462e+01,  5.55352547e+00,
-5.92798630e+01,  1.46897203e+01, -2.01244230e+01, -1.75070852e+01,
 8.90431224e+00,  6.70187138e+00,  1.16691550e+01,  9.92716695e+00,
 1.68692941e+01,  1.36730547e+00,  8.60868455e+00,  3.16835336e-01,
-1.09683781e+01, -1.94002056e+01,  9.03828820e+00,  9.71463032e+00,
-1.48208913e+01,  8.80741552e+00, -1.44991816e+01,  8.89170320e+00,
 1.51938265e+00, -1.86594174e+01,  9.74394957e+00,  1.06879612e-01,
 1.19007573e+01,  1.26250557e+01,  1.58292427e+01,  4.61622464e+00,
 2.87721041e+00,  1.19767989e+01, -7.29781796e+00, -9.49240693e+00,
 5.92303915e+00, -1.43031225e+01, -4.39064451e+00, -1.13038824e+00,
 7.71474549e+00, -1.61522892e+01,  1.51067592e+01,  7.20704143e+00,
 9.85850845e+00,  1.21876112e+01,  4.53890860e+00,  9.35781441e+00,
 6.90132296e+00,  6.66533365e+00, -9.11414869e+00, -2.22714381e+00,
 1.53262155e+01,  1.07859972e+01,  3.93723700e+00,  1.46669204e+01,
-6.89038443e+00,  1.14945017e+00, -4.01470624e-01,  1.16927064e+01,
-3.13019045e+01, -5.98766459e+00, -1.39963425e+01,  4.59926362e+00,
 1.32637738e+01, -9.45787337e+00, -1.49473220e+01,  9.80206416e+00,
-1.25848870e+01,  7.08387874e+00, -3.09146296e+01, -2.58957324e+01,
 8.43484297e+00,  1.03306146e+01,  1.07745293e+01, -2.01192372e+01,
-7.46904139e+00, -2.16606874e+01,  5.85198250e+00,  1.07827726e+01,
 8.63970558e+00,  7.72042193e+00,  1.62864665e+01, -2.07956135e+01,
 1.27291875e+01,  1.07094007e+01,  7.97750893e+00, -8.12687351e+00,
 1.08730040e+01, -4.14695860e+01,  1.45330826e+01,  2.26950928e+01,
-3.50354277e+01,  9.20420052e+00,  1.45428617e+01,  7.67848343e+00,
 4.26417060e+00,  5.89616058e+00,  5.56112242e+00, -2.22592144e+00,
-4.09902791e+01,  1.05136747e+01,  1.28681864e+01, -9.69926438e+00,
-8.78164112e+00,  9.95410449e+00,  1.73609654e+01,  1.00278251e+01,
 6.97509113e+00,  7.11677961e+00,  1.66383060e+01, -2.53535265e+01,
 2.63796479e+00, -1.47595736e+00,  1.23031010e+01,  7.18363308e+00,
 7.12552805e+00,  9.45557146e+00,  1.28629287e+01,  9.32927521e+00,
 1.14991324e+01, -1.14593354e+01, -7.86522192e+00,  1.24995055e+00,
 1.22632043e+01,  7.18034241e+00, -1.21296942e+01,  8.13756429e+00,
-9.69038081e+00,  9.11871286e+00,  7.70396140e+00,  9.94429260e+00,
 6.79697906e+00,  8.16130829e+00,  3.33717543e+00,  7.24161190e+00,
-1.13758930e+01,  3.60594959e+00, -4.38234847e+01,  4.65659821e+00,
 8.07450919e+00, -3.92802913e+01, -1.10053060e+01, -1.78847799e+01,
-2.08804778e+01,  5.94643910e+00,  1.05961240e+01, -7.83274746e+01,
 1.47297665e+01, -2.66404653e+01, -1.53843094e+00, -6.05743999e+00,
-7.72606535e+00,  9.26129187e+00, -5.83190748e+00,  1.08029084e+01,
-1.57598666e+00,  1.01776239e+01, -2.41303002e+00])
```

7. Now consider $y = \begin{cases} 1, & z \geq \delta \\ 0, & z < \delta \end{cases}$, where $\delta \in \mathbb{R}$ is threshold continuous value. Plot the **precision-recall curve**. Which threshold δ would you use to

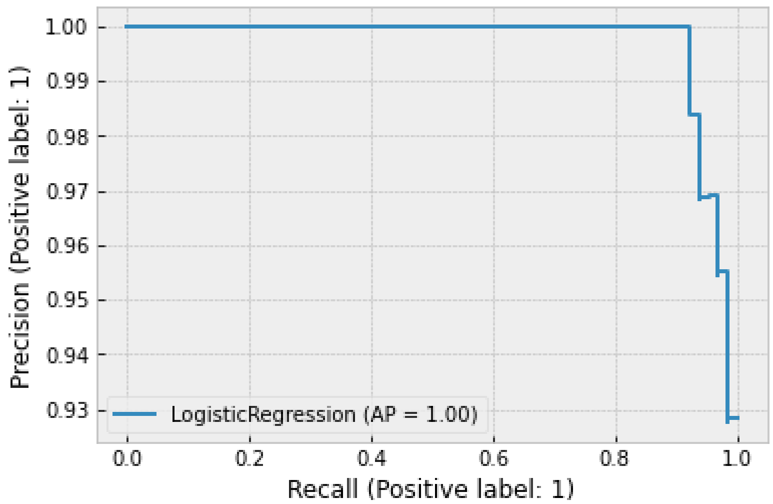
obtain a **recall of at least 80%**? Justify your answer based on these results.

In [658…
```
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import PrecisionRecallDisplay

precision, recall, thresholds = precision_recall_curve(t_train,y_scores)

recall_80_precision = recalls[np.argmax(precisions >= 0.80)]
threshold_80_precision = thresholds[np.argmax(precisions >= 0.80)]
precision_80_precision = precision[np.argmax(precisions >= 0.80)]

display = PrecisionRecallDisplay.from_estimator(final_model_log_reg, X_test,t_t
```



In [659…
```
threshold_80_precision = thresholds[np.argmax(precisions >= 0.80)]
threshold_80_precision
```

Out[659]:
```
-1.7490603985669015
```

In [660…
```
from sklearn.metrics import precision_score, recall_score

y_train_pred_80 = (y_scores >= threshold_80_precision)
precision_score(t_train, y_train_pred_80)
```

Out[660]:
```
0.948051948051948
```

In [661…
```
recall_score(t_train, y_train_pred_80)
```

Out[661]:
```
1.0
```

I would choose threshold of -1.749 due to its recall socre and precision score

8. Use the newly found threshold value to make predictions for the test set. Compare the results with those from part 4.

In [667…
```
#make predictions
threshold = -1.7490603985669015;
y_scores_test = cross_val_predict(final_model_log_reg, X_test, t_test,
                            cv=10, method='decision_function')
y_train = (y_scores > threshold)
y_test = (y_scores_test>threshold)
```

```
target_names = ['malignant', 'benign']
print(classification_report(t_train,y_train, target_names=target_names))
```

```
              precision    recall  f1-score   support

   malignant       0.99      0.90      0.95       163
      benign       0.95      1.00      0.97       292

    accuracy                           0.96       455
   macro avg       0.97      0.95      0.96       455
weighted avg       0.96      0.96      0.96       455
```

In [668…

```
print(classification_report(t_test,y_test, target_names=target_names))
```

```
              precision    recall  f1-score   support

   malignant       0.98      0.98      0.98        49
      benign       0.98      0.98      0.98        65

    accuracy                           0.98       114
   macro avg       0.98      0.98      0.98       114
weighted avg       0.98      0.98      0.98       114
```

The precision and recall scores for the test set is higher here than in part 4.

Question 2

In this problem you will work with the Immunotherapy dataset.

This dataset contains information about wart treatment results of 90 patients using immunotherapy. There are 7 features (sex, age, time, number of warts, type, area and induration diameter). The target label is the column "Result_of_Treatment", where 0 means not successful and 1 means the treatment was successful.

In [616…

```
import pandas as pd

df = pd.read_excel('Immunotherapy.xlsx')

df
```

Out[616]:

	sex	age	Time	Number_of_Warts	Type	Area	induration_diameter	Result_of_Treatment
0	1	22	2.25	14	3	51	50	1
1	1	15	3.00	2	3	900	70	1
2	1	16	10.50	2	1	100	25	1
3	1	27	4.50	9	3	80	30	1
4	1	20	8.00	6	1	45	8	1
...	...	...	...	...	...	...	...	...
85	1	40	5.50	8	3	69	5	1
86	1	38	7.50	8	2	56	45	1
87	1	46	11.50	4	1	91	25	0
88	1	32	12.00	9	1	43	50	0
89	2	23	6.75	6	1	19	2	1

90 rows × 8 columns

Answer the following questions:

1. Partition the data into training and test using a stratified 80/20 partition. For reproducible, fix the value for the `random_state` parameter.
2. Build a pipeline that includes data preprocessing and a decision tree classifier with `random_state=0`.
   - For data preprocessing, use one-hot encoding for categorical attributes (`sex` and `Type`) and min-max scaling for all other numerical attributes. Use the `ColumnTransform` function.
3. Carry hyperparameter tuning using grid search to experiment with `criterion`, `max_depth`, `min_samples_split` and `min_samples_leaf`. Train the final model pipeline.
4. Visualize the resulting decision tree.
5. Evaluate performance in training and test sets.

1. Partition the data into training and test using a stratified 80/20 partition. For reproducible, fix the value for the `random_state` parameter.

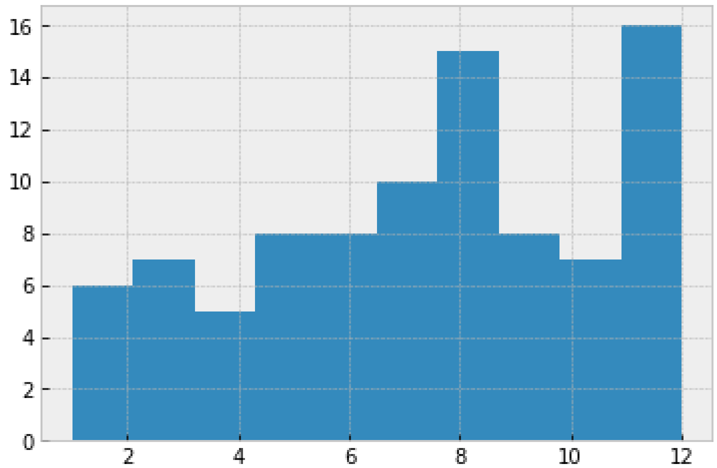
In [617…

```
corr_matrix = df.corr(method='pearson');
corr_matrix['Result_of_Treatment'].sort_values(ascending=False)
```

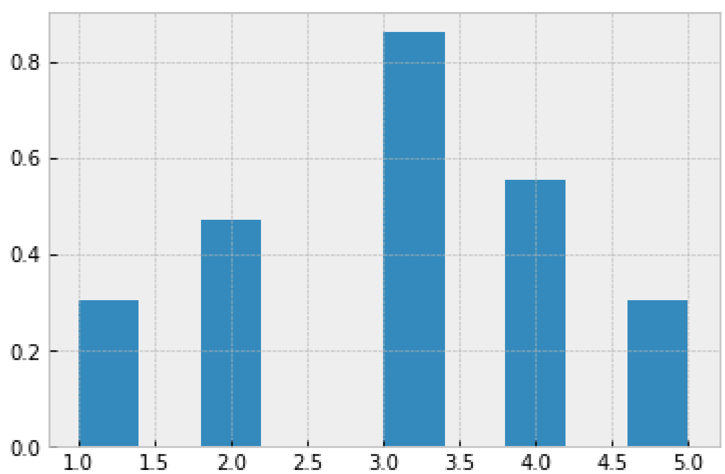
Out[617]:
```
Result_of_Treatment    1.000000
Type                   0.083396
Area                   0.043349
sex                    0.018831
induration_diameter   -0.031273
Number_of_Warts       -0.047160
age                   -0.188314
Time                  -0.361172
Name: Result_of_Treatment, dtype: float64
```

'Time' has the largest predictive value

In [618…
```
plt.hist(df['Time']);
```



In [619…
```
# train-test split with stratification
time_cat = pd.cut(df['Time'],
                  bins=[0.,2.5,5.5,8.5,11,np.inf],
                  labels=[1,2,3,4,5])
plt.hist(time_cat, density=True);
```



In [620…
```
train, test, cat_train, cat_test = train_test_split(df,
                                                    time_cat,
                                                    test_size=0.2,
                                                    shuffle=True,
                                                    random_state=42,
                                                   stratify=time_cat)
```

2. Build a pipeline that includes data preprocessing and a decision tree classifier with `random_state=0`.

- For data preprocessing, use one-hot encoding for categorical attributes (`sex` and `Type`) and min-max scaling for all other numerical attributes. Use the `ColumnTransform` function.

In [621…

```
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import MinMaxScaler
from sklearn.tree import DecisionTreeClassifier

num_train = train.drop(['sex','Type'], axis=1)

#attribute encoding
num_attribs = list(num_train.columns)
cat_attribs = ['sex','Type']

#pipeline for numerical attributes
num_pipeline = Pipeline([('min_max_scaler', MinMaxScaler())])

# complete pipeline
full_pipeline = ColumnTransformer([('num', num_pipeline, num_attribs),
                                   ('cat', OneHotEncoder(), cat_attribs)])

train_set_prepared = full_pipeline.fit_transform(train)
test_set_prepared = full_pipeline.transform(test)

# In pandas dataframe format
training_set = pd.DataFrame(train_set_prepared,
                            columns=np.hstack((num_attribs,
                                               ['Cat1','Cat2','Cat3','Cat4','Cat
                            index=train.index)
X_train = training_set.drop('Result_of_Treatment', axis=1).to_numpy()
t_train = training_set['Result_of_Treatment'].to_numpy()

test_set = pd.DataFrame(test_set_prepared,
                            columns=np.hstack((num_attribs,
                                               ['Cat1','Cat2','Cat3','Cat4','Cat
                            index=test.index)

X_test = test_set.drop('Result_of_Treatment', axis=1).to_numpy()
t_test = test_set['Result_of_Treatment'].to_numpy()
```

In [622…

```
X_train.shape, t_train.shape, X_test.shape, t_test.shape
```

Out[622]:

```
((72, 10), (72,), (18, 10), (18,))
```

In [623…

```
DTree = DecisionTreeClassifier(random_state=0)
dt_pipe=Pipeline([('dt', DTree)])
```

3. Carry hyperparameter tuning using grid search to experiment with `criterion`, `max_depth`, `min_samples_split` and `min_samples_leaf`. Train the final model pipeline.

In [624…

```
param_grid={'dt__criterion':['gini', 'entropy'],
```

```
            'dt__max_depth':np.arange(1,30),
            'dt__min_samples_split':np.arange(2,10),
            'dt__min_samples_leaf':np.arange(1,10)}
grid_search_dt = GridSearchCV(dt_pipe,
                             param_grid=param_grid,
                             cv=5,
                             scoring='neg_mean_squared_error',
                             refit=True)
grid_search_dt.fit(X_train, t_train);

print(grid_search_dt.best_params_)
```

```
{'dt__criterion': 'gini', 'dt__max_depth': 2, 'dt__min_samples_leaf': 4, 'dt__
min_samples_split': 2}
```

In [625…

```
final_dt = DecisionTreeClassifier(criterion='gini', max_depth=2,
                                 min_samples_leaf=4, min_samples_split=2,
                                 random_state=0)
```

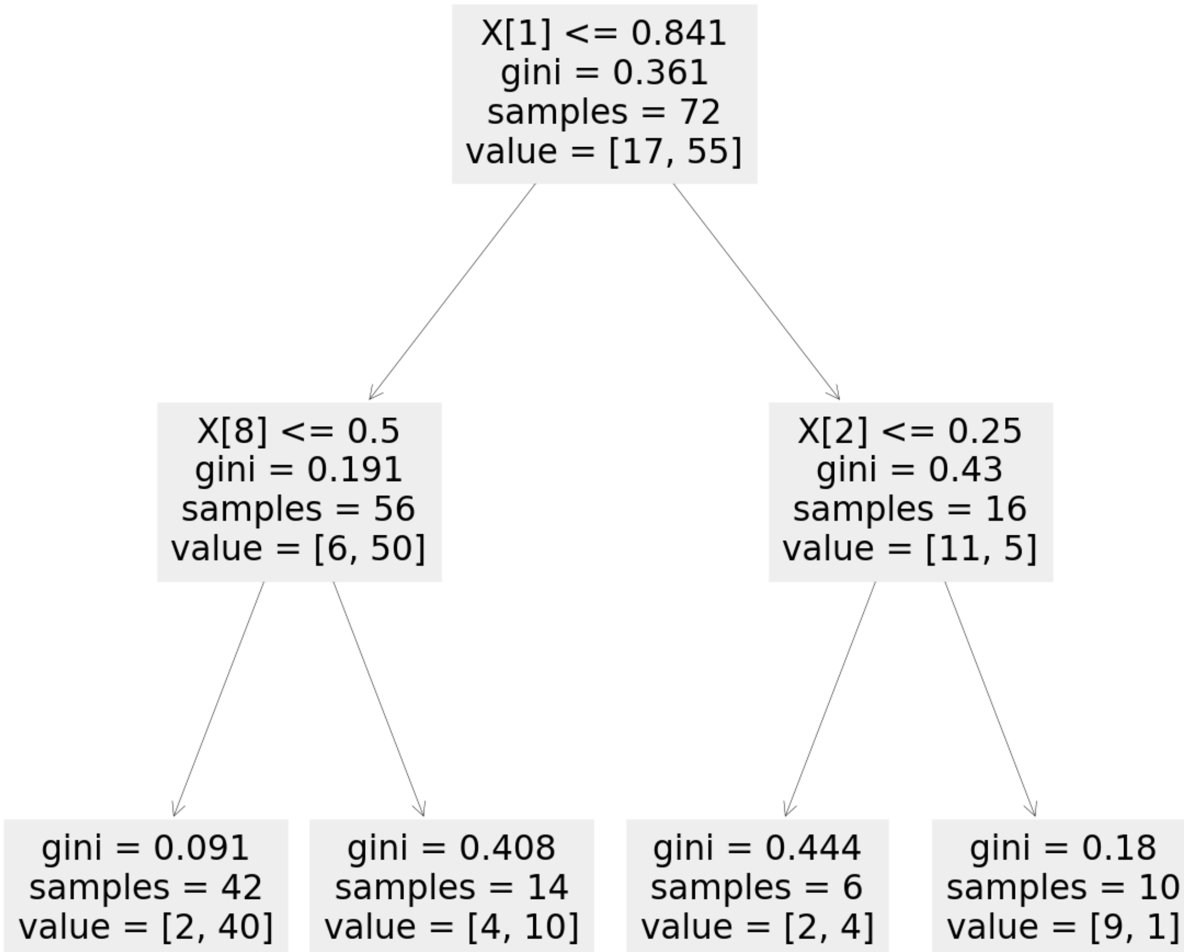
4. Visualize the resulting decision tree.

In [626…

```
from sklearn import tree

plt.figure(figsize=(20,20))

tree.plot_tree(final_dt.fit(X_train, t_train));
plt.show()
```



5. Evaluate performance in training and test sets.

In [627…

```
y_train = final_dt.predict(X_train)
y_test = final_dt.predict(X_test)
target_names = ['not successful', 'successful']
print(classification_report(t_train,y_train, target_names=target_names))
```

```
                precision    recall  f1-score   support

not successful       0.90      0.53      0.67        17
    successful       0.87      0.98      0.92        55

      accuracy                           0.88        72
     macro avg       0.89      0.76      0.79        72
  weighted avg       0.88      0.88      0.86        72
```

In [628…

```
print(classification_report(t_test,y_test, target_names=target_names))
```

```
                 precision    recall  f1-score   support

not successful       1.00      0.50      0.67         2
    successful       0.94      1.00      0.97        16

      accuracy                           0.94        18
     macro avg       0.97      0.75      0.82        18
  weighted avg       0.95      0.94      0.94        18
```

Question 3

In this problem you will be working with the Digits dataset.

Each sample corresponds to an 8×8 gray image of a handwritten digit. There is a total of 10 digits or labels $(0, 1, 2, \ldots, 9)$. The dataset contains 1797 samples.

In [629…

```
from sklearn.datasets import load_digits

digits = load_digits(return_X_y=False)

print(digits.DESCR)
```

```
.. _digits_dataset:

Optical recognition of handwritten digits dataset
--------------------------------------------------

**Data Set Characteristics:**

    :Number of Instances: 1797
    :Number of Attributes: 64
    :Attribute Information: 8x8 image of integer pixels in the range 0..16.
    :Missing Attribute Values: None
    :Creator: E. Alpaydin (alpaydin '@' boun.edu.tr)
    :Date: July; 1998

This is a copy of the test set of the UCI ML hand-written digits datasets
https://archive.ics.uci.edu/ml/datasets/Optical+Recognition+of+Handwritten+Dig
its

The data set contains images of hand-written digits: 10 classes where
each class refers to a digit.

Preprocessing programs made available by NIST were used to extract
normalized bitmaps of handwritten digits from a preprinted form. From a
total of 43 people, 30 contributed to the training set and different 13
to the test set. 32x32 bitmaps are divided into nonoverlapping blocks of
4x4 and the number of on pixels are counted in each block. This generates
an input matrix of 8x8 where each element is an integer in the range
0..16. This reduces dimensionality and gives invariance to small
distortions.

For info on NIST preprocessing routines, see M. D. Garris, J. L. Blue, G.
T. Candela, D. L. Dimmick, J. Geist, P. J. Grother, S. A. Janet, and C.
L. Wilson, NIST Form-Based Handprint Recognition System, NISTIR 5469,
1994.

.. topic:: References

  - C. Kaynak (1995) Methods of Combining Multiple Classifiers and Their
    Applications to Handwritten Digit Recognition, MSc Thesis, Institute of
    Graduate Studies in Science and Engineering, Bogazici University.
  - E. Alpaydin, C. Kaynak (1998) Cascading Classifiers, Kybernetika.
  - Ken Tang and Ponnuthurai N. Suganthan and Xi Yao and A. Kai Qin.
    Linear dimensionalityreduction using relevance weighted LDA. School of
    Electrical and Electronic Engineering Nanyang Technological University.
    2005.
  - Claudio Gentile. A New Approximate Maximal Margin Classification
    Algorithm. NIPS. 2000.
```

In [630…

```
# Obtaining data

# Each row corresponds to an image with 8x8=64 pixels/features
X = digits.data

# Labels
t = digits.target

X.shape, t.shape
```

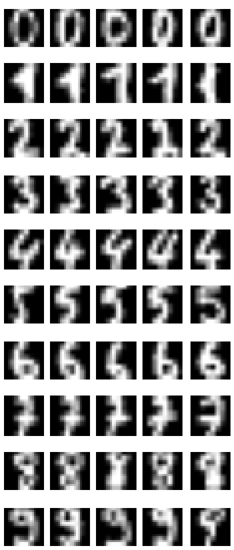
Out[630]: ((1797, 64), (1797,))

In [631…

```
# displaying some examples of digits

import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
plt.style.use('bmh')

plt.figure(figsize=(2,5))
grid_loc=1
for i in range(10):
    digits_labels_idx = np.where(t==i)[0]
    idx = np.random.randint(len(digits_labels_idx),size=5)
    for j in range(5):
        plt.subplot(10,5,grid_loc)
        plt.imshow(X[digits_labels_idx[idx[j]],:].reshape(8,8), cmap='gray')
        plt.axis('off')
        grid_loc+=1
```



Answer the following questions:

1. Partition the data into training and test using a stratified 80/20 partition. For reproducible, fix the value for the `random_state` parameter.

2. Build a pipeline that includes data preprocessing and a random forest classifier with `random_state=0`.

   - For data preprocessing, use the min-max scaler.

3. Carry hyperparameter tuning using grid search to experiment with number of trees, `criterion`, `max_depth`, `min_samples_split` and `min_samples_leaf`. Train the final model pipeline.

4. Print the `feature_importance_` for the final model. Reshape this vector as an 8×8 image and display it with `imshow`. Discuss observations.

5. Evaluate performance in training and test sets.

1. Partition the data into training and test using a stratified 80/20 partition. For reproducible, fix the value for the `random_state` parameter.

In [632…

```
X_train, X_test, t_train, t_test = train_test_split(X,t,
                                                    test_size=0.2,
                                                    shuffle=True,
                                                    random_state=42)
```

2. Build a pipeline that includes data preprocessing and a random forest classifier with `random_state=0`.

    * For data preprocessing, use the min-max scaler.

In [633…

```
from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier(random_state=0)
pipe = Pipeline([('min_max_scaler', MinMaxScaler()),
                 ('model', model)])
pipe.get_params()
```

Out[633]:

```
{'memory': None,
 'steps': [('min_max_scaler', MinMaxScaler()),
  ('model', RandomForestClassifier(random_state=0))],
 'verbose': False,
 'min_max_scaler': MinMaxScaler(),
 'model': RandomForestClassifier(random_state=0),
 'min_max_scaler__clip': False,
 'min_max_scaler__copy': True,
 'min_max_scaler__feature_range': (0, 1),
 'model__bootstrap': True,
 'model__ccp_alpha': 0.0,
 'model__class_weight': None,
 'model__criterion': 'gini',
 'model__max_depth': None,
 'model__max_features': 'auto',
 'model__max_leaf_nodes': None,
 'model__max_samples': None,
 'model__min_impurity_decrease': 0.0,
 'model__min_samples_leaf': 1,
 'model__min_samples_split': 2,
 'model__min_weight_fraction_leaf': 0.0,
 'model__n_estimators': 100,
 'model__n_jobs': None,
 'model__oob_score': False,
 'model__random_state': 0,
 'model__verbose': 0,
 'model__warm_start': False}
```

3. Carry hyperparameter tuning using grid search to experiment with number of trees, `criterion`, `max_depth`, `min_samples_split` and `min_samples_leaf`. Train the final model pipeline.

In [634…

```
param_grid={'model__criterion':['gini', 'entropy'],
            'model__max_depth':np.arange(2,10),
            'model__min_samples_split':np.arange(2,5),
```

```
            'model__min_samples_leaf':np.arange(1,5)}

grid_search_rfc = GridSearchCV(pipe,
                              param_grid=param_grid,
                              cv=5,
                              scoring='neg_mean_squared_error')
grid_search_rfc.fit(X_train, t_train);
print(grid_search_rfc.best_params_)
```

```
{'model__criterion': 'entropy', 'model__max_depth': 8, 'model__min_samples_lea
f': 1, 'model__min_samples_split': 3}
```

In [635…

```
final_model_rfc = grid_search_rfc.best_estimator_
```

4. Print the `feature_importance_` for the final model. Reshape this vector as an 8×8 image and display it with `imshow`. Discuss observations.

In [636…

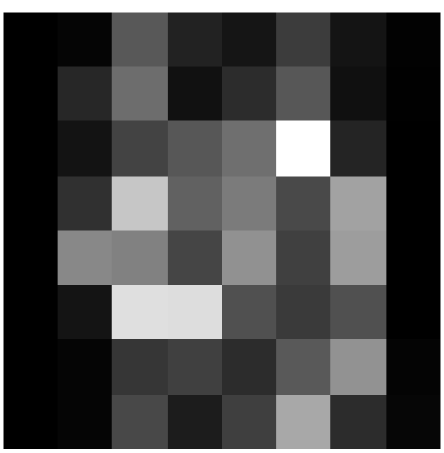
```
final_model_rfc.named_steps['model'].feature_importances_
```

Out[636]:

```
array([0.00000000e+00, 1.41042371e-03, 2.20375529e-02, 8.49282230e-03,
       5.33277452e-03, 1.53930078e-02, 5.12094704e-03, 5.16842874e-04,
       3.18684693e-05, 9.78979320e-03, 2.71522726e-02, 4.35634583e-03,
       1.12781735e-02, 2.17366146e-02, 4.03389123e-03, 2.52951783e-04,
       1.47272174e-05, 4.88367753e-03, 1.67158274e-02, 2.16712512e-02,
       2.77482675e-02, 6.36081548e-02, 9.28632520e-03, 1.50195146e-04,
       5.71867403e-05, 1.19440382e-02, 4.93673654e-02, 2.45121393e-02,
       3.07460272e-02, 1.82314851e-02, 4.04562410e-02, 7.75992657e-05,
       0.00000000e+00, 3.37982742e-02, 3.20568245e-02, 1.73545686e-02,
       3.60411487e-02, 1.61058220e-02, 3.91499075e-02, 0.00000000e+00,
       2.45568749e-05, 5.21364812e-03, 5.55069987e-02, 5.50690069e-02,
       1.99956594e-02, 1.44611185e-02, 1.99223988e-02, 1.56325840e-05,
       5.39318879e-05, 1.28562555e-03, 1.35631201e-02, 1.59619355e-02,
       1.11714863e-02, 2.25394374e-02, 3.63328229e-02, 1.16851660e-03,
       0.00000000e+00, 1.47463514e-03, 1.79279467e-02, 7.04712683e-03,
       1.57558488e-02, 4.17516829e-02, 1.11300903e-02, 1.71343721e-03])
```

In [637…

```
img = final_model_rfc.named_steps['model'].feature_importances_.reshape(8,8);
plt.imshow(img, cmap='gray')
plt.axis('off');
```



The image is hard to read but it looks like a 8, since most of its concentration is on the center of the loop

5. Evaluate performance in training and test sets.

In [638…

```
y_test = final_model_rfc.predict(X_test)
y_train = final_model_rfc.predict(X_train)
print(classification_report(t_train,y_train))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       145
           1       1.00      1.00      1.00       154
           2       1.00      1.00      1.00       144
           3       1.00      1.00      1.00       149
           4       1.00      1.00      1.00       135
           5       1.00      0.99      1.00       135
           6       1.00      1.00      1.00       146
           7       1.00      1.00      1.00       145
           8       0.99      1.00      1.00       144
           9       0.99      0.99      0.99       140

    accuracy                           1.00      1437
   macro avg       1.00      1.00      1.00      1437
weighted avg       1.00      1.00      1.00      1437
```

In [639…

```
print(classification_report(t_test,y_test))
```

```
              precision    recall  f1-score   support

           0       1.00      0.97      0.98        33
           1       1.00      1.00      1.00        28
           2       1.00      1.00      1.00        33
           3       1.00      1.00      1.00        34
           4       0.98      1.00      0.99        46
           5       0.96      0.96      0.96        47
           6       0.97      0.97      0.97        35
           7       0.94      0.97      0.96        34
           8       1.00      1.00      1.00        30
           9       0.95      0.93      0.94        40

    accuracy                           0.98       360
   macro avg       0.98      0.98      0.98       360
weighted avg       0.98      0.98      0.98       360
```

The precision score seems to be higher than the recall for both cases.

Submit Your Solution

Confirm that you've successfully completed the assignment.

Along with the Notebook, include a PDF of the notebook with your solutions.

`add` and `commit` the final version of your work, and `push` your code to your GitHub repository.

Submit the URL of your GitHub Repository as your assignment submission on Canvas.
