assignment2

October 2, 2023

You are currently looking at ***version 0.1*** *of this notebook. To download notebooks and datafiles, as well as get help on Jupyter notebooks in the Coursera platform, visit the Jupyter Notebook FAQ course resource.*

---

1 Assignment 2

In this assignment you'll explore the relationship between model complexity and generalization performance, by adjusting key parameters of various supervised learning models. Part 1 of this assignment will look at regression and Part 2 will look at classification.

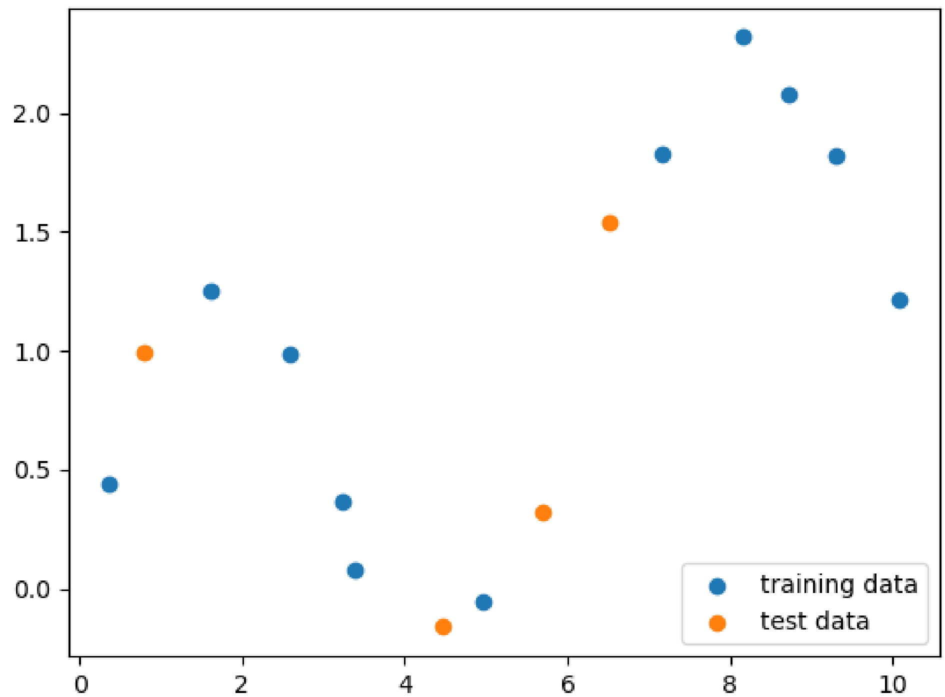
1.1 Part 1 - Regression

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split


np.random.seed(0)
n = 15
x = np.linspace(0,10,n) + np.random.randn(n)/5
y = np.sin(x)+x/6 + np.random.randn(n)/10


X_train, X_test, y_train, y_test = train_test_split(x, y, random_state=0)

def intro():
    %matplotlib notebook

    plt.figure()
    plt.scatter(X_train, y_train, label='training data')
    plt.scatter(X_test, y_test, label='test data')
    plt.legend(loc=4);

intro()
```

<IPython.core.display.Javascript object>

<IPython.core.display.HTML object>

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split


np.random.seed(0)
n = 15
x = np.linspace(0,10,n) + np.random.randn(n)/5
y = np.sin(x)+x/6 + np.random.randn(n)/10


X_train, X_test, y_train, y_test = train_test_split(x, y, random_state=0)

def intro():
    %matplotlib widget

    plt.figure()
    plt.scatter(X_train, y_train, label='training data')
    plt.scatter(X_test, y_test, label='test data')
    plt.legend(loc=4);

intro()
```



[2]:
```
x
```

[2]:
```
array([ 0.35281047,  0.79431716,  1.62431903,  2.59103578,  3.23065446,
        3.375973  ,  4.47573197,  4.96972856,  5.69364194,  6.51069113,
        7.17166586,  8.14799756,  8.72363612,  9.31004929, 10.08877265])
```

[3]:
```
y
```

[3]:
```
array([ 0.43770571,  0.99517935,  1.24877201,  0.98630796,  0.36408873,
        0.07512287, -0.16081   , -0.05233879,  0.3187423 ,  1.53763897,
        1.82595557,  2.31966323,  2.08031157,  1.81942995,  1.21213026])
```

[4]:
```
X_train
```

[4]:
```
array([10.08877265,  3.23065446,  1.62431903,  9.31004929,  7.17166586,
        4.96972856,  8.14799756,  2.59103578,  0.35281047,  3.375973  ,
        8.72363612])
```

[5]:
```
X_test
```

[5]:
```
array([0.79431716, 4.47573197, 5.69364194, 6.51069113])
```

[6]:
```
y_train
```

[6]:
```
array([ 1.21213026,  0.36408873,  1.24877201,  1.81942995,  1.82595557,
       -0.05233879,  2.31966323,  0.98630796,  0.43770571,  0.07512287,
        2.08031157])
```

[7]:
```
y_test
```

[7]:
```
array([ 0.99517935, -0.16081   ,  0.3187423 ,  1.53763897])
```

1.1.1 Question 1

Write a function that fits a polynomial LinearRegression model on the *training data* `X_train` for degrees 1, 3, 6, and 9. (Use PolynomialFeatures in sklearn.preprocessing to create the polynomial features and then fit a linear regression model) For each model, find 100 predicted values over the interval x = 0 to 10 (e.g. `np.linspace(0,10,100)`) and store this in a numpy array. The first row of this array should correspond to the output from the model trained on degree 1, the second row degree 3, the third row degree 6, and the fourth row degree 9.

The figure above shows the fitted models plotted on top of the original data (using `plot_one()`).

This function should return a numpy array with shape (4, 100)

[8]:
```
def answer_one():
#     from sklearn.linear_model import LinearRegression
#     from sklearn.preprocessing import PolynomialFeatures
#     import numpy as np
#     import pandas as pd
#     import matplotlib.pyplot as plt
#     from sklearn.model_selection import train_test_split

#     np.random.seed(0)
#     n = 15
#     x = np.linspace(0,10,n) + np.random.randn(n)/5
#     y = np.sin(x)+x/6 + np.random.randn(n)/10

#     X_train, X_test, y_train, y_test = train_test_split(x, y, random_state=0)

#     # degree_predictions = np.zeros((4,100))

#     # YOUR CODE HERE
#     poly_1 = PolynomialFeatures(degree=1)
#     poly_3 = PolynomialFeatures(degree=3)
#     poly_6 = PolynomialFeatures(degree=6)
#     poly_9 = PolynomialFeatures(degree=9)

#     X_train = X_train.reshape(-1,1)
```

```
#     X_poly_1 = poly_1.fit_transform(X_train)
#     X_poly_3 = poly_3.fit_transform(X_train)
#     X_poly_6 = poly_6.fit_transform(X_train)
#     X_poly_9 = poly_9.fit_transform(X_train)

#     model_1 = LinearRegression()
#     model_1.fit(X_poly_1, y_train)

#     model_3 = LinearRegression()
#     model_3.fit(X_poly_3, y_train)

#     model_6 = LinearRegression()
#     model_6.fit(X_poly_6, y_train)

#     model_9 = LinearRegression()
#     model_9.fit(X_poly_9, y_train)

#     test_numbers = np.linspace(1,10,100)
#     test_numbers = test_numbers.reshape(-1,1)

#     predictions_1 = model_1.predict(poly_1.transform(test_numbers))
#     predictions_3 = model_3.predict(poly_3.transform(test_numbers))
#     predictions_6 = model_6.predict(poly_6.transform(test_numbers))
#     predictions_9 = model_9.predict(poly_9.transform(test_numbers))

#     predictions_1 = predictions_1.reshape(1, -1)
#     predictions_3 = predictions_3.reshape(1, -1)
#     predictions_6 = predictions_6.reshape(1, -1)
#     predictions_9 = predictions_9.reshape(1, -1)

#     return np.concatenate((predictions_1, predictions_3, predictions_6,␣
↪predictions_9), axis = 0)

# ABOVE CODE ACHIEVES THE SAME THING, BUT BELOW CODE IS MORE EFFICIENT

    from sklearn.linear_model import LinearRegression
    from sklearn.preprocessing import PolynomialFeatures

    DEGREES = [1, 3, 6, 9]
    N_POINTS = 100
    result = np.zeros([len(DEGREES), N_POINTS])
    predict = np.linspace(0, 10, N_POINTS).reshape(-1, 1)
    X_tr = X_train.reshape(-1, 1)

    for i, deg in enumerate(DEGREES):

        poly = PolynomialFeatures(degree=deg)
```

```
        X_ = poly.fit_transform(X_tr)
        predict_ = poly.fit_transform(predict)
        reg = LinearRegression()
        reg.fit(X_, y_train)
        result[i, :] = reg.predict(predict_)

    return result

    raise NotImplementedError()
```

[9]:
```
answer_one()
```

[9]:
```
array([[ 2.53040195e-01,  2.69201547e-01,  2.85362899e-01,
         3.01524251e-01,  3.17685603e-01,  3.33846955e-01,
         3.50008306e-01,  3.66169658e-01,  3.82331010e-01,
         3.98492362e-01,  4.14653714e-01,  4.30815066e-01,
         4.46976417e-01,  4.63137769e-01,  4.79299121e-01,
         4.95460473e-01,  5.11621825e-01,  5.27783177e-01,
         5.43944529e-01,  5.60105880e-01,  5.76267232e-01,
         5.92428584e-01,  6.08589936e-01,  6.24751288e-01,
         6.40912640e-01,  6.57073992e-01,  6.73235343e-01,
         6.89396695e-01,  7.05558047e-01,  7.21719399e-01,
         7.37880751e-01,  7.54042103e-01,  7.70203454e-01,
         7.86364806e-01,  8.02526158e-01,  8.18687510e-01,
         8.34848862e-01,  8.51010214e-01,  8.67171566e-01,
         8.83332917e-01,  8.99494269e-01,  9.15655621e-01,
         9.31816973e-01,  9.47978325e-01,  9.64139677e-01,
         9.80301028e-01,  9.96462380e-01,  1.01262373e+00,
         1.02878508e+00,  1.04494644e+00,  1.06110779e+00,
         1.07726914e+00,  1.09343049e+00,  1.10959184e+00,
         1.12575320e+00,  1.14191455e+00,  1.15807590e+00,
         1.17423725e+00,  1.19039860e+00,  1.20655995e+00,
         1.22272131e+00,  1.23888266e+00,  1.25504401e+00,
         1.27120536e+00,  1.28736671e+00,  1.30352807e+00,
         1.31968942e+00,  1.33585077e+00,  1.35201212e+00,
         1.36817347e+00,  1.38433482e+00,  1.40049618e+00,
         1.41665753e+00,  1.43281888e+00,  1.44898023e+00,
         1.46514158e+00,  1.48130294e+00,  1.49746429e+00,
         1.51362564e+00,  1.52978699e+00,  1.54594834e+00,
         1.56210969e+00,  1.57827105e+00,  1.59443240e+00,
         1.61059375e+00,  1.62675510e+00,  1.64291645e+00,
         1.65907781e+00,  1.67523916e+00,  1.69140051e+00,
         1.70756186e+00,  1.72372321e+00,  1.73988457e+00,
         1.75604592e+00,  1.77220727e+00,  1.78836862e+00,
         1.80452997e+00,  1.82069132e+00,  1.83685268e+00,
         1.85301403e+00],
       [ 1.22989539e+00,  1.15143628e+00,  1.07722393e+00,
```

```
  1.00717881e+00,  9.41221419e-01,  8.79272234e-01,
  8.21251741e-01,  7.67080426e-01,  7.16678772e-01,
  6.69967266e-01,  6.26866391e-01,  5.87296632e-01,
  5.51178474e-01,  5.18432402e-01,  4.88978901e-01,
  4.62738455e-01,  4.39631549e-01,  4.19578668e-01,
  4.02500297e-01,  3.88316920e-01,  3.76949022e-01,
  3.68317088e-01,  3.62341603e-01,  3.58943051e-01,
  3.58041918e-01,  3.59558687e-01,  3.63413845e-01,
  3.69527874e-01,  3.77821261e-01,  3.88214491e-01,
  4.00628046e-01,  4.14982414e-01,  4.31198078e-01,
  4.49195522e-01,  4.68895233e-01,  4.90217694e-01,
  5.13083391e-01,  5.37412808e-01,  5.63126429e-01,
  5.90144741e-01,  6.18388226e-01,  6.47777371e-01,
  6.78232660e-01,  7.09674578e-01,  7.42023609e-01,
  7.75200238e-01,  8.09124950e-01,  8.43718230e-01,
  8.78900563e-01,  9.14592432e-01,  9.50714324e-01,
  9.87186723e-01,  1.02393011e+00,  1.06086498e+00,
  1.09791181e+00,  1.13499108e+00,  1.17202328e+00,
  1.20892890e+00,  1.24562842e+00,  1.28204233e+00,
  1.31809110e+00,  1.35369523e+00,  1.38877520e+00,
  1.42325149e+00,  1.45704459e+00,  1.49007498e+00,
  1.52226316e+00,  1.55352959e+00,  1.58379478e+00,
  1.61297919e+00,  1.64100332e+00,  1.66778766e+00,
  1.69325268e+00,  1.71731887e+00,  1.73990672e+00,
  1.76093671e+00,  1.78032933e+00,  1.79800506e+00,
  1.81388438e+00,  1.82788778e+00,  1.83993575e+00,
  1.84994877e+00,  1.85784732e+00,  1.86355189e+00,
  1.86698296e+00,  1.86806103e+00,  1.86670656e+00,
  1.86284006e+00,  1.85638200e+00,  1.84725286e+00,
  1.83537314e+00,  1.82066332e+00,  1.80304388e+00,
  1.78243530e+00,  1.75875808e+00,  1.73193269e+00,
  1.70187963e+00,  1.66851936e+00,  1.63177240e+00,
  1.59155920e+00],
[-1.99554310e-01, -3.95192721e-03,  1.79851753e-01,
  3.51005136e-01,  5.08831706e-01,  6.52819233e-01,
  7.82609240e-01,  8.97986721e-01,  9.98870117e-01,
  1.08530155e+00,  1.15743729e+00,  1.21553852e+00,
  1.25996233e+00,  1.29115292e+00,  1.30963316e+00,
  1.31599632e+00,  1.31089811e+00,  1.29504889e+00,
  1.26920626e+00,  1.23416782e+00,  1.19076415e+00,
  1.13985218e+00,  1.08230867e+00,  1.01902405e+00,
  9.50896441e-01,  8.78825970e-01,  8.03709344e-01,
  7.26434655e-01,  6.47876457e-01,  5.68891088e-01,
  4.90312256e-01,  4.12946874e-01,  3.37571147e-01,
  2.64926923e-01,  1.95718291e-01,  1.30608438e-01,
  7.02167560e-02,  1.51162118e-02, -3.41690365e-02,
 -7.71657635e-02, -1.13453547e-01, -1.42666382e-01,
```

```
 -1.64494044e-01, -1.78683194e-01, -1.85038228e-01,
 -1.83421873e-01, -1.73755533e-01, -1.56019368e-01,
 -1.30252132e-01, -9.65507462e-02, -5.50696231e-02,
 -6.01973195e-03,  5.03325883e-02,  1.13667071e-01,
  1.83611221e-01,  2.59742264e-01,  3.41589357e-01,
  4.28636046e-01,  5.20322987e-01,  6.16050916e-01,
  7.15183874e-01,  8.17052690e-01,  9.20958717e-01,
  1.02617782e+00,  1.13196463e+00,  1.23755703e+00,
  1.34218093e+00,  1.44505526e+00,  1.54539723e+00,
  1.64242789e+00,  1.73537785e+00,  1.82349336e+00,
  1.90604254e+00,  1.98232198e+00,  2.05166348e+00,
  2.11344114e+00,  2.16707864e+00,  2.21205680e+00,
  2.24792141e+00,  2.27429129e+00,  2.29086658e+00,
  2.29743739e+00,  2.29389257e+00,  2.28022881e+00,
  2.25656001e+00,  2.22312684e+00,  2.18030664e+00,
  2.12862347e+00,  2.06875850e+00,  2.00156065e+00,
  1.92805743e+00,  1.84946605e+00,  1.76720485e+00,
  1.68290491e+00,  1.59842194e+00,  1.51584842e+00,
  1.43752602e+00,  1.36605824e+00,  1.30432333e+00,
  1.25548743e+00],
[ 6.79501877e+00,  4.14319714e+00,  2.23123195e+00,
  9.10495039e-01,  5.49803327e-02, -4.41344174e-01,
 -6.66950030e-01, -6.94942449e-01, -5.85049217e-01,
 -3.85418102e-01, -1.34235851e-01,  1.38818670e-01,
  4.11275215e-01,  6.66715368e-01,  8.93747315e-01,
  1.08510182e+00,  1.23683955e+00,  1.34766042e+00,
  1.41830603e+00,  1.45104693e+00,  1.44924662e+00,
  1.41699501e+00,  1.35880409e+00,  1.27935949e+00,
  1.18332142e+00,  1.07516952e+00,  9.59085935e-01,
  8.38871928e-01,  7.17893068e-01,  5.99048939e-01,
  4.84763319e-01,  3.76991254e-01,  2.77239711e-01,
  1.86598853e-01,  1.05781227e-01,  3.51664578e-02,
 -2.51506700e-02, -7.53106421e-02, -1.15639771e-01,
 -1.46602278e-01, -1.68755085e-01, -1.82706256e-01,
 -1.89077879e-01, -1.88473948e-01, -1.81453660e-01,
 -1.68510358e-01, -1.50056232e-01, -1.26412707e-01,
 -9.78063691e-02, -6.43701362e-02, -2.61492812e-02,
  1.68881553e-02,  6.48371253e-02,  1.17838121e-01,
  1.76057180e-01,  2.39664066e-01,  3.08809354e-01,
  3.83601193e-01,  4.64082501e-01,  5.50209339e-01,
  6.41831222e-01,  7.38674048e-01,  8.40326319e-01,
  9.46229255e-01,  1.05567134e+00,  1.16778775e+00,
  1.28156502e+00,  1.39585128e+00,  1.50937206e+00,
  1.62075184e+00,  1.72854110e+00,  1.83124870e+00,
  1.92737900e+00,  2.01547327e+00,  2.09415450e+00,
  2.16217452e+00,  2.21846241e+00,  2.26217255e+00,
  2.29273075e+00,  2.30987650e+00,  2.31369910e+00,
```

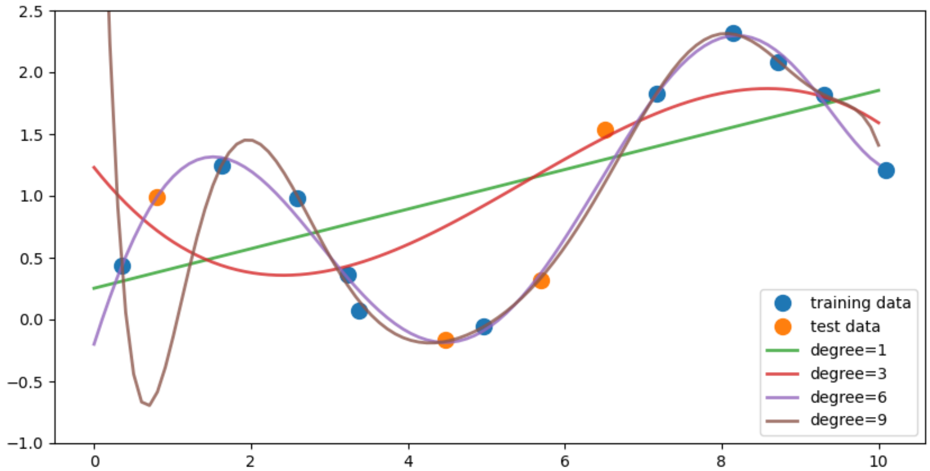
```
       2.30466527e+00,  2.28363544e+00,  2.25186569e+00,
       2.21099194e+00,  2.16299281e+00,  2.11012698e+00,
       2.05484079e+00,  1.99964137e+00,  1.94693015e+00,
       1.89879129e+00,  1.85672916e+00,  1.82134864e+00,
       1.79197149e+00,  1.76618168e+00,  1.73929211e+00,
       1.70372475e+00,  1.64829557e+00,  1.55739548e+00,
       1.41005768e+00]])
```

[10]:

```python
def answer_one_test():
    from sklearn.linear_model import LinearRegression
    from sklearn.preprocessing import PolynomialFeatures

    DEGREES = [1, 3, 6, 9]
    N_POINTS = 100
    result = np.zeros([len(DEGREES), N_POINTS])
    predict = np.linspace(0, 10, N_POINTS).reshape(-1, 1)
    X_tr = X_train.reshape(-1, 1)

    for i, deg in enumerate(DEGREES):

        poly = PolynomialFeatures(degree=deg)
        X_ = poly.fit_transform(X_tr)
        predict_ = poly.fit_transform(predict)
        reg = LinearRegression()
        reg.fit(X_, y_train)
        result[i, :] = reg.predict(predict_)

    return result
```

[11]:

```python
# feel free to use the function plot_one() to replicate the figure
# from the prompt once you have completed question one
def plot_one(degree_predictions):
    plt.figure(figsize=(10,5))
    plt.plot(X_train, y_train, 'o', label='training data', markersize=10)
    plt.plot(X_test, y_test, 'o', label='test data', markersize=10)
    for i,degree in enumerate([1,3,6,9]):
        plt.plot(np.linspace(0,10,100), degree_predictions[i], alpha=0.8, lw=2,␣
 ↪label='degree={}'.format(degree))
    plt.ylim(-1,2.5)
    plt.legend(loc=4)

plot_one(answer_one())
```



[12]:
```python
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures

poly_1 = PolynomialFeatures(degree=1)

X_train = X_train.reshape(-1,1)

test_numbers = np.linspace(0,10,100)
test_numbers = test_numbers.reshape(-1,1)

X_poly_1 = poly_1.fit_transform(X_train)

test_poly_1 = poly_1.fit_transform(test_numbers)

model_1 = LinearRegression()
model_1.fit(X_poly_1, y_train)

predictions_1 = model_1.predict(test_poly_1)

predictions_1 = predictions_1.reshape(1, -1)

predictions_1
```

[12]:
```
array([[0.2530402 , 0.26920155, 0.2853629 , 0.30152425, 0.3176856 ,
        0.33384695, 0.35000831, 0.36616966, 0.38233101, 0.39849236,
        0.41465371, 0.43081507, 0.44697642, 0.46313777, 0.47929912,
        0.49546047, 0.51162182, 0.52778318, 0.54394453, 0.56010588,
        0.57626723, 0.59242858, 0.60858994, 0.62475129, 0.64091264,
```

```
       0.65707399, 0.67323534, 0.6893967 , 0.70555805, 0.7217194 ,
       0.73788075, 0.7540421 , 0.77020345, 0.78636481, 0.80252616,
       0.81868751, 0.83484886, 0.85101021, 0.86717157, 0.88333292,
       0.89949427, 0.91565562, 0.93181697, 0.94797832, 0.96413968,
       0.98030103, 0.99646238, 1.01262373, 1.02878508, 1.04494644,
       1.06110779, 1.07726914, 1.09343049, 1.10959184, 1.1257532 ,
       1.14191455, 1.1580759 , 1.17423725, 1.1903986 , 1.20655995,
       1.22272131, 1.23888266, 1.25504401, 1.27120536, 1.28736671,
       1.30352807, 1.31968942, 1.33585077, 1.35201212, 1.36817347,
       1.38433482, 1.40049618, 1.41665753, 1.43281888, 1.44898023,
       1.46514158, 1.48130294, 1.49746429, 1.51362564, 1.52978699,
       1.54594834, 1.56210969, 1.57827105, 1.5944324 , 1.61059375,
       1.6267551 , 1.64291645, 1.65907781, 1.67523916, 1.69140051,
       1.70756186, 1.72372321, 1.73988457, 1.75604592, 1.77220727,
       1.78836862, 1.80452997, 1.82069132, 1.83685268, 1.85301403]])
```

[13]:
```python
# from sklearn.linear_model import LinearRegression
# from sklearn.preprocessing import PolynomialFeatures

# poly_1 = PolynomialFeatures(degree=1)
# poly_3 = PolynomialFeatures(degree=3)
# poly_6 = PolynomialFeatures(degree=6)
# poly_9 = PolynomialFeatures(degree=9)
```

[14]:
```python
# X_train
```

[15]:
```python
# X_train = X_train.reshape(-1,1)

# X_train
```

[16]:
```python
# X_poly_1 = poly_1.fit_transform(X_train)
# X_poly_3 = poly_3.fit_transform(X_train)
# X_poly_6 = poly_6.fit_transform(X_train)
# X_poly_9 = poly_9.fit_transform(X_train)
```

[17]:
```python
# model_1 = LinearRegression()
# model_1.fit(X_poly_1, y_train)

# model_3 = LinearRegression()
# model_3.fit(X_poly_3, y_train)

# model_6 = LinearRegression()
# model_6.fit(X_poly_6, y_train)

# model_9 = LinearRegression()
# model_9.fit(X_poly_9, y_train)
```

[18]:
```
# test_numbers = np.linspace(1,10,100)
# test_numbers = test_numbers.reshape(-1,1)

# predictions_1 = model_1.predict(poly_1.fit_transform(test_numbers))
# predictions_3 = model_3.predict(poly_3.fit_transform(test_numbers))
# predictions_6 = model_6.predict(poly_6.fit_transform(test_numbers))
# predictions_9 = model_9.predict(poly_9.fit_transform(test_numbers))
```

[19]:
```
# predictions_1 = predictions_1.reshape(1, -1)
# predictions_3 = predictions_3.reshape(1, -1)
# predictions_6 = predictions_6.reshape(1, -1)
# predictions_9 = predictions_9.reshape(1, -1)
```

[20]:
```
# result = np.concatenate((predictions_1, predictions_3, predictions_6,␣
↪predictions_9), axis = 0)
```

1.1.2 Question 2

Write a function that fits a polynomial LinearRegression model on the training data `X_train` for degrees 0 through 9. For each model compute the R^2 (coefficient of determination) regression score on the training data as well as the the test data, and return both of these arrays in a tuple.

This function should return a tuple of numpy arrays `(r2_train, r2_test)`*. Both arrays should have shape* `(10,)`

[37]:
```
def answer_two():
    import numpy as np
    import pandas as pd
    import matplotlib.pyplot as plt
    from sklearn.model_selection import train_test_split

    from sklearn.linear_model import LinearRegression
    from sklearn.preprocessing import PolynomialFeatures
    from sklearn.metrics import r2_score

    # r2_train = np.array([])
    # r2_test = np.array([])

    # YOUR CODE HERE

    # from sklearn.linear_model import LinearRegression
    # from sklearn.preprocessing import PolynomialFeatures
    # from sklearn.metrics import r2_score

    np.random.seed(0)
    n = 15
    x = np.linspace(0,10,n) + np.random.randn(n)/5
    y = np.sin(x)+x/6 + np.random.randn(n)/10
```

```python
    X_train, X_test, y_train, y_test = train_test_split(x, y, random_state=0)

    degrees = list(range(0,10))

    X_train = X_train.reshape(-1, 1)
    X_test = X_test.reshape(-1, 1)

    result_train = np.zeros([len(degrees), len(X_train)])
    result_test = np.zeros([len(degrees), len(X_test)])

    r2_train_array = np.empty(len(degrees))
    r2_test_array = np.empty(len(degrees))

    for i in degrees:
        poly = PolynomialFeatures(degree = i)
        X_transformed = poly.fit_transform(X_train)

        reg = LinearRegression()
        reg.fit(X_transformed, y_train)

        X_train_poly = poly.fit_transform(X_train)
        X_test_poly = poly.fit_transform(X_test)

        result_train[i, :] = reg.predict(X_train_poly)
        result_test[i, :] = reg.predict(X_test_poly)

        r2_train_array[i] = r2_score(y_train, result_train[i])
        r2_test_array[i] = r2_score(y_test, result_test[i])

    return (r2_train_array, r2_test_array)

    raise NotImplementedError()
```

[ ]:

```python
```

[38]:

```python
answer_two()
```

[38]:

```
(array([0.        , 0.42924578, 0.4510998 , 0.58719954, 0.91941945,
        0.97578641, 0.99018233, 0.99352509, 0.99637545, 0.99803706]),
 array([-0.47808642, -0.45237104, -0.06856984,  0.00533105,  0.73004943,
         0.87708301,  0.9214094 ,  0.92021504,  0.63247942, -0.64525285]))
```

[22]:

```python
# def answer_one_test():
#     from sklearn.linear_model import LinearRegression
#     from sklearn.preprocessing import PolynomialFeatures
```

```
#     DEGREES = [1, 3, 6, 9]
#     N_POINTS = 100
#     result = np.zeros([len(DEGREES), N_POINTS])
#     predict = np.linspace(0, 10, N_POINTS).reshape(-1, 1)
#     X_tr = X_train.reshape(-1, 1)

#     for i, deg in enumerate(DEGREES):

#         poly = PolynomialFeatures(degree=deg)
#         X_ = poly.fit_transform(X_tr)
#         predict_ = poly.fit_transform(predict)
#         reg = LinearRegression()
#         reg.fit(X_, y_train)
#         result[i, :] = reg.predict(predict_)

#     return result
```

[34]:
```
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import r2_score

degrees = list(range(0,10))

X_train = X_train.reshape(-1, 1)
X_test = X_test.reshape(-1, 1)

result_train = np.zeros([len(degrees), len(X_train)])
result_test = np.zeros([len(degrees), len(X_test)])

r2_train_array = np.empty(len(degrees))
r2_test_array = np.empty(len(degrees))

for i in degrees:
    poly = PolynomialFeatures(degree = i)
    X_transformed = poly.fit_transform(X_train)

    reg = LinearRegression()
    reg.fit(X_transformed, y_train)

    X_train_poly = poly.fit_transform(X_train)
    X_test_poly = poly.fit_transform(X_test)

    result_train[i, :] = reg.predict(X_train_poly)
    result_test[i, :] = reg.predict(X_test_poly)

    r2_train_array[i] = r2_score(y_train, result_train[i])
    r2_test_array[i] = r2_score(y_test, result_test[i])
```

```
[35]: r2_train_array
```

```
[35]: array([0.        , 0.42924578, 0.4510998 , 0.58719954, 0.91941945,
       0.97578641, 0.99018233, 0.99352509, 0.99637545, 0.99803706])
```

```
[36]: r2_test_array
```

```
[36]: array([-0.47808642, -0.45237104, -0.06856984,  0.00533105,  0.73004943,
        0.87708301,  0.9214094 ,  0.92021504,  0.63247942, -0.64525285])
```

1.1.3 Question 3

Based on the R^2 scores from question 2 (degree levels 0 through 9), what degree level corresponds to a model that is underfitting? What degree level corresponds to a model that is overfitting? What choice of degree level would provide a model with good generalization performance on this dataset?

(Hint: Try plotting the R^2 scores from question 2 to visualize the relationship)

This function should return a tuple with the degree values in this order: `(Underfitting, Overfitting, Good_Generalization)`

```
[39]: def answer_three():
          # YOUR CODE HERE

          return (4, 9, 6)

          raise NotImplementedError()
```

```
[ ]:
```

1.1.4 Question 4

Training models on high degree polynomial features can result in overfitting. Train two models: a non-regularized LinearRegression model and a Lasso Regression model (with parameters `alpha=0.01`, `max_iter=10000`, `tol=0.1`) on polynomial features of degree 12. Return the R^2 score for LinearRegression and Lasso model's test sets.

This function should return a tuple `(LinearRegression_R2_test_score, Lasso_R2_test_score)`

```
[63]: def answer_four():

      #     from sklearn.preprocessing import PolynomialFeatures
      #     from sklearn.linear_model import Lasso, LinearRegression
      #     from sklearn.metrics import r2_score

          import numpy as np
```

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Lasso, LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import r2_score

np.random.seed(0)
n = 15
x = np.linspace(0,10,n) + np.random.randn(n)/5
y = np.sin(x)+x/6 + np.random.randn(n)/10

X_train, X_test, y_train, y_test = train_test_split(x, y, random_state=0)

X_train = X_train.reshape(-1, 1)
X_test = X_test.reshape(-1, 1)

degree = 12
poly = PolynomialFeatures(degree=degree)
X_train_poly = poly.fit_transform(X_train)
X_test_poly = poly.transform(X_test)

#Linear Regression
reg = LinearRegression()
reg.fit(X_train_poly, y_train)
linear_r2_score = r2_score(y_test, reg.predict(X_test_poly))

#Lasso Regression
alpha=0.01
max_iter=10000
tol = 0.1

linlasso = Lasso(alpha=alpha, max_iter = max_iter, tol = tol).
↪fit(X_train_poly, y_train)
lasso_r2_score = r2_score(y_test, linlasso.predict(X_test_poly))

return (linear_r2_score, lasso_r2_score)

# poly = PolynomialFeatures(degree = 12)
# X_transformed = poly.fit_transform(X_train)
# reg = LinearRegression()
# reg.fit(X_transformed, y_train)

# X_train_poly = poly.fit_transform(X_train)
# X_test_poly = poly.fit_transform(X_test)

# result_train[i, :] = reg.predict(X_train_poly)
```

```python
    # result_test[i, :] = reg.predict(X_test_poly)

    # r2_train_array[i] = r2_score(y_train, result_train[i])
    # r2_test_array[i] = r2_score(y_test, result_test[i])



    # YOUR CODE HERE
    raise NotImplementedError()
```

[64]:
```python
answer_four()
```

[64]: (-4.3119675863308435, 0.6051396919570036)

[ ]:
```python

```

[ ]:
```python

```

[65]:
```python
#     from sklearn.preprocessing import PolynomialFeatures
#     from sklearn.linear_model import Lasso, LinearRegression
#     from sklearn.metrics import r2_score


import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Lasso, LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import r2_score

np.random.seed(0)
n = 15
x = np.linspace(0,10,n) + np.random.randn(n)/5
y = np.sin(x)+x/6 + np.random.randn(n)/10

X_train, X_test, y_train, y_test = train_test_split(x, y, random_state=0)

X_train = X_train.reshape(-1, 1)
X_test = X_test.reshape(-1, 1)

degree = 12
poly = PolynomialFeatures(degree=degree)
X_train_poly = poly.fit_transform(X_train)
X_test_poly = poly.transform(X_test)

#Linear Regression
```

```
reg = LinearRegression()
reg.fit(X_train_poly, y_train)
linear_r2_score = r2_score(y_test, reg.predict(X_test_poly))

#Lasso Regression
alpha=0.01
max_iter=10000
tol = 0.1

linlasso = Lasso(alpha=alpha, max_iter = max_iter, tol = tol).fit(X_train_poly,␣
 ↪y_train)
lasso_r2_score = r2_score(y_test, linlasso.predict(X_test_poly))

(linear_r2_score, lasso_r2_score)

# poly = PolynomialFeatures(degree = 12)
# X_transformed = poly.fit_transform(X_train)
# reg = LinearRegression()
# reg.fit(X_transformed, y_train)

# X_train_poly = poly.fit_transform(X_train)
# X_test_poly = poly.fit_transform(X_test)

# result_train[i, :] = reg.predict(X_train_poly)
# result_test[i, :] = reg.predict(X_test_poly)

# r2_train_array[i] = r2_score(y_train, result_train[i])
# r2_test_array[i] = r2_score(y_test, result_test[i])

# YOUR CODE HERE
```

[65]: (-4.3119675863308435, 0.6051396919570036)

1.2 Part 2 - Classification

For this section of the assignment we will be working with the UCI Mushroom Data Set stored in `mushrooms.csv`. The data will be used to trian a model to predict whether or not a mushroom is poisonous. The following attributes are provided:

Attribute Information:

1. cap-shape: bell=b, conical=c, convex=x, flat=f, knobbed=k, sunken=s
2. cap-surface: fibrous=f, grooves=g, scaly=y, smooth=s
3. cap-color: brown=n, buff=b, cinnamon=c, gray=g, green=r, pink=p, purple=u, red=e, white=w, yellow=y
4. bruises?: bruises=t, no=f
5. odor: almond=a, anise=l, creosote=c, fishy=y, foul=f, musty=m, none=n, pungent=p, spicy=s

6. gill-attachment: attached=a, descending=d, free=f, notched=n
7. gill-spacing: close=c, crowded=w, distant=d
8. gill-size: broad=b, narrow=n
9. gill-color: black=k, brown=n, buff=b, chocolate=h, gray=g, green=r, orange=o, pink=p, purple=u, red=e, white=w, yellow=y
10. stalk-shape: enlarging=e, tapering=t
11. stalk-root: bulbous=b, club=c, cup=u, equal=e, rhizomorphs=z, rooted=r, missing=?
12. stalk-surface-above-ring: fibrous=f, scaly=y, silky=k, smooth=s
13. stalk-surface-below-ring: fibrous=f, scaly=y, silky=k, smooth=s
14. stalk-color-above-ring: brown=n, buff=b, cinnamon=c, gray=g, orange=o, pink=p, red=e, white=w, yellow=y
15. stalk-color-below-ring: brown=n, buff=b, cinnamon=c, gray=g, orange=o, pink=p, red=e, white=w, yellow=y
16. veil-type: partial=p, universal=u
17. veil-color: brown=n, orange=o, white=w, yellow=y
18. ring-number: none=n, one=o, two=t
19. ring-type: cobwebby=c, evanescent=e, flaring=f, large=l, none=n, pendant=p, sheathing=s, zone=z
20. spore-print-color: black=k, brown=n, buff=b, chocolate=h, green=r, orange=o, purple=u, white=w, yellow=y
21. population: abundant=a, clustered=c, numerous=n, scattered=s, several=v, solitary=y
22. habitat: grasses=g, leaves=l, meadows=m, paths=p, urban=u, waste=w, woods=d

The data in the mushrooms dataset is currently encoded with strings. These values will need to be encoded to numeric to work with sklearn. We'll use pd.get_dummies to convert the categorical variables into indicator variables.

[1]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split


mush_df = pd.read_csv('assets/mushrooms.csv')
mush_df2 = pd.get_dummies(mush_df)

X_mush = mush_df2.iloc[:,2:]
y_mush = mush_df2.iloc[:,1]


X_train2, X_test2, y_train2, y_test2 = train_test_split(X_mush, y_mush,␣
 ↪random_state=0)
```

[3]:
```
X_train2
```

[3]:
```
      cap-shape_b  cap-shape_c  cap-shape_f  cap-shape_k  cap-shape_s  \
5832            0            0            1            0            0
601             0            0            0            0            0
1601            0            0            1            0            0
```

```
4941            0            0            0            0            0
7492            0            0            1            0            0
…           …          …          …          …          …
4931            0            0            0            0            0
3264            0            0            0            0            0
1653            0            0            0            0            0
2607            0            0            1            0            0
2732            0            0            0            0            0

      cap-shape_x  cap-surface_f  cap-surface_g  cap-surface_s  cap-surface_y  \
5832            0              0              0              0              1
601             1              0              0              0              1
1601            0              0              0              1              0
4941            1              1              0              0              0
7492            0              0              0              0              1
…           …            …            …            …            …
4931            1              0              0              0              1
3264            1              1              0              0              0
1653            1              0              0              1              0
2607            0              1              0              0              0
2732            1              0              0              0              1

      …  population_s  population_v  population_y  habitat_d  habitat_g  \
5832  …             0             0             1          0          1
601   …             0             0             1          0          1
1601  …             0             0             0          0          1
4941  …             0             1             0          0          0
7492  …             0             1             0          1          0
…  …           …           …           …         …        …
4931  …             0             0             0          0          0
3264  …             0             0             1          0          0
1653  …             1             0             0          0          1
2607  …             0             1             0          1          0
2732  …             0             0             1          1          0

      habitat_l  habitat_m  habitat_p  habitat_u  habitat_w
5832          0          0          0          0          0
601           0          0          0          0          0
1601          0          0          0          0          0
4941          0          0          1          0          0
7492          0          0          0          0          0
…         …        …        …        …        …
4931          0          0          0          0          1
3264          0          0          1          0          0
1653          0          0          0          0          0
2607          0          0          0          0          0
2732          0          0          0          0          0
```

```
[6093 rows x 117 columns]
```

[4]:
```
y_train2
```

[4]:
```
5832    1
601     0
1601    0
4941    1
7492    1
       ..
4931    0
3264    1
1653    0
2607    0
2732    0
Name: class_p, Length: 6093, dtype: uint8
```

1.2.1 Question 5

Using `X_train` and `y_train` from the preceeding cell, train a DecisionTreeClassifier with default parameters and random_state=0. What are the 5 most important features found by the decision tree?

This function should return a list of length 5 of the feature names in descending order of importance.

[29]:
```
def answer_five():
    from sklearn.tree import DecisionTreeClassifier

    # YOUR CODE HERE

    classifier = DecisionTreeClassifier(random_state = 0)

    classifier.fit(X_train2, y_train2)

    feature_importances = pd.DataFrame(classifier.feature_importances_,
                                       index = X_train2.columns)

    feature_importances.rename(columns={feature_importances.columns[0]:␣
 ↪"importance" }, inplace = True)

    feature_importances = feature_importances.sort_values(by = "importance",␣
 ↪ascending = False)

    return list(feature_importances.index[0:5])

    raise NotImplementedError()
```

[ ]:

```

```

[30]:

```python
list(answer_five())
```

[30]:

```
['odor_n', 'stalk-root_c', 'stalk-root_r', 'spore-print-color_r', 'odor_l']
```

[6]:

```python
from sklearn.tree import DecisionTreeClassifier

classifier = DecisionTreeClassifier(random_state = 0)

classifier.fit(X_train2, y_train2)
```

[6]:

```
DecisionTreeClassifier(random_state=0)
```

[10]:

```python
feature_importances = pd.DataFrame(classifier.feature_importances_,
                                   index = X_train2.columns)
```

[17]:

```python
feature_importances.rename(columns={feature_importances.columns[0]:␣
 ↪"importance" }, inplace = True)
```

[20]:

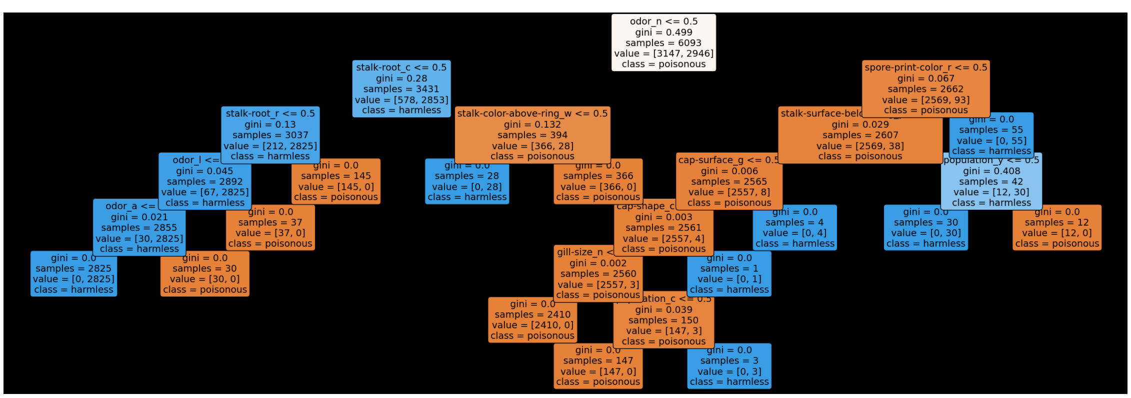
```python
feature_importances = feature_importances.sort_values(by = "importance",␣
 ↪ascending = False)
```

[31]:

```python
answer = list(feature_importances.index[0:5])
```

[46]:

```python
from sklearn import tree
import matplotlib.pyplot as plt

plt.figure(figsize=(30,10), facecolor ='k')

label = ['poisonous', 'harmless']

a = tree.plot_tree(classifier, feature_names = X_train2.columns, class_names =␣
 ↪label, rounded = True, filled = True, fontsize=14)

plt.show()
```



1.2.2 Question 6

For this question, use the `validation_curve` function in `sklearn.model_selection` to determine training and test scores for a Support Vector Classifier (`SVC`) with varying parameter values.

Create an `SVC` with default parameters (i.e. `kernel='rbf', C=1`) and `random_state=0`. Recall that the kernel width of the RBF kernel is controlled using the `gamma` parameter. Explore the effect of `gamma` on classifier accuracy by using the `validation_curve` function to find the training and test scores for 6 values of `gamma` from `0.0001` to `10` (i.e. `np.logspace(-4,1,6)`).

For each level of `gamma`, `validation_curve` will use 3-fold cross validation (use `cv=3, n_jobs=2` as parameters for `validation_curve`), returning two 6x3 (6 levels of gamma x 3 fits per level) arrays of the scores for the training and test sets in each fold.

Find the mean score across the five models for each level of `gamma` for both arrays, creating two arrays of length 6, and return a tuple with the two arrays.

e.g.

if one of your array of scores is

```
array([[ 0.5,  0.4,  0.6],
       [ 0.7,  0.8,  0.7],
       [ 0.9,  0.8,  0.8],
       [ 0.8,  0.7,  0.8],
       [ 0.7,  0.6,  0.6],
       [ 0.4,  0.6,  0.5]])
```

it should then become

```
array([ 0.5,  0.73333333,  0.83333333,  0.76666667,  0.63333333, 0.5])
```

This function should return a tuple of numpy arrays `(training_scores, test_scores)` where each array in the tuple has shape `(6,)`.

[71]:
```python
def answer_six():
    from sklearn.svm import SVC
    from sklearn.model_selection import validation_curve
    # YOUR CODE HERE

    from sklearn.svm import SVC
    from sklearn.model_selection import validation_curve

    svc = SVC(kernel='rbf', C = 1, random_state = 0)
    # svc.fit(X_mush, y_mush)
    # don't need to fit the SVC in this case

    gamma_range = np.logspace(-4,1,6)
```

```
    train_score, test_score = validation_curve(svc, X_mush, y_mush, scoring =␣
↪'accuracy', param_name = 'gamma', param_range = gamma_range, cv = 3, n_jobs␣
↪= 2)

    train_score_mean = np.mean(train_score, axis = 1)
    test_score_mean = np.mean(test_score, axis = 1)

    return (train_score_mean, test_score_mean)

    raise NotImplementedError()
```

[ ]:

[74]:
```
answer_six()
```

[74]:
```
(array([0.89838749, 0.98104382, 0.99895372, 1.        , 1.        ,
        1.        ]),
 array([0.88749385, 0.82951748, 0.84170359, 0.86582964, 0.83616445,
        0.51797144]))
```

[48]:
```
from sklearn.svm import SVC
from sklearn.model_selection import validation_curve

svc = SVC(kernel='rbf', C = 1, random_state = 0)
svc.fit(X_train2, y_train2)
```

[48]:
```
SVC(C=1, random_state=0)
```

[49]:
```
gamma_range = np.logspace(-4,1,6)
```

[50]:
```
gamma_range
```

[50]:
```
array([1.e-04, 1.e-03, 1.e-02, 1.e-01, 1.e+00, 1.e+01])
```

[73]:
```
train_score, test_score = validation_curve(svc, X_mush, y_mush, scoring =␣
↪'accuracy', param_name = 'gamma', param_range = gamma_range, cv = 3, n_jobs␣
↪= 2)
```

[75]:
```
train_score
```

[75]:
```
array([[0.93149926, 0.91174298, 0.85192024],
       [0.97913589, 0.97895126, 0.98504431],
       [0.99796898, 0.99889217, 1.        ],
       [1.        , 1.        , 1.        ],
       [1.        , 1.        , 1.        ],
       [1.        , 1.        , 1.        ]])
```

[76]:
```
test_score
```

[76]:
```
array([[0.82717873, 0.87370753, 0.96159527],
       [0.83493353, 0.87370753, 0.77991137],
       [0.8478582 , 0.96085672, 0.71639586],
       [0.90546529, 0.99002954, 0.70199409],
       [0.86779911, 0.94202363, 0.69867061],
       [0.51772526, 0.51809453, 0.51809453]])
```

[77]:
```
train_score_mean = np.mean(train_score, axis = 1)
test_score_mean = np.mean(test_score, axis = 1)
```

1.2.3 Question 7

Based on the scores from question 6, what gamma value corresponds to a model that is underfitting? What gamma value corresponds to a model that is overfitting? What choice of gamma would provide a model with good generalization performance on this dataset?

(Hint: Try plotting the scores from question 6 to visualize the relationship)

This function should return a tuple with the degree values in this order: ***(Underfitting, Overfitting, Good_Generalization)***

[79]:
```
def answer_seven():
    # YOUR CODE HERE

    return (1.e-01, 1.e-02 , 1.e-04)

    raise NotImplementedError()
```

[ ]:
```
```

[80]:
```
answer_seven()
```

[80]:
```
(0.1, 0.01, 0.0001)
```

[ ]:
```
```