ML Challenge (Optional)

Train, test, optimize, and analyze the performance of a classification model using a methodology of your choice for the randomly generated moons dataset. You are not being evaluated for the performance of your model. Instead, we are interested in whether you can implement a simple but rigorous ML workflow. Show all of your work in this notebook.

```
[ ]  #you are free to use any package you deem fit
```

Dataset

```
[129] # DO NOT MODIFY
      from sklearn.datasets import make_moons
      X, Y = make_moons(random_state=42, n_samples=(50, 450), noise=0.25)
```

```
[130] X
```

```
array([[ 1.06510461e+00, -6.96283210e-01],
       [ 5.37056378e-01, -2.60130153e-01],
       [ 1.27460054e+00, -2.42533052e-01],
       [ 1.91891162e+00,  6.88085727e-01],
       [ 4.12700482e-01, -3.88179900e-01],
       [ 1.49593625e+00, -8.41318795e-01],
       [ 2.28407156e+00,  8.74540085e-01],
       [ 1.52506263e+00, -9.91162812e-01],
       [ 1.04727023e+00,  8.57224389e-02],
       [-7.53662868e-02,  4.63340361e-01],
       [ 1.28328060e+00, -2.36310050e-01],
       [ 3.17294296e-01,  6.74434282e-01],
       [ 2.74057219e+00,  2.11910088e-01],
       [-5.43187742e-02, -2.20043740e-01],
       [ 1.32535413e+00, -2.18106658e-03],
       [ 8.42913109e-01, -2.45049195e-01],
       [ 2.04463391e+00,  9.34206039e-01],
       [ 3.57659664e-01, -2.91583150e-03],
       [ 3.59391860e-01, -1.44689860e-01],
       [-3.02760027e-01,  3.23542602e-01],
       [-1.34732495e+00,  4.02589504e-02],
       [ 1.56494286e+00, -6.61585995e-01],
       [ 1.23942357e+00, -3.92524081e-01],
       [ 8.57668905e-01, -2.95219115e-01],
       [ 1.52889137e-02,  1.01579728e+00],
       [ 1.08691664e+00, -7.42662866e-02],
       [ 1.69643024e+00,  2.39556212e-01],
       [ 3.64178303e-01, -9.71812369e-02],
       [ 8.65186312e-01, -7.23822275e-01],
       [ 1.78251508e+00,  2.50206105e-01],
       [ 4.12474392e-01, -1.35080265e-01],
       [ 1.81764674e+00, -7.84939912e-01],
       [ 1.19095529e+00,  7.09796185e-02],
       [ 2.15567456e+00,  3.94334679e-02],
       [ 1.19629708e+00, -2.67936304e-01],
       [ 1.06279807e+00, -5.55949624e-01],
       [ 1.31144008e+00, -9.52394193e-02],
       [ 1.61502582e-02, -4.03184194e-01],
       [ 3.25786551e-01,  8.34119289e-02],
       [ 4.96690017e-01, -6.86657396e-03],
       [ 1.41544816e+00, -2.22204697e-01],
       [-2.27333241e-01,  1.01251114e-01],
       [ 1.75619762e+00, -3.66319581e-01],
       [ 6.50336774e-01, -2.29047989e-01],
       [ 1.82226120e+00, -1.73979131e-02],
       [ 1.20355242e+00, -5.86293740e-01],
       [ 2.57613954e-01, -2.23558565e-01],
       [ 1.83668778e+00,  3.12042947e-01],
       [ 1.21705879e-02, -1.49315110e-01],
       [ 2.41806981e+00,  7.30733046e-01],
       [-1.82522513e-01,  4.65133491e-01],
       [ 6.91701898e-01, -4.52335249e-01],
       [-4.01495165e-01,  6.15027745e-01],
       [ 1.18067135e-01,  1.81017518e-01],
       [ 1.17090412e+00, -8.26998650e-01],
       [ 1.08402387e+00,  2.45863467e-02],
       [ 1.54211042e+00, -2.38542406e-01],
       [-1.90147102e-01,  6.64123803e-01]])
```

```
[131] Y
```

```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0,
       1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1,
```

```
1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0,
1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1,
1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1])
```

[32]
```
#X = np.array(X)
#Y = np.array(Y)
```

[132]
```
len(X)
```
```
500
```

[133]
```
len(Y)
```
```
500
```

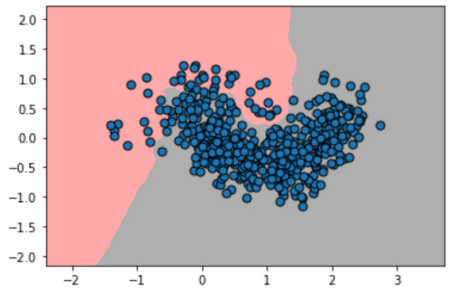
[134]
```
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, random_state=0)
```

[ ]

[135]
```
from sklearn.preprocessing import MinMaxScaler
from sklearn.svm import SVC
scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

clf = SVC(C=10).fit(X_train_scaled, Y_train)
print(clf.score(X_train_scaled, Y_train))
print(clf.score(X_test_scaled, Y_test))
```
```
0.9786666666666667
0.96
```

[136]
```
import matplotlib.cm as cm
from matplotlib.colors import ListedColormap, BoundaryNorm
import matplotlib.patches as mpatches
import matplotlib.patches as mpatches
from sklearn.neighbors import KNeighborsClassifier

X_train, X_test, y_train, y_test = train_test_split(X, Y, random_state=0)

n_neighbors = 5
weights = 'uniform'

def plot_moon_knn(X, Y, n_neighbors, weights):
    X_mat = X
    y_mat = Y
# Create color maps
    cmap_light = ListedColormap(['#FFAAAA', '#AAFFAA', '#AAAAFF','#AFAFAF'])
    cmap_bold  = ListedColormap(['#FF0000', '#00FF00', '#0000FF','#AFAFAF'])
    clf = KNeighborsClassifier(n_neighbors, weights=weights)
    clf.fit(X_mat, y_mat)
# Plot the decision boundary by assigning a color in the color map
    # to each mesh point.

    mesh_step_size = .01  # step size in the mesh
    plot_symbol_size = 50

    x_min, x_max = X_mat[:, 0].min() - 1, X_mat[:, 0].max() + 1
    y_min, y_max = X_mat[:, 1].min() - 1, X_mat[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, mesh_step_size),
                         np.arange(y_min, y_max, mesh_step_size))
    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
# Put the result into a color plot
    Z = Z.reshape(xx.shape)
    plt.figure()
    plt.pcolormesh(xx, yy, Z, cmap=cmap_light)
# Plot training points
    plt.scatter(X_mat[:, 0], X_mat[:, 1], s=plot_symbol_size, c=y, cmap=cmap_bold, edgecolor = 'black')
    plt.xlim(xx.min(), xx.max())
```

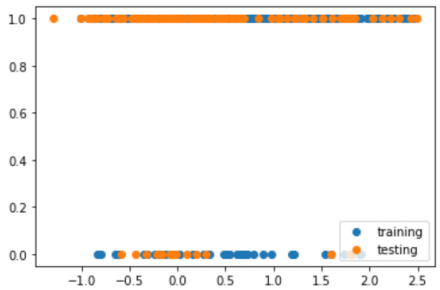
[83] plot_moon_knn(X, Y, n_neighbors, weights)



[137]
```
from sklearn.model_selection import cross_val_score

clf = KNeighborsClassifier(n_neighbors=5)

cv_scores = cross_val_score(clf, X, Y)

print('Cross-validation scores (3-fold):', cv_scores)
print('Mean cross-validation score (3-fold): {:.3f}'.format(np.mean(cv_scores)))
```

```
Cross-validation scores (3-fold): [0.95 0.96 0.99 0.95 0.96]
Mean cross-validation score (3-fold): 0.962
```

[139]
```
X = X[:500]
Y = Y[:500]
X = X.reshape(-1, 1)
Y = Y.reshape(-1, 1)

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, random_state = 0)

def data_scatter():
  plt.figure()
  plt.scatter(X_train, Y_train, label='training')
  plt.scatter(X_test, Y_test, label='testing')
  plt.legend(loc=4)

data_scatter()
```

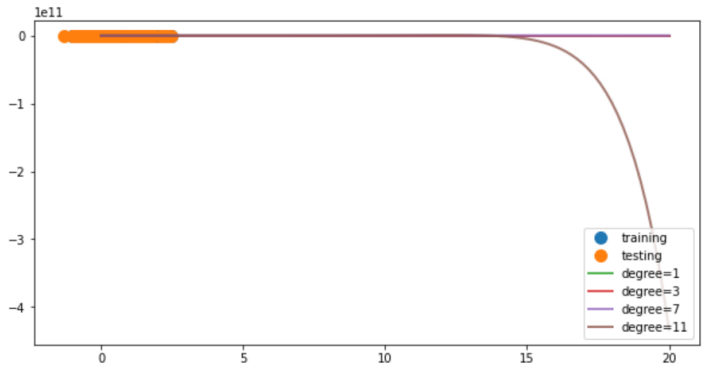


[140]
```
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures

degs = (1, 3, 7, 11)
def part_one():
  preds = np.arange(100).reshape((1,100))
  X_predict_input = np.linspace(0, 20, 100).reshape(-1, 1)

  for deg in degs:
    poly = PolynomialFeatures(degree=deg)
    X_poly = poly.fit_transform(X)
    X_train, X_test, Y_train, Y_test = train_test_split(X_poly, Y, random_state=0)
    reg = LinearRegression().fit(X_train, Y_train)
    deg_X_predict_input = poly.fit_transform(X_predict_input)
    pred = reg.predict(deg_X_predict_input)
    pred = pred.transpose()
    preds = np.vstack([preds, pred])

  preds = np.delete(preds, 0, 0)

  return preds

part_one()
```

```
       -2.82976307e+03, -3.76019043e+03, -4.93818427e+03,
       -6.41577718e+03, -8.25332099e+03, -1.05204380e+04,
       -1.32970417e+04, -1.66744289e+04, -2.07564464e+04,
       -2.56607338e+04, -3.15200458e+04, -3.84836556e+04,
       -4.67188423e+04, -5.64124641e+04, -6.77726210e+04,
       -8.10304081e+04, -9.64417619e+04, -1.14289404e+05.
```

```
       -1.34884881e+05, -1.58570709e+05, -1.85722614e+05,
       -2.16751889e+05, -2.52107846e+05, -2.92280391e+05,
       -3.37802703e+05, -3.89254032e+05, -4.47262614e+05,
       -5.12508706e+05, -5.85727746e+05, -6.67713631e+05,
       -7.59322135e+05, -8.61474439e+05, -9.75160811e+05,
       -1.10144441e+06, -1.24146523e+06, -1.39644419e+06,
       -1.56768734e+06, -1.75659025e+06, -1.96464252e+06,
       -2.19343245e+06, -2.44465185e+06, -2.72010101e+06,
       -3.02169383e+06, -3.35146313e+06, -3.71156605e+06,
       -4.10428969e+06, -4.53205689e+06, -4.99743219e+06,
       -5.50312790e+06, -6.05201043e+06, -6.64710675e+06,
       -7.29161101e+06, -7.98889140e+06, -8.74249715e+06,
       -9.55616569e+06, -1.04338301e+07, -1.13796266e+07,
       -1.23979023e+07, -1.34932234e+07, -1.46703829e+07,
       -1.59344095e+07, -1.72905756e+07, -1.87444065e+07,
       -2.03016893e+07, -2.19684817e+07, -2.37511220e+07,
       -2.56562383e+07, -2.76907582e+07, -2.98619194e+07,
       -3.21772794e+07],
      [ 9.54324347e-01,  9.56504203e-01,  8.92384489e-01,
        8.21392717e-01,  8.19777520e-01,  8.92058096e-01,
        9.60993306e-01,  9.46800174e-01,  8.64350539e-01,
        8.31261814e-01,  9.29568389e-01,  1.02957758e+00,
        9.88309853e-01,  2.08949724e+00,  1.12015350e+01,
        4.98892852e+01,  1.71611624e+02,  4.91146674e+02,
        1.23147953e+03,  2.79452283e+03,  5.86316989e+03,
        1.15432456e+04,  2.15548549e+04,  3.84833576e+04,
        6.61006378e+04,  1.09767389e+05,  1.76926711e+05,
        2.77698282e+05,  4.25580632e+05,  6.38266467e+05,
        9.38572403e+05,  1.35547980e+06,  1.92527741e+06,
        2.69278907e+06,  3.71266081e+06,  5.05067059e+06,
        6.78501139e+06,  9.00748320e+06,  1.18245119e+07,
        1.53578933e+07,  1.97451376e+07,  2.51392634e+07,
        3.17078612e+07,  3.96312142e+07,  4.90992262e+07,
        6.03068665e+07,  7.34477975e+07,  8.87057981e+07,
        1.06243544e+08,  1.26188246e+08,  1.48613578e+08,
        1.73517260e+08,  2.00793579e+08,  2.30200057e+08,
        2.61317355e+08,  2.93501454e+08,  3.25826991e+08,
        3.57020555e+08,  3.85382606e+08,  4.08696560e+08,
        4.24123426e+08,  4.28080248e+08,  4.16100435e+08,
        3.82673898e+08,  3.21064716e+08,  2.23103888e+08,
        7.89544893e+07, -1.23153639e+08, -3.97222781e+08,
       -7.59823172e+08, -1.23046938e+09, -1.83204109e+09,
       -2.59125254e+09, -3.53917494e+09, -4.71181685e+09,
       -6.15076741e+09, -7.90390811e+09, -1.00261987e+10,
       -1.25805440e+10, -1.56387469e+10, -1.92825567e+10,
       -2.36048185e+10, -2.87107318e+10, -3.47192281e+10,
       -4.17644746e+10, -4.99975150e+10, -5.95880564e+10,
       -7.07264130e+10, -8.36256191e+10, -9.85237210e+10,
       -1.15686262e+11, -1.35408973e+11, -1.58020682e+11,
       -1.83886459e+11, -2.13411011e+11, -2.47042336e+11,
       -2.85275669e+11, -3.28657722e+11, -3.77791245e+11,
       -4.33339921e+11]])
```

[141]
```
preds = part_one()
def plot_one(preds):
  plt.figure(figsize=(10,5))
  plt.plot(X_train, Y_train, 'o', label='training', markersize=10)
  plt.plot(X_test, Y_test, 'o', label='testing', markersize=10)
  for i, deg in enumerate(degs):
    plt.plot(np.linspace(0, 20, 100), preds[i], alpha=0.8, lw=2, label=f"degree={deg}")
  plt.legend(loc=4)

plot_one(preds)
```



[142]
```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score

def part_two():
  r2_train, r2_test = [], []

  for i in ([1,3,7,11]):
    poly = PolynomialFeatures()
    X_train_poly = poly.fit_transform(X_train.reshape(-1,1))
    X_test_poly = poly.fit_transform(X_test.reshape(-1,1))
```

```
    linreg = LinearRegression().fit(X_train_poly, Y_train)
    r2_train.append(linreg.score(X_train_poly, Y_train))
    r2_test.append(linreg.score(X_test_poly, Y_test))

  return r2_train, r2_test

part_two()
```

```
([0.0013400554267293252,
  0.0013400554267293252,
  0.0013400554267293252,
  0.0013400554267293252],
 [-0.010821556755350903,
  -0.010821556755350903,
  -0.010821556755350903,
  -0.010821556755350903])
```

```
[143] from sklearn.neighbors import KNeighborsRegressor

def part_three():
  r2_train_test = part_two()
  X_train, X_test, Y_train, Y_test = train_test_split(X, Y, random_state=0)
  knnreg = KNeighborsRegressor(n_neighbors=5).fit(X_train, Y_train)
  r2 = knnreg.score(X_test, Y_test)

  return r2

part_three()
```

```
-0.2030303030303029
```

Use Lasso Regression for Polynomial Fitting

```
[144] from sklearn.linear_model import Lasso
def part_four():
  preds = []
  lasso_reg = []
  pred_vec = []
  data = np.linspace(0, 20, 100).reshape(-1,1)

  X_train, X_test, Y_train, Y_test = train_test_split(X, Y, random_state=0)

  for d in degs:
    poly = PolynomialFeatures(degree=d)
    X1_F_poly = poly.fit_transform(X)
    trans_data = poly.fit_transform(data)
    X_train, X_test, Y_train, Y_test = train_test_split(X1_F_poly, Y, random_state=0)
    lasso_reg = Lasso(alpha=0.01, max_iter=10000).fit(X_train, Y_train)
    prediction = lasso_reg.predict(trans_data)
    pred_vec.append(prediction)
    preds = np.array(pred_vec)

  preds = preds.reshape(4,100)

  return preds

part_four()
```

```
        1.52025643e+02,  1.91648754e+02,  2.39814329e+02,
        2.97999060e+02,  3.67876634e+02,  4.51336782e+02,
        5.50505525e+02,  6.67766644e+02,  8.05784412e+02,
        9.67527625e+02,  1.15629497e+03,  1.37574175e+03,
        1.62990803e+03,  1.92324826e+03,  2.26066225e+03,
        2.64752785e+03,  3.08973500e+03,  3.59372148e+03,
        4.16651027e+03,  4.81574848e+03,  5.54974812e+03,
        6.37752847e+03,  7.30886029e+03,  8.35431182e+03,
        9.52529660e+03,  1.08341232e+04,  1.22940467e+04,
        1.39193227e+04,  1.57252622e+04,  1.77282896e+04,
        1.99460024e+04,  2.23972328e+04,  2.51021111e+04,
        2.80821327e+04,  3.13602252e+04,  3.49608198e+04,
        3.89099233e+04,  4.32351936e+04,  4.79660168e+04,
        5.31335871e+04,  5.87709887e+04,  6.49132810e+04,
        7.15975849e+04,  7.88631733e+04,  8.67515625e+04,
        9.53066075e+04,  1.04574599e+05,  1.14604365e+05,
        1.25447370e+05,  1.37157825e+05,  1.49792792e+05,
        1.63412299e+05,  1.78079450e+05,  1.93860544e+05,
        2.10825196e+05,  2.29046455e+05,  2.48600935e+05,
        2.69568942e+05,  2.92034605e+05,  3.16086011e+05,
        3.41815344e+05,  3.69319025e+05,  3.98697858e+05,
        4.30057177e+05,  4.63506997e+05,  4.99162166e+05,
        5.37142527e+05,  5.77573077e+05,  6.20584130e+05,
        6.66311486e+05],
       [ 9.07849097e-01,  9.07848817e-01,  9.07831332e-01,
         9.07652621e-01,  9.06809240e-01,  9.04273308e-01,
         8.98811946e-01,  8.90306540e-01,  8.82252266e-01,
         8.84395374e-01,  9.11763237e-01,  9.71432077e-01,
         1.02039505e+00,  8.65838215e-01, -3.81337871e-02,
        -2.96703024e+00, -1.05771175e+01, -2.79736105e+01,
```

```
        -6.43905609e+01, -1.35742244e+02, -2.68385901e+02,
        -5.04532737e+02, -9.09860353e+02, -1.58401764e+03,
        -2.67487507e+03, -4.39756214e+03, -7.05955161e+03,
        -1.10933011e+04, -1.70982485e+04, -2.58942819e+04,
        -3.85891732e+04, -5.66628728e+04, -8.20720294e+04,
        -1.17378607e+05, -1.65907048e+05, -2.31935050e+05,
        -3.20923744e+05, -4.39793797e+05, -5.97254817e+05,
        -8.04196357e+05, -1.07414981e+06, -1.42383155e+06,
        -1.87377894e+06, -2.44909190e+06, -3.18029448e+06,
        -4.10433183e+06, -5.26572018e+06, -6.71786860e+06,
        -8.52459359e+06, -1.07618492e+07, -1.35196977e+07,
        -1.69045481e+07, -2.10416919e+07, -2.60781686e+07,
        -3.21859954e+07, -3.95657994e+07, -4.84508929e+07,
        -5.91118359e+07, -7.18615333e+07, -8.70609184e+07,
        -1.05125278e+08, -1.26531274e+08, -1.51824739e+08,
        -1.81629285e+08, -2.16655839e+08, -2.57713139e+08,
        -3.05719311e+08, -3.61714589e+08, -4.26875287e+08,
        -5.02529115e+08, -5.90171952e+08, -6.91486188e+08,
        -8.08360742e+08, -9.42912910e+08, -1.09751215e+09,
        -1.27480597e+09, -1.47774805e+09, -1.70962875e+09,
        -1.97410827e+09, -2.27525241e+09, -2.61757143e+09,
        -3.00606195e+09, -3.44625220e+09, -3.94425087e+09,
        -4.50679969e+09, -5.14133014e+09, -5.85602435e+09,
        -6.65988063e+09, -7.56278380e+09, -8.57558065e+09,
        -9.71016087e+09, -1.09795437e+10, -1.23979708e+10,
        -1.39810052e+10, -1.57456377e+10, -1.77104001e+10,
        -1.98954857e+10, -2.23228785e+10, -2.50164908e+10,
        -2.80023090e+10]])
```

[145]
```
def part_five():
  best_deg = []

  x_p5 = np.linspace(0, 20, 100)
  y_p5 = (x_p5**3/20-x_p5**2-x_p5)

  x_p5 = x_p5.reshape(-1,1)
  y_p5 = y_p5.reshape(-1,1)
  X_train_p5, X_test_p5, Y_train_p5, Y_test_p5 = train_test_split(x_p5, y_p5, random_state=0)

  r2_score = []

  for deg in degs:
    poly = PolynomialFeatures(degree=deg)
    X_poly_p5 = poly.fit_transform(x_p5)
    X_train_p5, X_test_p5, Y_train_p5, Y_test_p5 = train_test_split(X_poly_p5, y_p5, random_state=0)
    reg = Lasso(alpha=0.01, max_iter=10000).fit(X_train_p5, Y_train_p5)
    r2_score.append(reg.score(X_test_p5, Y_test_p5))

  print(r2_score)

  best_score = 0

  for i in range(4):
    if r2_score[i] > best_score:
      best_score = r2_score[i]
      best_deg = degs[i]

  return best_deg

part_five()
```

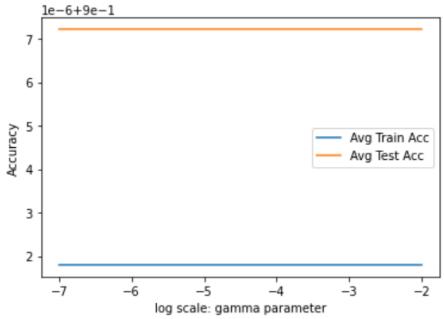
```
[0.5746265922370499, 0.9999998018443087, 0.9999251884788674, 0.9999206323336404]
3
```

Applying a SVC Classifier to the dataset

[146]
```
from sklearn.svm import SVC
from sklearn.model_selection import validation_curve

data = np.hstack([X, Y.reshape(-1,1)])
col_names = np.hstack(['X', 'Y'])
df = pd.DataFrame(data, columns=col_names)

X, Y = df.iloc[:, :-1], df.iloc[:, -1]

def part_six():
  parameter_range = np.logspace(-7, -2, 6, endpoint=True)
  train_scores, test_scores = validation_curve(SVC(), X, Y, param_name='gamma', param_range=parameter_range, cv=3)
  train_score_mean = np.mean(train_scores, axis=1)
  test_score_mean = np.mean(test_scores, axis=1)

  results = (train_score_mean, test_score_mean)

  return results

part_six()
```

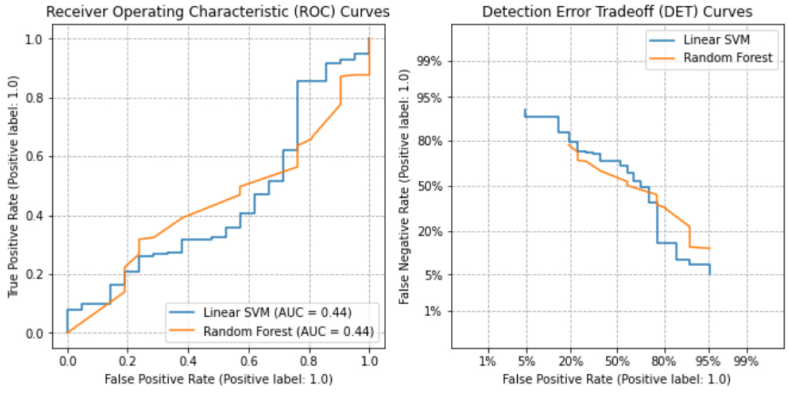
```
(array([0.9000018, 0.9000018, 0.9000018, 0.9000018, 0.9000018, 0.9000018]),
 array([0.90000721, 0.90000721, 0.90000721, 0.90000721, 0.90000721,
        0.90000721]))
```

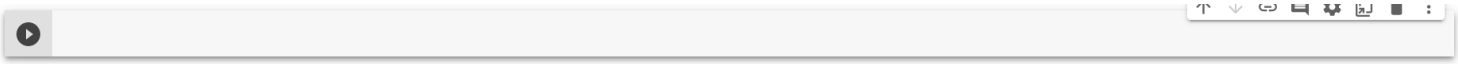
[147]
```
a, b = part_six()

x = np.arange(-7, -1, 1)
plt.figure()
plt.plot(x, a, label='Avg Train Acc')
plt.plot(x, b, label='Avg Test Acc')
plt.xticks(x)
plt.xlabel('log scale: gamma parameter')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```



Training

[148]
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import DetCurveDisplay, RocCurveDisplay
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.svm import LinearSVC

classifiers = {
    "Linear SVM": make_pipeline(StandardScaler(), LinearSVC(C=0.025)),
    "Random Forest": RandomForestClassifier(
        max_depth=5, n_estimators=10, max_features=1
    ),
}


X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.4, random_state=0)

#prepare plots
fig, [ax_roc, ax_det] = plt.subplots(1, 2, figsize=(11,5))

for name, clf in classifiers.items():
  clf.fit(X_train, Y_train)

  RocCurveDisplay.from_estimator(clf, X_test, Y_test, ax=ax_roc, name=name)
  DetCurveDisplay.from_estimator(clf, X_test, Y_test, ax=ax_det, name=name)

ax_roc.set_title("Receiver Operating Characteristic (ROC) Curves")
ax_det.set_title("Detection Error Tradeoff (DET) Curves")

ax_roc.grid(linestyle="--")
ax_det.grid(linestyle="--")

plt.legend()
plt.show()
```





Colab paid products - Cancel contracts here

✓ 0s completed at 10:15 PM