Hands-on Exercise CLASS Module

November 7, 2021

1 Hands-on Exercise CLASS Module

[1]:
```
!pip install --user mlxtend
```

```
Requirement already satisfied: mlxtend in ./.local/lib/python3.6/site-packages
Requirement already satisfied: matplotlib>=3.0.0 in ./.local/lib/python3.6/site-
packages (from mlxtend)
Requirement already satisfied: joblib>=0.13.2 in ./.local/lib/python3.6/site-
packages (from mlxtend)
Requirement already satisfied: scikit-learn>=0.20.3 in
./.local/lib/python3.6/site-packages (from mlxtend)
Requirement already satisfied: numpy>=1.16.2 in ./.local/lib/python3.6/site-
packages (from mlxtend)
Requirement already satisfied: scipy>=1.2.1 in ./.local/lib/python3.6/site-
packages (from mlxtend)
Requirement already satisfied: pandas>=0.24.2 in ./.local/lib/python3.6/site-
packages (from mlxtend)
Requirement already satisfied: setuptools in
/usr/local/anaconda5/lib/python3.6/site-packages (from mlxtend)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.3 in
/usr/local/anaconda5/lib/python3.6/site-packages (from
matplotlib>=3.0.0->mlxtend)
Requirement already satisfied: kiwisolver>=1.0.1 in ./.local/lib/python3.6/site-
packages (from matplotlib>=3.0.0->mlxtend)
Requirement already satisfied: cycler>=0.10 in
/usr/local/anaconda5/lib/python3.6/site-packages (from
matplotlib>=3.0.0->mlxtend)
Requirement already satisfied: python-dateutil>=2.1 in
./.local/lib/python3.6/site-packages (from matplotlib>=3.0.0->mlxtend)
Requirement already satisfied: pillow>=6.2.0 in ./.local/lib/python3.6/site-
packages (from matplotlib>=3.0.0->mlxtend)
Requirement already satisfied: threadpoolctl>=2.0.0 in
./.local/lib/python3.6/site-packages (from scikit-learn>=0.20.3->mlxtend)
Requirement already satisfied: pytz>=2017.2 in
/usr/local/anaconda5/lib/python3.6/site-packages (from pandas>=0.24.2->mlxtend)
Requirement already satisfied: six in /usr/local/anaconda5/lib/python3.6/site-
packages (from cycler>=0.10->matplotlib>=3.0.0->mlxtend)
```

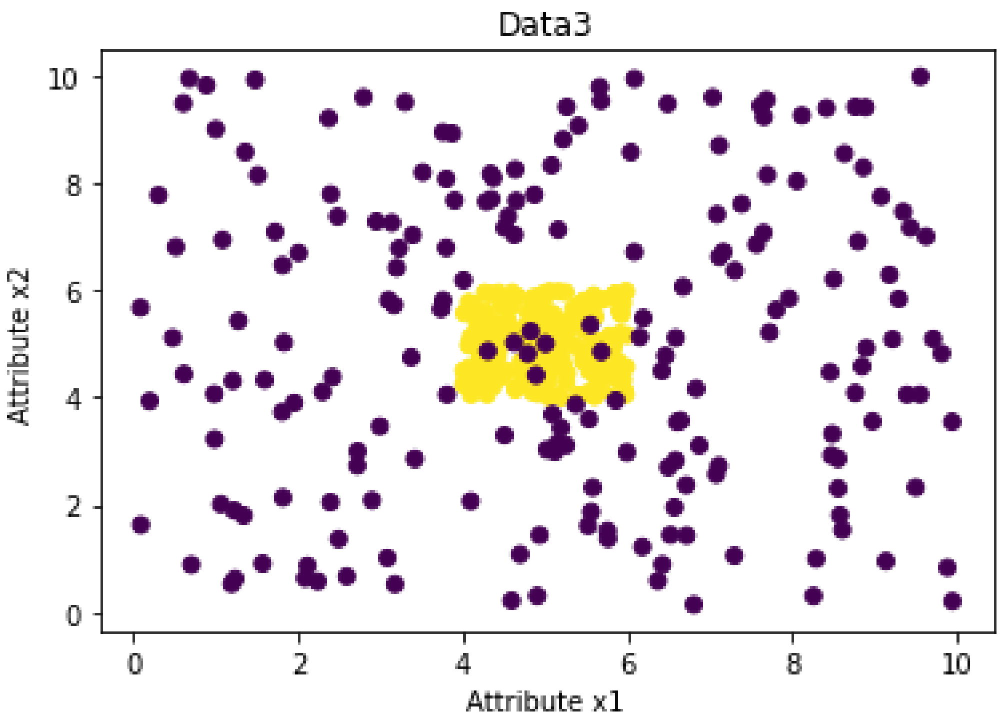
```
You are using pip version 9.0.1, however version 21.3.1 is available.
You should consider upgrading via the 'pip install --upgrade pip' command.
```

[2]:
```
import numpy as np

#Plotting packages
import matplotlib.pyplot as plt
#import seaborn as sns

#Classification Algorithms
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC

#Ensemble Methods
from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import BaggingRegressor
from sklearn.model_selection import cross_val_score, train_test_split
from sklearn.ensemble import AdaBoostClassifier

#Mlxtend for visualizing classification decision boundaries
from mlxtend.plotting import plot_decision_regions
```

[3]:
```
# Generating Data1

np.random.seed(100)

a = np.random.multivariate_normal([2,2],[[0.5,0], [0,0.5]], 200)
b = np.random.multivariate_normal([4,4],[[0.5,0], [0,0.5]], 200)

Data1_X = np.vstack((a,b))
Data1_Y = np.hstack((np.ones(200).T,np.zeros(200).T)).astype(int)



# Generating Data2

np.random.seed(100)

a1 = np.random.multivariate_normal([2,2],[[0.25,0], [0,0.25]],200)
a2 = np.random.multivariate_normal([2,4],[[0.25,0], [0,0.25]],200)
a3 = np.random.multivariate_normal([4,2],[[0.25,0], [0,0.25]],200)
a4 = np.random.multivariate_normal([4,4],[[0.25,0], [0,0.25]],200)

Data2_X = np.vstack((a1,a4,a2,a3))
Data2_Y = np.hstack((np.ones(400).T,np.zeros(400).T)).astype(int)
```

```
# Generating Data3

np.random.seed(100)

a1 = np.random.uniform(4,6,[200,2])
a2 = np.random.uniform(0,10,[200,2])

Data3_X = np.vstack((a1,a2))
Data3_Y = np.hstack((np.ones(200).T,np.zeros(200).T)).astype(int)


# Generating Data4

np.random.seed(100)

Data4_X = np.random.uniform(0,12,[500,2])
Data4_Y = np.ones([500]).astype(int)
Data4_Y[np.multiply(Data4_X[:,0],Data4_X[:,0]) + np.multiply(Data4_X[:
 ↪,1],Data4_X[:,1]) - 100 < 0 ] = 0
```

[4]:
```
import matplotlib.pyplot as plt
plt.scatter(Data3_X[:,0],Data3_X[:,1], c=Data3_Y)
plt.xlabel('Attribute x1')
plt.ylabel('Attribute x2')
plt.title('Data3')
plt.show()
```



[5]:
```
Data3_Y
```

[5]:
```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 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, 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, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0])
```

[6]:
```
Data3_X
```

```
[6]: array([[5.08680988, 4.55673877],
            [4.84903518, 5.68955226],
            [4.00943771, 4.24313824],
            [5.34149817, 5.65170551],
            [4.27341318, 5.15018666],
            [5.78264391, 4.41840424],
            [4.37065644, 4.21675378],
            [4.43939499, 5.95724757],
            [5.6233663 , 4.34388203],
            [5.6324495 , 4.54814749],
            [4.86340837, 5.88005964],
            [5.63529876, 4.6722239 ],
            [4.35082091, 4.74566409],
            [4.01137701, 4.50485271],
            [5.59132502, 4.03050994],
            [5.19768675, 5.20760908],
            [4.21029537, 4.76388689],
            [4.07295211, 5.78082313],
            [5.96184171, 4.11988398],
            [5.78109189, 5.153803  ],
            [5.48495938, 5.26036787],
            [5.16368438, 4.04087826],
            [4.42005316, 5.08936976],
            [5.53823034, 4.50139046],
            [4.57179138, 5.70479018],
            [5.95001299, 5.76970659],
            [4.71901569, 5.19771789],
            [4.70959122, 4.68038043],
            [4.35616198, 4.47538842],
            [4.08972456, 5.01086286],
            [4.75250491, 5.1856108 ],
            [5.25988375, 4.28520063],
            [5.8676826 , 5.89275976],
            [5.20459332, 4.77553256],
            [4.72637601, 4.40869055],
            [4.55353012, 4.49307176],
            [4.347216  , 5.93321939],
            [5.9140252 , 5.19594737],
            [5.46260151, 4.68077045],
            [4.18411121, 4.92699604],
            [5.01739779, 4.17692035],
            [5.05607045, 5.98431607],
            [4.79007186, 4.67119288],
            [5.61090107, 5.50869799],
            [4.62613288, 5.26807337],
            [5.08080915, 4.5935875 ],
            [4.2215758 , 4.6252806 ],
            [4.91395826, 5.31788014],
            [4.50851504, 5.28220252],
            [4.40024721, 5.31524961],
            [5.55657843, 5.5591968 ],
            [5.22065631, 4.6180007 ],
            [5.39546982, 5.71923659],
            [5.25064752, 5.96481566],
            [5.95300025, 4.33338826],
            [4.04635627, 4.3214891 ],
            [5.84699365, 5.9070997 ],
            [4.42195684, 4.7210505 ],
            [5.09875052, 4.5436617 ],
            [4.92120324, 5.39232313],
            [5.00071179, 5.43214198],
            [5.05191187, 4.00279805],
            [4.78940057, 4.98433394],
            [4.80576066, 4.7085966 ],
            [5.00122864, 4.89035326],
            [4.18086558, 4.54712584],
            [5.8869542 , 4.05308928],
            [4.07999738, 4.56628072],
            [5.16468834, 5.98178561],
            [5.98528447, 5.98623474],
            [4.22009666, 5.32896289],
            [5.04797367, 4.34629982],
            [5.88592049, 4.48372017],
            [5.99786454, 5.16538763],
            [4.366558  , 4.77369084],
            [4.37934706, 4.82154135],
            [5.18936014, 5.43317219],
            [4.97378296, 4.61917964],
            [5.15488275, 4.88341564],
            [4.71935621, 4.64266386],
            [4.41641448, 4.90251725],
            [4.98368582, 5.79815263],
            [5.45872092, 5.54017955],
            [4.7508785 , 4.68747907],
            [5.31007041, 5.42207599],
            [4.22707515, 4.26605738],
            [4.91207812, 4.31947246],
            [5.92328381, 5.67523149],
            [5.04032137, 4.43654452],
            [4.26983745, 5.95814069],
            [5.41408699, 5.71995111],
            [4.77434526, 4.50166804],
            [4.59887604, 5.71379106],
            [4.94596798, 5.32655409],
            [5.61145721, 4.50596101],
            [4.15914688, 5.46552121],
            [5.92279496, 5.90760947],
            [4.9809981 , 5.26438413],
            [5.46599004, 5.80481901],
            [4.32449384, 4.81176264],
            [4.83418147, 5.39118206],
            [4.84969448, 5.71622845],
            [5.69386496, 4.14039823],
            [4.60350483, 5.95924736],
            [4.07125399, 4.98478529],
            [5.90475371, 5.62114752],
            [4.58866088, 5.19246704],
            [4.8623557 , 5.18479501],
            [5.78750421, 5.10804238],
            [4.98573301, 4.63854091],
            [4.52673157, 5.08456123],
            [4.16452905, 5.27127342],
            [5.59281045, 5.90949501],
            [5.36924854, 4.97658633],
            [4.97082862, 5.93338584],
            [4.42269577, 4.82329628],
            [5.97933115, 4.05682371],
            [5.40265303, 4.05034313],
            [4.64176345, 4.14705412],
            [4.12176913, 4.22281263],
            [4.33853782, 5.25537256],
            [4.87678619, 5.66180753],
            [4.47958438, 4.38010542],
            [5.42379932, 5.71658985],
            [5.11811177, 5.40884082],
            [5.21022407, 5.11843457],
            [5.72078838, 5.83951072],
            [5.69921465, 4.50893307],
            [5.75511108, 4.87026038],
            [5.45898869, 4.82528154],
            [4.38167209, 5.41203904],
            [4.48126564, 5.70264885],
            [5.64820458, 5.05042357],
            [4.77268159, 5.18176158],
            [4.27504723, 5.61654082],
            [5.93165163, 5.55959161],
            [4.47867016, 5.73452083],
            [5.61623003, 4.12736225],
            [4.46245661, 5.1793709 ],
            [4.2749739 , 5.35688141],
            [5.98438138, 4.57150397],
            [5.52182552, 4.09305434],
            [4.66507181, 5.88910558],
            [5.27303408, 5.20369721],
            [5.85636936, 4.36335882],
            [4.03564637, 4.38014435],
            [5.0437436 , 4.99164397],
            [5.60098241, 5.71887262],
            [4.42591206, 4.87453768],
            [4.84323502, 4.10943475],
            [4.01986739, 5.57953136],
            [4.55062644, 5.43548001],
            [4.84271184, 4.28667174],
            [4.38504336, 4.62763047],
            [5.61034033, 4.02525166],
            [4.09821195, 5.13200077],
            [5.37362139, 5.45362181],
            [4.95938752, 4.73531344],
            [5.6799402 , 4.9083271 ],
            [4.64273168, 4.18543973],
            [4.12087586, 4.18190233],
            [5.36541291, 5.36147153],
            [4.48634833, 5.28092288],
            [4.13827837, 5.74583992],
            [4.2192139 , 4.33811153],
            [4.93475598, 5.55189844],
            [5.70888903, 4.42077289],
            [4.15328374, 5.57782959],
            [5.095     , 5.57250973],
            [5.84009409, 4.96194553],
            [4.91910734, 5.19795831],
            [5.19863756, 5.0087469 ],
            [4.61375706, 5.08270596],
            [5.84985389, 5.9411016 ],
            [4.79158922, 5.59749055],
            [5.2701763 , 4.45993833],
            [4.10241419, 4.05692761],
            [4.2456955 , 4.44042504],
            [5.65804551, 4.57098366],
            [5.56212817, 5.00933163],
            [4.27689785, 5.5560731 ],
            [5.84266359, 5.88603726],
            [5.40887159, 5.38783289],
            [5.09310363, 4.73843446],
            [5.96493515, 4.13121845],
            [5.79535662, 4.52786198],
            [5.14895168, 5.02573254],
            [5.10895362, 5.29433466],
            [4.37094832, 4.54395641],
            [4.29687732, 4.06060834],
            [5.87851117, 4.69350826],
            [4.21912921, 4.75665399],
            [4.76815892, 5.33084738],
            [4.48898186, 5.32296195],
            [4.19698577, 5.1617255 ],
            [4.21373101, 5.09650901],
            [5.03950342, 4.59246824],
            [4.9114581 , 4.07733304],
            [5.19800605, 4.00097352],
            [5.00362719, 5.00345224],
            [0.60669696, 9.49836939],
            [6.08659045, 6.72002684],
            [4.62773524, 7.04273095],
            [1.81067141, 6.47582177],
            [5.68108735, 9.54138315],
            [7.96690239, 5.85310393],
            [4.55354967, 7.38451531],
            [8.12236278, 9.27291174],
            [8.26375607, 0.29956598],
            [7.72802822, 5.21777394],
            [8.85387332, 4.58561667],
            [5.40722576, 9.07764998],
            [5.5631004 , 1.87114722],
            [6.81111368, 0.13846685],
            [3.80091658, 8.0868449 ],
            [5.76125412, 1.51784505],
            [3.2985476 , 9.51773146],
            [0.30384723, 7.78173527],
            [1.8138386 , 2.13656868],
            [6.52596223, 1.44091946],
            [3.81007271, 4.05734533],
            [6.58965435, 5.11822674],
            [4.82585104, 5.24021798],
            [8.06285873, 8.04391081],
            [1.47700436, 9.93453577],
            [0.67767736, 9.95815807],
            [1.23449693, 0.61929198],
            [7.04035442, 9.60979653],
            [7.12150948, 6.63081322],
            [1.33674962, 1.80345169],
            [1.56715869, 0.90473481],
            [9.71866148, 5.0986113 ],
            [4.50877476, 3.29700978],
            [4.93512042, 1.43600985],
            [1.50920291, 8.15338579],
            [2.11281884, 0.86542322],
            [3.87257418, 8.93403414],
            [4.28643298, 7.66304012],
            [9.95265402, 0.20062851],
            [1.08198229, 6.94742671],
            [8.9066405 , 4.92476324],
            [0.7026799 , 0.88437364],
            [1.06208159, 2.01258878],
            [8.77364005, 9.43222502],
            [1.35636138, 8.58726973],
            [7.57158282, 6.86171614],
            [8.5639339 , 2.30877753],
            [5.19337979, 3.43333445],
            [6.58772929, 2.82790254],
            [5.02466434, 3.0327752 ],
            [5.52529423, 1.61127097],
            [5.68548965, 4.85709868],
            [1.27519544, 5.43692169],
            [2.00490609, 6.7016086 ],
            [5.58112217, 2.32378317],
            [5.15588486, 3.14991973],
            [8.46592549, 4.4737628 ],
            [1.00085224, 9.01592085],
            [8.56088249, 2.86329874],
            [2.5901477 , 0.66371374],
            [3.17631672, 0.51842995],
            [9.44188318, 7.17217302],
            [5.536592  , 3.5974477 ],
            [1.59182301, 4.32958041],
            [2.79362177, 9.61037632],
            [0.98132156, 4.06995552],
            [0.08376446, 5.6805893 ],
            [5.76609768, 1.37144261],
            [6.72218566, 1.42873727],
            [5.09231497, 3.68761953],
            [2.49107389, 1.36282122],
            [1.19291132, 0.52388126],
            [4.34899315, 7.70705381],
            [8.50914182, 6.21282667],
            [3.79887688, 6.79911036],
            [3.13765504, 7.26637983],
            [9.14483191, 0.94895481],
            [6.64976952, 3.56872794],
            [7.6229092 , 9.45005691],
            [2.23742203, 0.57253219],
            [6.4280185 , 0.88777109],
            [6.67938415, 6.07221385],
            [0.1959982 , 3.93793344],
            [4.59507303, 0.21136012],
            [3.73690901, 5.65567137],
            [0.08875177, 1.62845346],
            [8.98131737, 3.55433319],
            [3.09753515, 5.82156678],
            [5.22164622, 8.82455769],
            [2.72344465, 2.99820057],
            [3.08386707, 1.01004272],
            [2.95021699, 7.28904792],
            [3.41878893, 2.86068156],
            [9.56347606, 9.99479151],
            [3.37351921, 4.75121557],
            [8.2964966 , 0.99405146],
            [5.08114642, 8.33836566],
            [7.70122993, 8.1600035 ],
            [0.51549976, 6.81323788],
            [9.63293633, 7.00947457],
            [4.09846472, 2.07040589],
            [9.56022667, 4.06011177],
            [7.16880131, 6.72331408],
            [8.42137103, 9.40740392],
            [4.69801107, 1.07716524],
            [5.66685399, 9.7977703 ],
            [4.33914702, 8.18034436],
            [4.37249597, 8.10534299],
            [2.29608466, 4.10978145],
            [7.3095114 , 6.37314406],
            [7.69142681, 9.56224042],
            [3.76006662, 8.95701422],
            [2.08480496, 0.6319644 ],
            [2.39421673, 7.80199249],
            [3.76075442, 5.8148373 ],
            [2.41702136, 4.38554587],
            [1.80452273, 3.73120468],
            [8.63933391, 8.55993675],
            [9.4007955 , 4.0573999 ],
            [9.82310214, 4.82603944],
            [8.77120782, 4.0887922 ],
            [3.19773226, 6.42468598],
            [7.09365446, 7.42346817],
            [6.04514627, 8.58441547],
            [6.49906838, 2.70485889],
            [8.58474866, 1.80824968],
            [5.37705115, 3.86929918],
            [6.19585209, 5.48586306],
            [4.63894012, 8.26444838],
            [4.00997827, 6.1950328 ],
            [5.00709118, 5.00889504],
            [6.08903104, 9.95365241],
            [2.47949519, 7.38433406],
            [9.30043918, 5.84412076],
            [8.80647152, 6.9205912 ],
            [3.51418279, 8.20699451],
            [2.37084961, 9.21693668],
            [9.18848905, 6.29530712],
            [5.12223117, 2.99851521],
            [4.51062099, 7.17143583],
            [2.8971077 , 2.08844155],
            [3.16536919, 5.73234617],
            [4.62526227, 5.02618564],
            [4.87241544, 7.7922727 ],
            [3.9013589 , 7.68535178],
            [8.61468263, 1.53450442],
            [6.15354499, 5.13231552],
            [9.89265639, 0.82825288],
            [7.11469926, 2.73154417],
            [9.94941426, 3.54727914],
            [0.88625414, 9.83250377],
            [4.79242982, 4.8216702 ],
            [5.26545416, 9.43074327],
            [4.90703852, 0.30130396],
            [9.35470343, 7.47280001],
            [7.30411492, 1.05281154],
            [7.65602563, 7.07879045],
            [9.22538144, 5.09071362],
            [7.12109968, 8.71048322],
            [4.30661815, 4.87128691],
            [1.222318  , 1.90016084],
            [5.9974053 , 2.98077629],
            [1.82683368, 5.0302769 ],
            [9.50669637, 2.32580092],
            [7.39082044, 7.61671314],
            [7.6605648 , 9.24495802],
            [3.22808726, 6.78942884],
            [1.7181999 , 7.09573944],
            [4.89323385, 4.41921269],
            [6.46575798, 4.78433582],
            [7.80983565, 5.6378446 ],
            [8.49160372, 3.32487493],
            [6.87638346, 3.10325858],
            [6.60795462, 3.54474678],
            [0.61312486, 4.43908485],
            [7.0832585 , 2.57418658],
            [6.18363028, 1.21625279],
            [6.37324308, 0.58291747],
            [0.47624101, 5.11944559],
            [2.99314991, 3.46436135],
            [6.49217435, 9.4888827 ],
            [5.16241879, 7.13543441],
            [5.86161249, 3.94935838],
            [6.71816525, 2.37170576],
            [8.48209146, 2.92473167],
            [1.20656326, 4.31184195],
            [2.71951561, 2.73726276],
            [6.42202569, 4.49912328],
            [9.08484773, 7.75397168],
            [3.3911252 , 7.0374065 ],
            [2.39118621, 2.05161076],
            [6.83968601, 4.16695289],
            [1.95211447, 3.90329575],
            [0.98523423, 3.2254438 ],
            [8.89478136, 9.42271915],
            [6.57470757, 1.95628749],
            [5.25678761, 3.10809104],
            [5.55348394, 5.35529807],
            [4.65112929, 7.67864594],
            [8.86946972, 8.29809368]])
```

1.0.1 1. Decision Tree

Use **Data3** to answer the following questions.

Question 1a: Compute and print the 10-fold cross-validation accuracy using decision tree classifiers with max_depth = 2,4,6,8,10, and 50.

[7]:
```
dt2 = DecisionTreeClassifier(max_depth=2)
dt_scores2 = cross_val_score(dt2, Data3_X, Data3_Y, cv=10, scoring='accuracy')
sum2 = 0
for i in range(len(dt_scores2)):
    sum2 = sum2 + dt_scores2[i]
print(sum2/len(dt_scores2))
```

```
0.875
```

[8]:
```
dt4 = DecisionTreeClassifier(max_depth=4)
dt_scores4 = cross_val_score(dt4, Data3_X, Data3_Y, cv=10, scoring='accuracy')
sum4 = 0
for i in range(len(dt_scores4)):
    sum4 = sum4 + dt_scores4[i]
print(sum4/len(dt_scores4))
```

```
0.97
```

[9]:
```
dt6 = DecisionTreeClassifier(max_depth=6)
dt_scores6 = cross_val_score(dt6, Data3_X, Data3_Y, cv=10, scoring='accuracy')
sum6 = 0
for i in range(len(dt_scores6)):
    sum6 = sum6 + dt_scores6[i]
print(sum6/len(dt_scores6))
```

```
0.9674999999999999
```

[10]:
```
dt8 = DecisionTreeClassifier(max_depth=8)
dt_scores8 = cross_val_score(dt8, Data3_X, Data3_Y, cv=10, scoring='accuracy')
sum8 = 0
for i in range(len(dt_scores8)):
    sum8 = sum8 + dt_scores8[i]
print(sum8/len(dt_scores8))
```

```
0.95
```

[11]:
```
dt10 = DecisionTreeClassifier(max_depth=10)
dt_scores10 = cross_val_score(dt10, Data3_X, Data3_Y, cv=10, scoring='accuracy')
sum10 = 0
for i in range(len(dt_scores10)):
    sum10 = sum10 + dt_scores10[i]
print(sum10/len(dt_scores10))
```

```
0.9424999999999999
```

[12]:
```
dt50 = DecisionTreeClassifier(max_depth=50)
dt_scores50 = cross_val_score(dt50, Data3_X, Data3_Y, cv=10, scoring='accuracy')
sum50 = 0
for i in range(len(dt_scores50)):
    sum50 = sum50 + dt_scores50[i]
print(sum50/len(dt_scores50))
```

```
0.9450000000000001
```

Question 1b: For what values of max_depth did you observe the lowest accuracy? What is this phenomenon called?

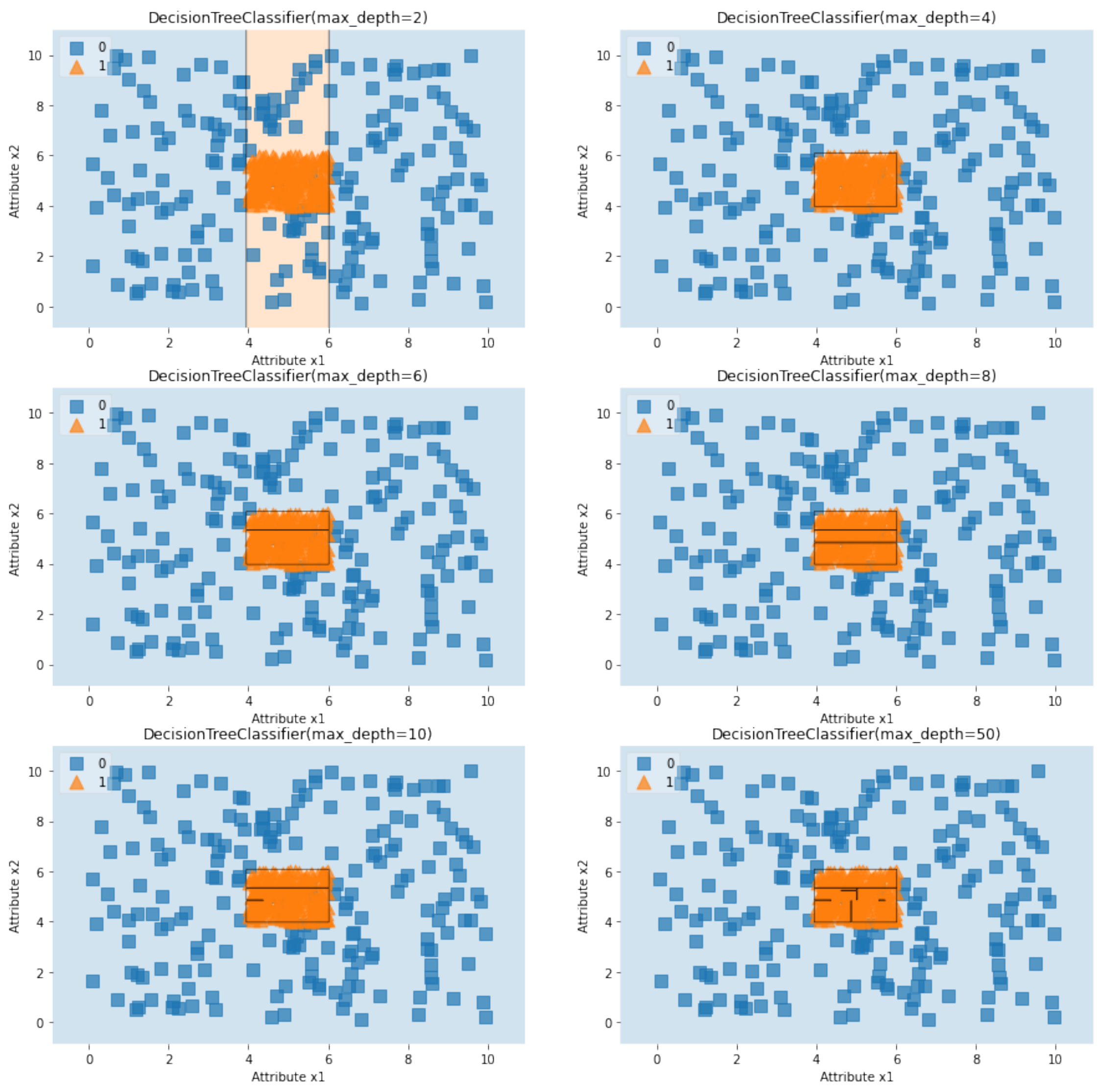
Answer: The deeper you allow the model to go, the more complex your model will become. For training error if we increase max_depth, training error will always go down (or at least not go up). For testing error if we set max_depth too high, then the decision tree might simply overfit the training data without capturing useful patterns as we would like; this will cause testing error to increase. But if we set it too low, then we might be giving the decision tree too little flexibility to capture the patterns and interactions in the training data. This will also cause the testing error to increase. For our data the nice golden spot where the max depth is not too low and not too high give the best accuracy i.e. 97% for max depth=4.

Question 1c: What accuracy did you observe for max depth=50? What is the difference between this accuracy and the highest accuracy? What is this phenomenon called?

Answer: Depth = 50 has accuracy of 94.50% and the highest accuracy is observed for Max Depth = 4 i.e. 97%. So, as we increase the max_depth the accuracy of model decreases as the model is overfitting the training set.

Question 1d: Plot decision regions for the above decision tree models

[13]:
```
# parameters to set size or markers, contours, transparency, models and labels
scatter_kwargs = {'s': 120, 'edgecolor': None, 'alpha': 0.7}
contourf_kwargs = {'alpha': 0.2}
scatter_highlight_kwargs = {'s': 120, 'label': 'Test data', 'alpha': 0.7}
clf_list = [dt2, dt4, dt6, dt8, dt10, dt50]
labels =␣
 ↪['Decision_Tree_MaxDepth2','Decision_Tree_MaxDepth4','Decision_Tree_MaxDepth6','Decision_Tr
 ↪'Decision_Tree_MaxDepth10', 'Decision_Tree_MaxDepth50']

# Plotting the decision boundaries
fig = plt.figure(figsize=(15, 15))
count = 0;

for clf, label in zip(clf_list, labels):
    count = count + 1;
    clf.fit(Data3_X, Data3_Y)
    ax = plt.subplot(3,2,count)
    fig = plot_decision_regions(X=Data3_X, y=Data3_Y, clf=clf, legend=2,
                          scatter_kwargs=scatter_kwargs,
                          contourf_kwargs=contourf_kwargs,
                          scatter_highlight_kwargs=scatter_highlight_kwargs)
    plt.xlabel('Attribute x1')
    plt.ylabel('Attribute x2')
    plt.title(str(clf))
plt.show()
```



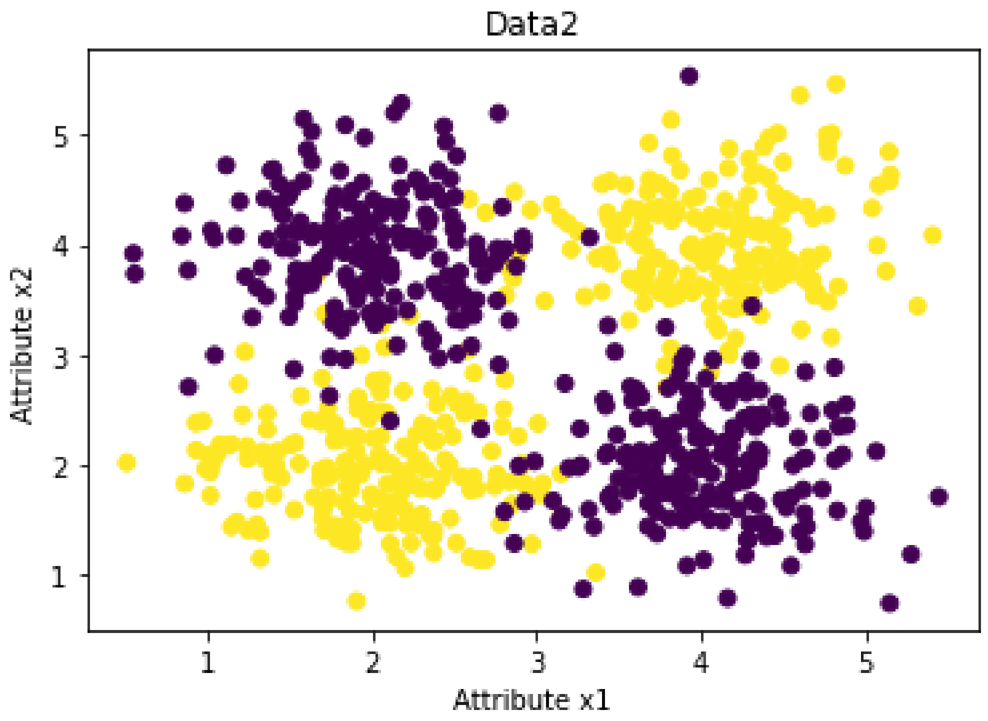
Question 1e: Based on the decision regions, which depth is better suited for this data? Explain your reason.

Answer: As we have highest accuracy for max depth 4 and also it is the point where the training data set is not underfitting nor overfitting, so max depth 4 is better suited for this data.

1.0.2 2. k Nearest Neighbor

Use **Data2** to answer the following questions.

```
[14]: import matplotlib.pyplot as plt
      plt.scatter(Data2_X[:,0],Data2_X[:,1], c=Data2_Y)
      plt.xlabel('Attribute x1')
      plt.ylabel('Attribute x2')
      plt.title('Data2')
      plt.show()
```



Question 2a: Compute and print the 10-fold cross-validation accuracy for a kNN classifier with n_neighbors = 1, 5, 10, 50

[15]:
```
knn = KNeighborsClassifier(n_neighbors=1)
knn_scores1 = cross_val_score(knn, Data2_X, Data2_Y, cv=10, scoring='accuracy')
print(knn_scores1)
sumk1 = 0
for i in range(len(knn_scores1)):
    sumk1 = sumk1 + knn_scores1[i]
print(sumk1/len(knn_scores1))
```

```
[0.925  0.8875 0.925  0.8875 0.925  0.9375 0.8875 0.925  0.9375 0.8875]
0.9125
```

[16]:
```
knn1 = KNeighborsClassifier(n_neighbors=5)
knn_scores5 = cross_val_score(knn1, Data2_X, Data2_Y, cv=10, scoring='accuracy')
print(knn_scores5)
sumk5 = 0
for i in range(len(knn_scores5)):
    sumk5 = sumk5 + knn_scores5[i]
print(sumk5/len(knn_scores5))
```

```
[0.9875 0.9125 0.925  0.9125 0.95   0.95   0.8625 0.95   0.9625 0.9375]
0.9349999999999999
```

[17]:
```
knn2 = KNeighborsClassifier(n_neighbors=10)
knn_scores10 = cross_val_score(knn2, Data2_X, Data2_Y, cv=10,␣
 ↪scoring='accuracy')
print(knn_scores10)
sumk10 = 0
for i in range(len(knn_scores10)):
    sumk10 = sumk10 + knn_scores10[i]
print(sumk10/len(knn_scores10))
```

```
[0.9875 0.9    0.95   0.925  0.9625 0.95   0.8625 0.9375 0.9625 0.9625]
0.9400000000000001
```

[18]:
```
knn3 = KNeighborsClassifier(n_neighbors=50)
knn_scores50 = cross_val_score(knn3, Data2_X, Data2_Y, cv=10,␣
 ↪scoring='accuracy')
print(knn_scores50)
sumk50 = 0
for i in range(len(knn_scores50)):
    sumk50 = sumk50 + knn_scores50[i]
print(sumk50/len(knn_scores50))
```

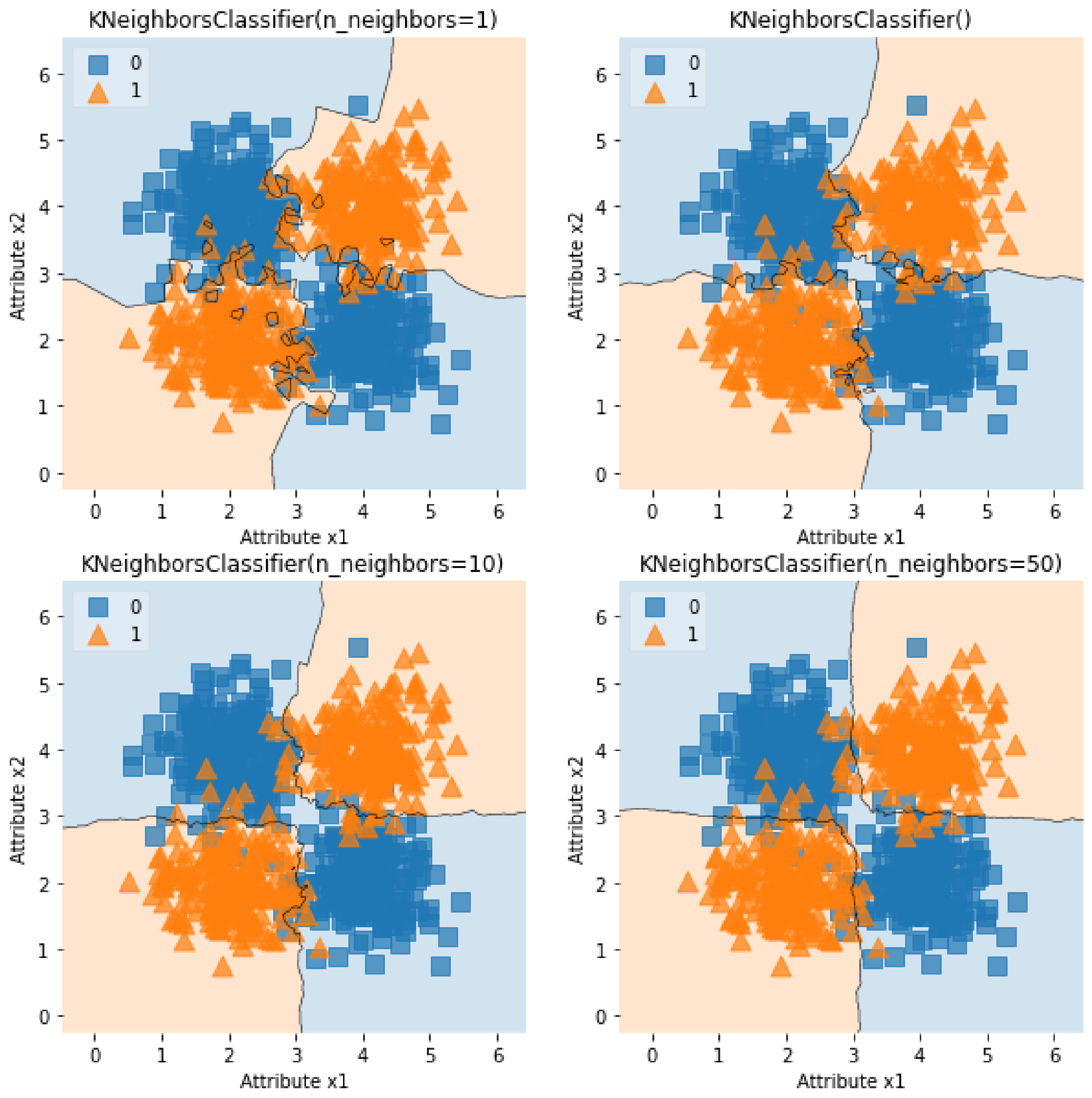
```
[0.9875 0.9    0.9625 0.9125 0.9625 0.9375 0.8875 0.9375 0.9625 0.9625]
0.9412500000000001
```

Question 2b: For what values of n_neighbors did you observe the lowest accuracy? What is this phenomenon called?

Answer: Lowest accuracy is observed for nearest neighbours with value as 1, this is because of the underfitting of data. The model is overfitting the training set; when new test data comes into the picture the it will give more errors, so there will be low bias and high variance for the model.

Question 2c: Plot decision regions for a kNN classifier with n_neighbors = 1, 5, 10, 50

[19]:
```
# parameters to set models and labels
clf_list = [knn, knn1, knn2, knn3]
labels =␣
 ↪['kNN_neighbours_1','kNN_neighbours_5','kNN_neighbours_10','kNN_neighbours_50']

# Plotting the decision boundaries
fig = plt.figure(figsize=(10,10))
count = 0;

for clf, label in zip(clf_list, labels):
    count = count + 1;
    clf.fit(Data2_X, Data2_Y)
    ax = plt.subplot(2,2,count)
    fig = plot_decision_regions(X=Data2_X, y=Data2_Y, clf=clf, legend=2,
                          scatter_kwargs=scatter_kwargs,
                          contourf_kwargs=contourf_kwargs,
                          scatter_highlight_kwargs=scatter_highlight_kwargs)
    plt.xlabel('Attribute x1')
    plt.ylabel('Attribute x2')
    plt.title(str(clf))
plt.show()
```



Question 2d: From the plots for **Question 2c** what do you notice about the nature of decision boundary as the n_neighbors are increasing.

Answer: As the k-nearest-neighbours are increased from 1 to 50 the model tries to generalize over the testing set, which lets us get more points in considertaion to predict the class of test data point. From the above graphs we can see that, when k=1 the model is trying to capture each and every training instance for that particular class, which may overfit the data and create errors when test data comes into picture, while when k=50 the model gives more generalized results.

1.0.3 3. Naive Bayes

Question 3a: Compute and print the 10-fold cross-validation accuracy for a NB classifier on all four datasets: Data1, Data2, Data3, Data4

[20]:
```
nb1 = GaussianNB()
nb_scores1 = cross_val_score(nb1, Data1_X, Data1_Y, cv=10, scoring='accuracy')
print(nb_scores1)
sumd1 = 0
for i in range(len(nb_scores1)):
    sumd1 = sumd1 + nb_scores1[i]
print(sumd1/len(nb_scores1))
```

```
[0.975 1.    1.    0.925 0.95  0.975 0.975 0.9   0.975 1.   ]
0.9675
```

[21]:
```
nb2 = GaussianNB()
nb_scores2 = cross_val_score(nb2, Data2_X, Data2_Y, cv=10, scoring='accuracy')
print(nb_scores2)
sumd2 = 0
for i in range(len(nb_scores2)):
    sumd2 = sumd2 + nb_scores2[i]
print(sumd2/len(nb_scores2))
```

```
[0.075  0.0625 0.0125 0.0875 0.0875 0.025  0.05   0.05   0.0125 0.0375]
0.05
```

[22]:
```
nb3 = GaussianNB()
nb_scores3 = cross_val_score(nb3, Data3_X, Data3_Y, cv=10, scoring='accuracy')
print(nb_scores3)
sumd3 = 0
for i in range(len(nb_scores3)):
    sumd3 = sumd3 + nb_scores3[i]
print(sumd3/len(nb_scores3))
```

```
[1.    0.95  0.975 0.975 0.975 0.975 0.925 0.9   0.975 0.95 ]
0.96
```

[23]:
```
nb4 = GaussianNB()
nb_scores4 = cross_val_score(nb4, Data4_X, Data4_Y, cv=10, scoring='accuracy')
print(nb_scores4)
sumd4 = 0
for i in range(len(nb_scores4)):
    sumd4 = sumd4 + nb_scores4[i]
print(sumd4/len(nb_scores4))
```

```
[0.92 1.   0.98 0.98 0.96 0.96 0.94 0.96 0.98 0.96]
0.9640000000000001
```

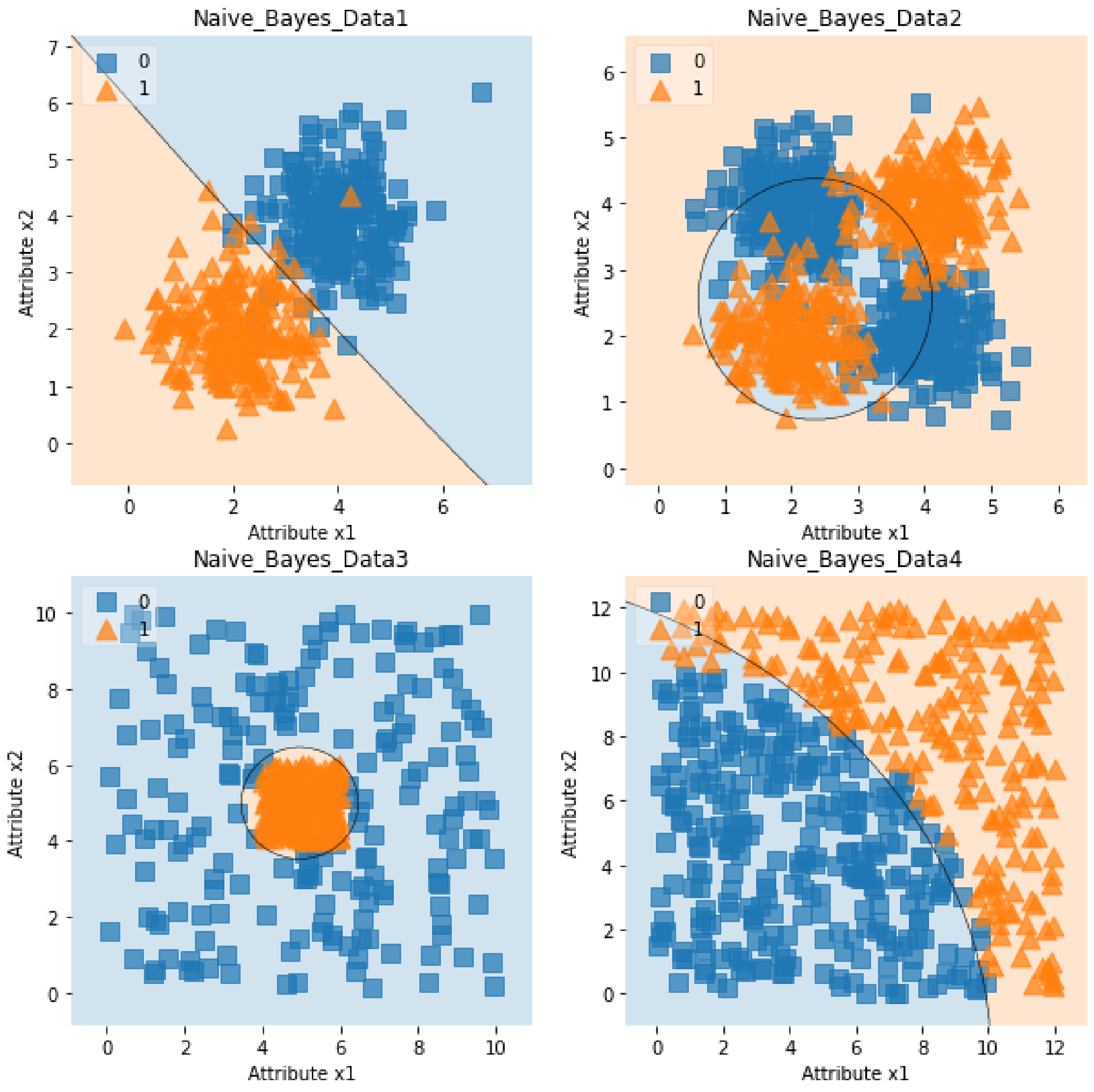
Question 3b: State your observations on the datasets the NB algorithm performed poorly.

Answer: We can see that because of the nature of dataset 2 there is high possibility that points will be misclassified as there can be multiple unique gaussians plotted for both the classes. Hence, dataset 2 has the lowest accuracy.

Question 3c: Plot decision regions for a NB classifier on each of the four datasets

[24]:
```
# parameters to set models and labels
clf_list = [nb1, nb2, nb3, nb4]
labels =␣
 ↪['Naive_Bayes_Data1','Naive_Bayes_Data2','Naive_Bayes_Data3','Naive_Bayes_Data4']
a = [1, 2, 3, 4]

# Plotting the decision boundaries
fig = plt.figure(figsize=(10,10))

clf_list[0].fit(Data1_X, Data1_Y)
ax = plt.subplot(2,2,1)
fig = plot_decision_regions(X=Data1_X, y=Data1_Y, clf=clf_list[0], legend=2,
                  scatter_kwargs=scatter_kwargs,
                  contourf_kwargs=contourf_kwargs,
                  scatter_highlight_kwargs=scatter_highlight_kwargs)
plt.xlabel('Attribute x1')
plt.ylabel('Attribute x2')
plt.title(labels[0])

clf_list[1].fit(Data2_X, Data2_Y)
ax = plt.subplot(2,2,2)
fig = plot_decision_regions(X=Data2_X, y=Data2_Y, clf=clf_list[1], legend=2,
                  scatter_kwargs=scatter_kwargs,
                  contourf_kwargs=contourf_kwargs,
                  scatter_highlight_kwargs=scatter_highlight_kwargs)
plt.xlabel('Attribute x1')
plt.ylabel('Attribute x2')
plt.title(labels[1])

clf_list[2].fit(Data3_X, Data3_Y)
ax = plt.subplot(2,2,3)
fig = plot_decision_regions(X=Data3_X, y=Data3_Y, clf=clf_list[2], legend=2,
                  scatter_kwargs=scatter_kwargs,
                  contourf_kwargs=contourf_kwargs,
                  scatter_highlight_kwargs=scatter_highlight_kwargs)
```

```
plt.xlabel('Attribute x1')
plt.ylabel('Attribute x2')
plt.title(labels[2])

clf_list[3].fit(Data4_X, Data4_Y)
ax = plt.subplot(2,2,4)
fig = plot_decision_regions(X=Data4_X, y=Data4_Y, clf=clf_list[3], legend=2,
                  scatter_kwargs=scatter_kwargs,
                  contourf_kwargs=contourf_kwargs,
                  scatter_highlight_kwargs=scatter_highlight_kwargs)
plt.xlabel('Attribute x1')
plt.ylabel('Attribute x2')
plt.title(labels[3])

plt.show()
```



Question 3d: Describe the shape of the decision boundary on all four datasets. Explain the reason.

Answer: Decision boundary for dataset 1 is linear, for dataset 2 and 3 it is somewhat circular in shapr and for dataset 4 it is a curve.

Question 3e: Based on your plots in **Question 3c** explain the poor performance of NB on some datasets.

Answer: Naive Bayes performs poorly on dataset 2 because. It tries fitting a curve/ellipse around a test point and finds the probability of the point to be in a particular class and assigns the the class with high posteriror probability i.e. probability of the class given data point. For, Data 2 Naive Bayes failed to plot such decision boundary/ ellipse that can clearly split the data of different classes.

1.0.4 4. Support Vector Machines (Linear)

Question 4a: Based on the visualization of the four datasets, assess how well a linear SVM is expected to perform. Specifically, rank the datasets in the order of decreasing accuracy when a linear SVM is used. No need to compute accuracy to answer this question.

Answer: The decreasing order of perfromance for linear SVM will be as followe: Data1, Data4, Data3, Data2. It will work better on Data1 and Data4 as we can draw a clear hyper-plane seperating both the classes. Linear svm will not work better on data2 and data3 as we cannot draw a hyper-plane that seperates the classes.

Question 4b: Compute and print the 10-fold cross-validation accuracy for a linear SVM classifier on all four datasets: Data1, Data2, Data3, Data4

[25]:
```
svm_linear1 = SVC(C=0.5, kernel='linear')
svm_linear_scores1 = cross_val_score(svm_linear1, Data1_X, Data1_Y, cv=10,␣
 ↪scoring='accuracy')
print(svm_linear_scores1)
svm1 = 0
for i in range(len(svm_linear_scores1)):
    svm1 = svm1 + svm_linear_scores1[i]
print(svm1/len(svm_linear_scores1))
```

```
[0.975 1.    1.    0.95  0.95  0.95  0.975 0.9   0.975 1.   ]
0.9675
```

[26]:
```
svm_linear2 = SVC(C=0.5, kernel='linear')
svm_linear_scores2 = cross_val_score(svm_linear2, Data2_X, Data2_Y, cv=10,␣
 ↪scoring='accuracy')
print(svm_linear_scores1)
svm2 = 0
for i in range(len(svm_linear_scores2)):
    svm2 = svm2 + svm_linear_scores2[i]
```

```
print(svm2/len(svm_linear_scores2))
```

```
[0.975 1.    1.    0.95  0.95  0.95  0.975 0.9   0.975 1.   ]
0.14125000000000001
```

[27]:
```
svm_linear3 = SVC(C=0.5, kernel='linear')
svm_linear_scores3 = cross_val_score(svm_linear3, Data3_X, Data3_Y, cv=10,␣
 ↪scoring='accuracy')
print(svm_linear_scores3)
svm3 = 0
for i in range(len(svm_linear_scores3)):
    svm3 = svm3 + svm_linear_scores3[i]
print(svm3/len(svm_linear_scores3))
```

```
[0.625 0.625 0.65  0.6   0.65  0.7   0.65  0.675 0.625 0.625]
0.6425
```

[28]:
```
svm_linear4 = SVC(C=0.5, kernel='linear')
svm_linear_scores4 = cross_val_score(svm_linear4, Data4_X, Data4_Y, cv=10,␣
 ↪scoring='accuracy')
print(svm_linear_scores4)
svm4 = 0
for i in range(len(svm_linear_scores4)):
    svm4 = svm4 + svm_linear_scores4[i]
print(svm4/len(svm_linear_scores4))
```

```
[0.94 0.92 0.9  0.94 0.94 0.9  0.94 0.92 0.96 0.86]
0.9219999999999999
```

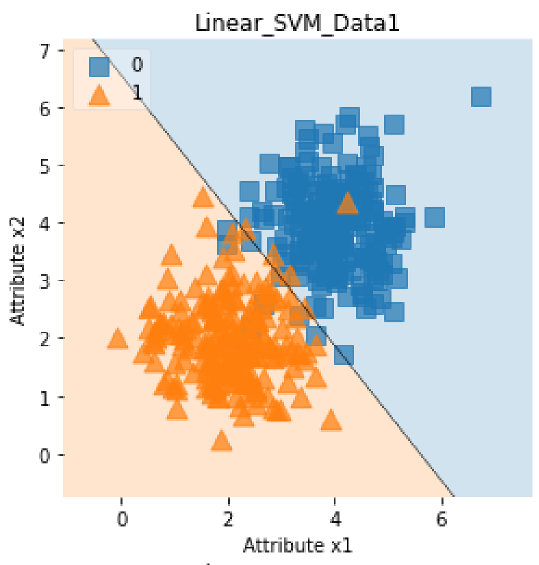
Question 4c: Rank the datasets in the decreasing order of accuracy of SVM.

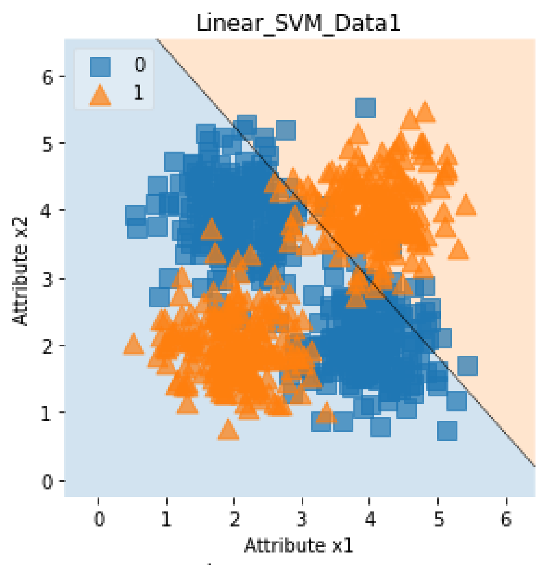
Answer: Decreasing Order of Accuracy - Data1, Data4, Data3, Data2

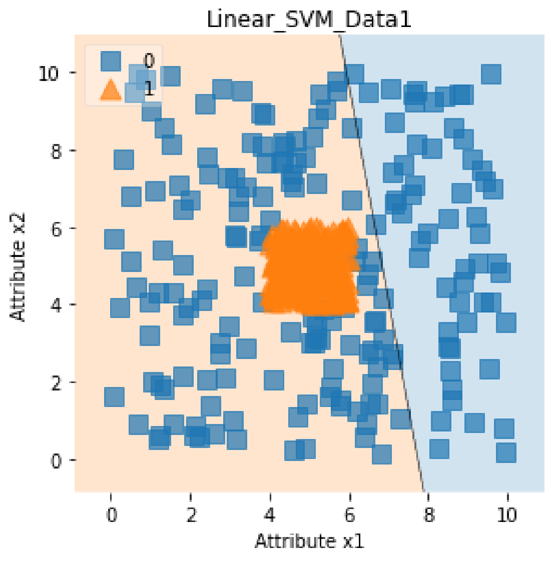
Question 4d: Plot decision regions for a linear SVM classifier on each of the four datasets

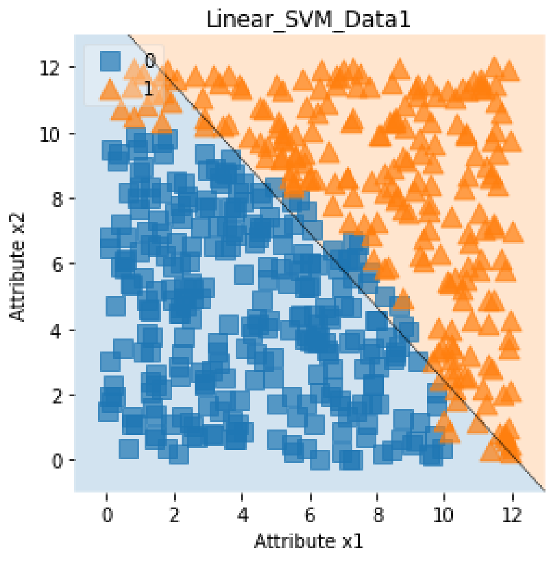
[29]:
```
# parameters to set models and labels
clf_list = [svm_linear1, svm_linear2, svm_linear3, svm_linear4]
labels =␣
 ↪['Linear_SVM_Data1','Linear_SVM_Data1','Linear_SVM_Data1','Linear_SVM_Data1']
a = [1, 2, 3, 4]

# Plotting the decision boundaries
fig = plt.figure(figsize=(10,10))

clf_list[0].fit(Data1_X, Data1_Y)
ax = plt.subplot(2,2,1)
fig = plot_decision_regions(X=Data1_X, y=Data1_Y, clf=clf_list[0], legend=2,
                  scatter_kwargs=scatter_kwargs,
                  contourf_kwargs=contourf_kwargs,
```

```python
                  scatter_highlight_kwargs=scatter_highlight_kwargs)
plt.xlabel('Attribute x1')
plt.ylabel('Attribute x2')
plt.title(labels[0])

clf_list[1].fit(Data2_X, Data2_Y)
ax = plt.subplot(2,2,2)
fig = plot_decision_regions(X=Data2_X, y=Data2_Y, clf=clf_list[1], legend=2,
                  scatter_kwargs=scatter_kwargs,
                  contourf_kwargs=contourf_kwargs,
                  scatter_highlight_kwargs=scatter_highlight_kwargs)
plt.xlabel('Attribute x1')
plt.ylabel('Attribute x2')
plt.title(labels[1])

clf_list[2].fit(Data3_X, Data3_Y)
ax = plt.subplot(2,2,3)
fig = plot_decision_regions(X=Data3_X, y=Data3_Y, clf=clf_list[2], legend=2,
                  scatter_kwargs=scatter_kwargs,
                  contourf_kwargs=contourf_kwargs,
                  scatter_highlight_kwargs=scatter_highlight_kwargs)
plt.xlabel('Attribute x1')
plt.ylabel('Attribute x2')
plt.title(labels[2])

clf_list[3].fit(Data4_X, Data4_Y)
ax = plt.subplot(2,2,4)
fig = plot_decision_regions(X=Data4_X, y=Data4_Y, clf=clf_list[3], legend=2,
                  scatter_kwargs=scatter_kwargs,
                  contourf_kwargs=contourf_kwargs,
                  scatter_highlight_kwargs=scatter_highlight_kwargs)
plt.xlabel('Attribute x1')
plt.ylabel('Attribute x2')
plt.title(labels[3])

plt.show()
```









Question 4e: Explain the reason for your observations in **Question 4c** using observations from the above decision regions.

Answer: For data set 1 and 4 SVM is able to find a linear seperation between classes thay classifies most of the points correctly. Accuracy is less for Data 4 because density of the points near the boundary is more when compared with Data1. For Data3 SVM is able to classify the points of one class completely correct and poins of other class partially which shows less accuracy when compared with Data1 and Data4. Lastly, for Data2 it is not able to classify any of the classes completely correct which resulted in very poor accuracy

1.0.5 5. Non-linear Support Vector Machines

Use **Data2** to answer the following questions.

Question 5a: Compute and print the 10-fold cross-validation accuracy for an SVM with a polynomial kernel and degree values 1, 2, and 3.

[30]:
```
svm_poly1 = SVC(C=0.5, kernel='poly',degree=1, gamma = 'auto')
svm_poly_scores1 = cross_val_score(svm_poly1, Data2_X, Data2_Y, cv=10,␣
 ↪scoring='accuracy')
print(svm_poly_scores1)
svmp1 = 0
for i in range(len(svm_poly_scores1)):
    svmp1 = svmp1 + svm_poly_scores1[i]
print(svmp1/len(svm_poly_scores1))
```

```
[0.1375 0.125  0.0125 0.0875 0.175  0.1875 0.1    0.1625 0.1875 0.1625]
0.13375
```

[31]:
```
svm_poly2 = SVC(C=0.5, kernel='poly',degree=2, gamma = 'auto')
svm_poly_scores2 = cross_val_score(svm_poly2, Data2_X, Data2_Y, cv=10,␣
 ↪scoring='accuracy')
print(svm_poly_scores2)
svmp2 = 0
for i in range(len(svm_poly_scores2)):
    svmp2 = svmp2 + svm_poly_scores2[i]
print(svmp2/len(svm_poly_scores2))
```

```
[0.8125 0.8375 0.8875 0.8375 0.8875 0.8875 0.8625 0.8875 0.9125 0.8375]
0.8649999999999999
```

[32]:
```
svm_poly3 = SVC(C=0.5, kernel='poly',degree=3, gamma = 'auto')
svm_poly_scores3 = cross_val_score(svm_poly3, Data2_X, Data2_Y, cv=10,␣
 ↪scoring='accuracy')
print(svm_poly_scores3)
svmp3 = 0
for i in range(len(svm_poly_scores3)):
    svmp3 = svmp3 + svm_poly_scores3[i]
print(svmp3/len(svm_poly_scores3))
```

```
[0.825  0.875  0.8875 0.8625 0.925  0.9    0.8625 0.8875 0.8875 0.85  ]
0.8762500000000001
```

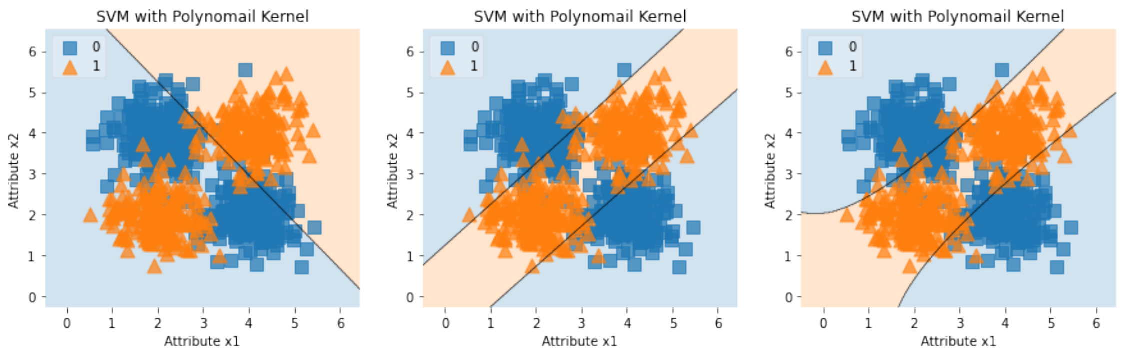
Question 5b: Rank the polynomial kernels in decreasing order of accuracy.

Answer: Decreasing Order of Accuracy as per degree value - Degree3, Degree2, Degree1

Question 5c: Plot decision regions for a polynomial kernel SVM with degree values 1, 2, and 3.

[33]:
```
# parameters to set models and labels
clf_list = [svm_poly1, svm_poly2, svm_poly3]
labels = ['SVM_poly_deg1','SVM_poly_deg2','SVM_poly_deg3']
```

```
# Plotting the decision boundaries
fig = plt.figure(figsize=(15,4))
count = 0;

for clf, label in zip(clf_list, labels):
    count = count + 1;
    clf.fit(Data2_X, Data2_Y)
    ax = plt.subplot(1,3,count)
    fig = plot_decision_regions(X=Data2_X, y=Data2_Y, clf=clf, legend=2,
                          scatter_kwargs=scatter_kwargs,
                          contourf_kwargs=contourf_kwargs,
                          scatter_highlight_kwargs=scatter_highlight_kwargs)
    plt.xlabel('Attribute x1')
    plt.ylabel('Attribute x2')
    plt.title('SVM with Polynomail Kernel')
plt.show()
```



Question 5d: Based on the decision regions, explain the reason for your observations in **Question 5c**.

Answer: For degree = 1, the polynomial kernel works same like linear SVM kernel and tries to find the boundary for the data. When we increase the degree value of the polynomial kernel the accuracy increased. For degree 2 it tries to fit a boundary with single curve i.e. quadratic curve. For degree 3 it tries to draw a decision boundry with a curve with two heads

Question 5e: Compute the 10-fold cross-validation accuracy for an SVM with an RBF kernel and gamma values 0.01, 0.1, and 1.

[34]:
```
svm_rbf1 = SVC(C = 0.5, kernel='rbf', gamma=0.01)
svm_rbf_scores1 = cross_val_score(svm_rbf1, Data2_X, Data2_Y, cv=10,␣
 ↪scoring='accuracy')
print(svm_rbf_scores1)
svmr1 = 0
for i in range(len(svm_rbf_scores1)):
    svmr1 = svmr1 + svm_rbf_scores1[i]
```

```
print(svmr1/len(svm_rbf_scores1))
```

```
[0.375  0.3125 0.0875 0.25   0.4375 0.3375 0.3    0.3    0.275  0.3375]
0.30124999999999999
```

[35]:
```
svm_rbf2 = SVC(C = 0.5, kernel='rbf', gamma=0.1)
svm_rbf_scores2 = cross_val_score(svm_rbf2, Data2_X, Data2_Y, cv=10,␣
 ↪scoring='accuracy')
print(svm_rbf_scores2)
svmr2 = 0
for i in range(len(svm_rbf_scores2)):
    svmr2 = svmr2 + svm_rbf_scores2[i]
print(svmr2/len(svm_rbf_scores2))
```

```
[0.975  0.9    0.9375 0.9    0.9625 0.9375 0.8875 0.9375 0.9625 0.9625]
0.93625
```

[36]:
```
svm_rbf3 = SVC(C = 0.5, kernel='rbf', gamma=1)
svm_rbf_scores3 = cross_val_score(svm_rbf3, Data2_X, Data2_Y, cv=10,␣
 ↪scoring='accuracy')
print(svm_rbf_scores3)
svmr3 = 0
for i in range(len(svm_rbf_scores3)):
    svmr3 = svmr3 + svm_rbf_scores3[i]
print(svmr3/len(svm_rbf_scores3))
```

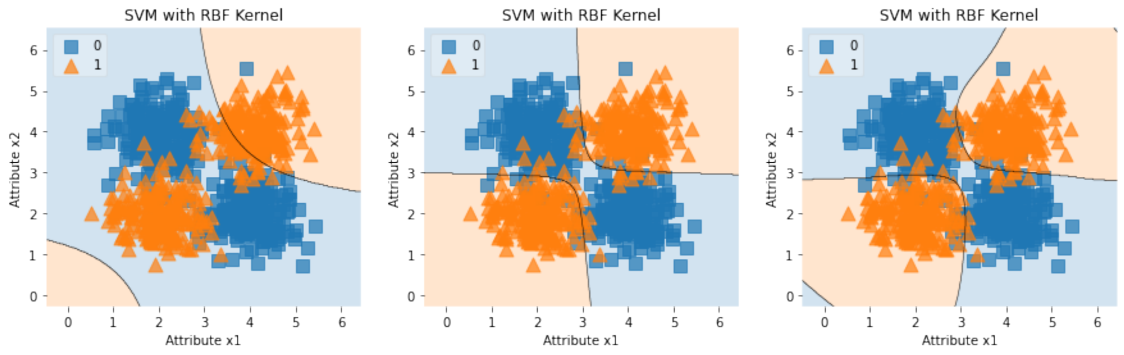
```
[0.9875 0.9125 0.95   0.925  0.9625 0.9375 0.875  0.9375 0.9625 0.95  ]
0.9399999999999998
```

Question 5f: Rank the RBF kernels in decreasing order of accuracy.

Answer: Decreasing Order of Accuracy as per gamma value - 1, 0.1, 0.01

Question 5g: Plot decision regions for the above RBF Kernels

[37]:
```
# parameters to set models and labels
clf_list = [svm_rbf1, svm_rbf2, svm_rbf3]
labels = ['SVM_rbf_0.01','SVM_rbf_0.1','SVM_rbf_1']

# Plotting the decision boundaries
fig = plt.figure(figsize=(15,4))
count = 0;

for clf, label in zip(clf_list, labels):
    count = count + 1;
    clf.fit(Data2_X, Data2_Y)
    ax = plt.subplot(1,3,count)
    fig = plot_decision_regions(X=Data2_X, y=Data2_Y, clf=clf, legend=2,
                      scatter_kwargs=scatter_kwargs,
                              contourf_kwargs=contourf_kwargs,
                              scatter_highlight_kwargs=scatter_highlight_kwargs)
    plt.xlabel('Attribute x1')
    plt.ylabel('Attribute x2')
    plt.title('SVM with RBF Kernel')
plt.show()
```



Question 5h: Explain the reason for your observations in **Question 5f** from the above decision regions.

Answer: For gamma equal to 0.01 the curve of decision boundry is very low and thus the decision region is very broad. when the gamma value increased from 0.001 to 0.1 and 1 the decision boundry started to cover the each class points more accurately. So, as the gamma value increased the spread of the decision region of the kernel also increases.

Question 5i: Between SVM with a Polynomial kernel and SVM with an RBF kernel, which one is ideally suited of Data2? Explain your reason.

Answer: We can see that RBF kernel works better for Data2 as Polynomial SVM kernel generates different hyperplane boundaries. But, on the other hand RBF kernel takes more time when compared with Linear or Polynomial SVM kernel.

1.0.6 6. Classification Evaluation

Question 6a:

Run SVM classifier (with RBF kernel and gamma=0.1) on **Data2** and compute the mean of k-fold cross-validation accuracies for cv = 3, 4, 5 and 6. Report the mean of accuracies for each choice of ‘cv’ and explain the reason for any differences in the mean accuracy you observe.

[38]:
```
svm_rbf4 = SVC(C = 0.5, kernel='rbf', gamma=0.1)
svm_rbf_scoresc3 = cross_val_score(svm_rbf4, Data2_X, Data2_Y, cv=3,␣
 ↪scoring='accuracy')
svm_rbf_scoresc4 = cross_val_score(svm_rbf4, Data2_X, Data2_Y, cv=4,␣
 ↪scoring='accuracy')
svm_rbf_scoresc5 = cross_val_score(svm_rbf4, Data2_X, Data2_Y, cv=5,␣
 ↪scoring='accuracy')
```

```
svm_rbf_scoresc6 = cross_val_score(svm_rbf4, Data2_X, Data2_Y, cv=6,␣
↪scoring='accuracy')
print("Accuraciy Matrices for SVM RBF: \n", svm_rbf_scoresc3, svm_rbf_scoresc4,␣
↪svm_rbf_scoresc5, svm_rbf_scoresc6, "\n")
svmrc3 = 0
svmrc4 = 0
svmrc5 = 0
svmrc6 = 0
for i in range(len(svm_rbf_scoresc3)):
    svmrc3 = svmrc3 + svm_rbf_scoresc3[i]

for i in range(len(svm_rbf_scoresc4)):
    svmrc4 = svmrc4 + svm_rbf_scoresc4[i]

for i in range(len(svm_rbf_scoresc5)):
    svmrc5 = svmrc5 + svm_rbf_scoresc5[i]

for i in range(len(svm_rbf_scoresc6)):
    svmrc6 = svmrc6 + svm_rbf_scoresc6[i]

print("Mean Accuracies For SVM: \n", (svmrc3/len(svm_rbf_scoresc3)), (svmrc4/
↪len(svm_rbf_scoresc4)), (svmrc5/len(svm_rbf_scoresc5)), (svmrc6/
↪len(svm_rbf_scoresc6)))
```

```
Accuraciy Matrices for SVM RBF:
 [0.87265918 0.93632959 0.90225564] [0.91  0.92  0.895 0.94 ] [0.91875 0.9
0.95    0.9125  0.95625] [0.95522388 0.89552239 0.94736842 0.90977444 0.93984962
0.94736842]

Mean Accuracies For SVM:
 0.903748134380896 0.91625 0.9275 0.9325178618187259
```

Answer: Mean accuracies keeps on increasing as we increase the value of cv for Data 2. As the number of folds increases the accuracy value increases and also the # of validations increases which in turn reults in the increase of the mean accuracy.

Question 6b:

For DT, NB, kNN, Linear SVM, Polynomial Kernel SVM, and SVM with RBF kernel classifiers, compute the 30-fold crossvalidation **accuracies** and **precision** (use scoring='precision' when calling cross_val_score()) on **Data3**. Rank the classifiers based on accuracy and precision scores. Are the best classifiers ranked according to accuracy and precision the same? If not, explain the reason.

For the classifiers, feel free to choose any parameter settings you prefer.

[39]:
```
#Decision Trees
dt30 = DecisionTreeClassifier(max_depth=3)
dt_scores30a = cross_val_score(dt30, Data3_X, Data3_Y, cv=30,␣
↪scoring='accuracy')
```

```
dt_scores30p = cross_val_score(dt30, Data3_X, Data3_Y, cv=30,␣
 ↪scoring='precision')
print(" Accuraciy Matrix: \n", dt_scores30a, "\n\n Precision Matrix: \n",␣
 ↪dt_scores30p, "\n")
sum5a = 0
sum5p = 0
for i in range(len(dt_scores30a)):
    sum5a = sum5a + dt_scores30a[i]
    sum5p = sum5p + dt_scores30p[i]
```

```
 Accuraciy Matrix:
 [0.71428571 1.         1.         0.92857143 1.         0.92857143
 1.         0.64285714 1.         1.         0.84615385 1.
 1.         0.92307692 0.92307692 0.92307692 0.84615385 1.
 0.76923077 0.84615385 0.92307692 0.76923077 1.         0.84615385
 0.92307692 0.92307692 0.84615385 0.84615385 1.         0.84615385]

 Precision Matrix:
 [0.66666667 1.         1.         0.875      1.         0.875
 1.         0.58333333 1.         1.         0.85714286 1.
 1.         0.875      0.875      0.875      0.77777778 1.
 1.         0.77777778 0.85714286 0.66666667 1.         0.75
 0.85714286 0.85714286 0.75       0.75       1.         0.75      ]
```

[40]:
```
#Naive Bayes
nb5 = GaussianNB()
nb_scores5a = cross_val_score(nb5, Data3_X, Data3_Y, cv=30, scoring='accuracy')
nb_scores5p = cross_val_score(nb5, Data3_X, Data3_Y, cv=30, scoring='precision')
print(" Accuraciy Matrix: \n", nb_scores5a, "\n\n Precision Matrix: \n",␣
 ↪nb_scores5p, "\n")
sumd5a = 0
sumd5p = 0
for i in range(len(nb_scores5a)):
    sumd5a = sumd5a + nb_scores5a[i]
    sumd5p = sumd5p + nb_scores5p[i]
```

```
 Accuraciy Matrix:
 [1.         1.         0.92857143 0.92857143 1.         1.
 1.         0.92857143 1.         0.92857143 1.         1.
 0.92307692 1.         1.         1.         1.         0.92307692
 1.         0.84615385 0.92307692 0.92307692 0.92307692 0.92307692
 0.92307692 0.92307692 1.         0.92307692 1.         0.92307692]

 Precision Matrix:
 [1.         1.         0.875      0.875      1.         1.
 1.         0.875      1.         0.875      1.         1.
```

```
0.875      1.         1.         1.         1.         0.875
1.         0.77777778 0.85714286 0.85714286 0.85714286 0.85714286
0.85714286 0.85714286 1.         0.85714286 1.         0.85714286]
```

[41]:
```
#knn
knn4 = KNeighborsClassifier(n_neighbors=50)
knn_scores50a = cross_val_score(knn4, Data3_X, Data3_Y, cv=30,␣
 ↪scoring='accuracy')
knn_scores50p = cross_val_score(knn4, Data3_X, Data3_Y, cv=30,␣
 ↪scoring='precision')
print(" Accuraciy Matrix: \n", knn_scores50a, "\n\n Precision Matrix: \n",␣
 ↪knn_scores50p, "\n")
sumk5a = 0
sumk5p = 0
for i in range(len(knn_scores50a)):
    sumk5a = sumk5a + knn_scores50a[i]
    sumk5p = sumk5p + knn_scores50p[i]
```

```
 Accuraciy Matrix:
 [0.78571429 1.         0.92857143 0.85714286 0.85714286 1.
 0.92857143 0.78571429 0.92857143 0.92857143 0.92307692 0.84615385
 0.84615385 0.92307692 0.92307692 1.         0.92307692 0.84615385
 0.92307692 0.76923077 0.92307692 0.69230769 0.92307692 0.92307692
 0.92307692 0.84615385 0.92307692 0.76923077 0.84615385 0.84615385]

 Precision Matrix:
 [0.7        1.         0.875      0.77777778 0.77777778 1.
 0.875      0.7        0.875      0.875      0.875      0.77777778
 0.77777778 0.875      0.875      1.         0.875      0.77777778
 0.875      0.7        0.85714286 0.6        0.85714286 0.85714286
 0.85714286 0.75       0.85714286 0.66666667 0.75       0.75      ]
```

[42]:
```
#Linear SVM
svm_linear5 = SVC(C=0.5, kernel='linear')
svm_linear_scores5a = cross_val_score(svm_linear5, Data3_X, Data3_Y, cv=30,␣
 ↪scoring='accuracy')
svm_linear_scores5a = cross_val_score(svm_linear5, Data3_X, Data3_Y, cv=30,␣
 ↪scoring='precision')
print(" Accuraciy Matrix: \n", svm_linear_scores5a, "\n\n Precision Matrix:␣
 ↪\n", svm_linear_scores5a, "\n")
svm5a = 0
svm5p = 0
for i in range(len(svm_linear_scores5a)):
    svm5a = svm5a + svm_linear_scores5a[i]
    svm5p = svm5p + svm_linear_scores5a[i]
```

```
Accuraciy Matrix:
 [0.53846154 0.63636364 0.5        0.58333333 0.58333333 0.58333333
 0.63636364 0.5        0.63636364 0.5        0.58333333 0.7
 0.58333333 0.58333333 0.7        0.7        0.63636364 0.58333333
 0.77777778 0.58333333 0.54545455 0.5        0.6        0.6
 0.6        0.6        0.46153846 0.54545455 0.5        0.54545455]

 Precision Matrix:
 [0.53846154 0.63636364 0.5        0.58333333 0.58333333 0.58333333
 0.63636364 0.5        0.63636364 0.5        0.58333333 0.7
 0.58333333 0.58333333 0.7        0.7        0.63636364 0.58333333
 0.77777778 0.58333333 0.54545455 0.5        0.6        0.6
 0.6        0.6        0.46153846 0.54545455 0.5        0.54545455]
```

[43]:
```
#SVM with Polynomial Kernal
svm_poly4 = SVC(C=0.5, kernel='poly',degree=2, gamma = 'auto')
svm_poly_scores4a = cross_val_score(svm_poly4, Data3_X, Data3_Y, cv=30,␣
 ↪scoring='accuracy')
svm_poly_scores4p = cross_val_score(svm_poly4, Data3_X, Data3_Y, cv=30,␣
 ↪scoring='precision')
print(" Accuraciy Matrix: \n", svm_poly_scores4a, "\n\n Precision Matrix: \n",␣
 ↪svm_poly_scores4p, "\n")
svmp4a = 0
svmp4p = 0
for i in range(len(svm_poly_scores4a)):
    svmp4a = svmp4a + svm_poly_scores4a[i]
    svmp4p = svmp4p + svm_poly_scores4p[i]
```

```
Accuraciy Matrix:
 [0.92857143 1.         0.64285714 0.71428571 0.71428571 0.85714286
 0.78571429 0.92857143 0.85714286 0.85714286 0.84615385 0.84615385
 0.92307692 0.76923077 0.84615385 0.84615385 0.76923077 0.76923077
 0.92307692 0.84615385 0.84615385 0.84615385 0.92307692 0.84615385
 0.76923077 0.92307692 1.         0.84615385 0.76923077 0.92307692]

 Precision Matrix:
 [0.875      1.         0.66666667 0.63636364 0.63636364 0.77777778
 0.7        0.875      0.77777778 0.85714286 0.77777778 0.77777778
 0.875      0.7        0.77777778 0.77777778 0.75       0.7
 0.875      0.77777778 0.83333333 0.75       0.85714286 0.75
 0.66666667 0.85714286 1.         0.83333333 0.66666667 0.85714286]
```

[44]:
```
#SVM with RBF Kernel
svm_rbf5 = SVC(C = 0.5, kernel='rbf', gamma=10)
```

```
svm_rbf_scoresc5a = cross_val_score(svm_rbf5, Data3_X, Data3_Y, cv=30,␣
 ↪scoring='accuracy')
svm_rbf_scoresc5p = cross_val_score(svm_rbf5, Data3_X, Data3_Y, cv=30,␣
 ↪scoring='precision')
print(" Accuraciy Matrix: \n", svm_rbf_scoresc5a, "\n\n Precision Matrix: \n",␣
 ↪svm_rbf_scoresc5p, "\n")
svmrc5a = 0
svmrc5b = 0
for i in range(len(svm_rbf_scoresc5a)):
    svmrc5a = svmrc5a + svm_rbf_scoresc5a[i]
    svmrc5b = svmrc5b + svm_rbf_scoresc5p[i]
```

```
Accuraciy Matrix:
[1.         1.         1.         0.92857143 1.         1.
1.         0.92857143 1.         1.         1.         1.
1.         1.         1.         1.         1.         1.
1.         0.92307692 0.92307692 0.92307692 0.92307692 0.92307692
0.92307692 0.92307692 1.         0.92307692 1.         0.92307692]

Precision Matrix:
[1.         1.         1.         0.875      1.         1.
1.         0.875      1.         1.         1.         1.
1.         1.         1.         1.         1.         1.
1.         0.875      0.85714286 0.85714286 0.85714286 0.85714286
0.85714286 0.85714286 1.         0.85714286 1.         0.85714286]
```

[45]:
```
print(" Mean Accuracies For Decision Trees: \n", (sum5a/len(dt_scores30a)), "\n␣
 ↪Mean Precision For Decision Trees: \n", (sum5p/len(dt_scores30a)), "\n")
print(" Mean Accuracies For Naive Bayes: \n", (sumd5a/len(nb_scores5a)), "\n␣
 ↪Mean Precision For Naive Bayes: \n", (sumd5p/len(nb_scores5a)), "\n")
print(" Mean Accuracies For kNN: \n", (sumk5a/len(knn_scores50a)), "\n Mean␣
 ↪Precision For kNN: \n", (sumk5p/len(knn_scores50a)), "\n")
print(" Mean Accuracies For Linear SVM: \n", (svm5a/len(svm_linear_scores5a)),␣
 ↪"\n Mean Precision For Linear SVM: \n", (svm5p/len(svm_linear_scores5a)),␣
 ↪"\n")
print(" Mean Accuracies For SVM with Polynomial Kernel with degree 2: \n",␣
 ↪(svmp4a/len(svm_poly_scores4a)), "\n Mean Precision For SVM with Polynomial␣
 ↪Kernel with degree 2: \n", (svmp4p/len(svm_poly_scores4a)), "\n")
print(" Mean Accuracies For SVM with RBF Kernel with gamma as 10: \n", (svmrc5a/
 ↪len(svm_rbf_scoresc5a)), "\n Mean Precision For SVM with RBF Kernel with␣
 ↪gamma as 10: \n", (svmrc5b/len(svm_rbf_scoresc5a)), "\n")
```

```
Mean Accuracies For Decision Trees:
0.9071428571428574
Mean Precision For Decision Trees:
0.8758597883597885
```

```
Mean Accuracies For Naive Bayes:
0.9597069597069599
Mean Precision For Naive Bayes:
0.9294973544973547

Mean Accuracies For kNN:
0.8846153846153849
Mean Precision For kNN:
0.8222089947089949

Mean Accuracies For Linear SVM:
0.5875420875420876
Mean Precision For Linear SVM:
0.5875420875420876

Mean Accuracies For SVM with Polynomial Kernel with degree 2:
0.8454212454212457
Mean Precision For SVM with Polynomial Kernel with degree 2:
0.7887469937469939

Mean Accuracies For SVM with RBF Kernel with gamma as 10:
0.9721611721611723
Mean Precision For SVM with RBF Kernel with gamma as 10:
0.9494047619047621
```

Answer: Ranking in decreasing order based on accuracy - RBF Kernel with gamma as 10, Naive Bayes, Decision Trees, kNN, SVM with polynomial kernel of degree 2, Linear SVM **Answer:** Ranking in decreasing order based on precision - RBF Kernel with gamma as 10, Naive Bayes, Decision Trees, kNN, SVM with polynomial kernel of degree 2, Linear SVM **Answer:** Yes, the best classifiers are ranked according to accuracy and the precision the same.

1.0.7 7. Ensemble Methods

Question 7a: **Bagging:** Create bagging classifiers each with n_estimators = 1,2,3,4,5,10, and 20. Use a **linear SVM** (with C=0.5) as a base classifier. Using **Data3**, compute the mean **5-fold** cross validation accuracies and standard deviation for each of the bagging classifiers. State your observations on how bagging affected the mean and standard deviation of the base classifier. Explain your reason for what may have lead to these observations.

[46]:
```python
svm_linear6 = SVC(C=0.5, kernel='linear')
n_est_list = [1,2,3,4,5,10,20]
for n_est in n_est_list:
    # create an instance of bagging classifier with 'n_est' estimators
    bagging = BaggingClassifier(base_estimator=svm_linear6, n_estimators=n_est)
    # compute cross-validation accuracy for each bagging classifier
    scores = cross_val_score(bagging, Data3_X, Data3_Y, cv=5,␣
↪scoring='accuracy')
    print("Bagging Accuracy: %.2f (+/- %.2f) #estimators: %d" % (scores.mean(),␣
↪scores.std(), n_est))
```

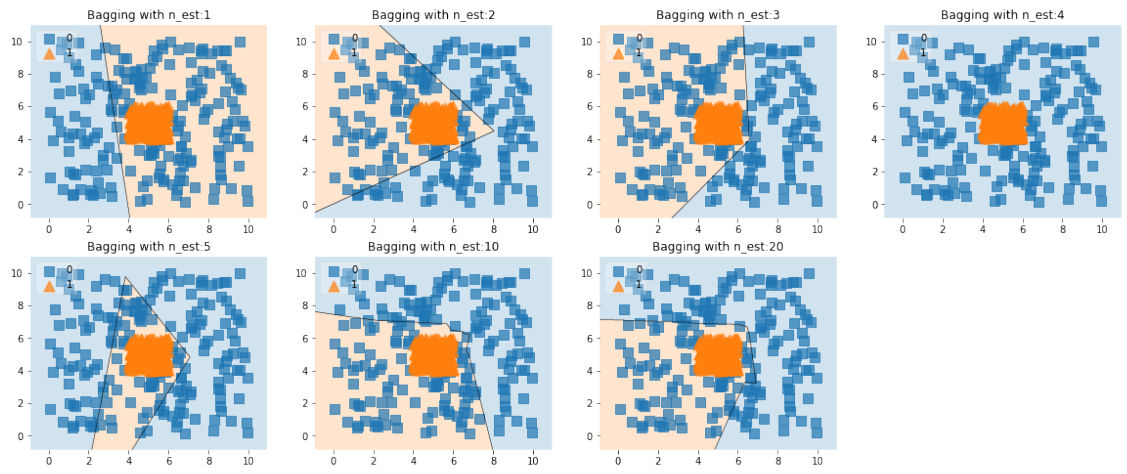
```
Bagging Accuracy: 0.58 (+/- 0.07) #estimators: 1
Bagging Accuracy: 0.57 (+/- 0.09) #estimators: 2
Bagging Accuracy: 0.64 (+/- 0.04) #estimators: 3
Bagging Accuracy: 0.59 (+/- 0.08) #estimators: 4
Bagging Accuracy: 0.62 (+/- 0.04) #estimators: 5
Bagging Accuracy: 0.74 (+/- 0.11) #estimators: 10
Bagging Accuracy: 0.68 (+/- 0.06) #estimators: 20
```

Answer: Bagging is a bootstrap aggregation method to improve the accuracy of the model. It merges the output of weak learners to generate a more stronger model. It helps to reduce the variance and avoid over-fitting. Hence, increase in the number of estimators increases the accuracy for our dataset.

Question 7b: Plot decision regions for the above bagging classifiers.

[47]:
```
fig = plt.figure(figsize=(20, 8))
count = 0;
for n_est in n_est_list:
    count = count + 1;
    bagging = BaggingClassifier(base_estimator=svm_linear6, n_estimators=n_est)
    bagging.fit(Data3_X, Data3_Y)
    ax = plt.subplot(2,4,count)
    fig = plot_decision_regions(X=Data3_X, y=Data3_Y, clf=bagging, legend=2,
                        scatter_kwargs=scatter_kwargs,
                        contourf_kwargs=contourf_kwargs,
                        scatter_highlight_kwargs=scatter_highlight_kwargs)
    plt.title('Bagging with n_est:'+str(n_est))

plt.show()
```

```
/users/PES0801/sharmahl/.local/lib/python3.6/site-
packages/mlxtend/plotting/decision_regions.py:246: UserWarning: No contour
levels were found within the data range.
  antialiased=True)
```



Question 7c: Comment on the quality of the decision regions for a bagging classifiers with many estimators when compared to that of only one estimator.

Answer: The quality of the decision boundry increases because as the estimator count is increased then mis-classified data points are given more weightage and it improves the quality of the boundry by adjusting the boundry for the misclassified points as they are considered for next estimator. But, as the estimators count increased the model tries to give more and more weightage to misclassified data and this increased the variance in the data which is causing over-fitting, hence the low accuracy for n_est = 5, 10, 20.

Question 7d: **Boosting:** Create boosting classifiers each with n_estimators = 1,2,3,4,5,10, 20, and 40. Use a **Decision Tree** (with max_depth=2) as a base classifier. Using **Data2**, compute the mean **10-fold** cross validation accuracies and standard deviation for each of the bagging classifiers. State your observations on how boosting affected the mean and standard deviation of the base classifier.

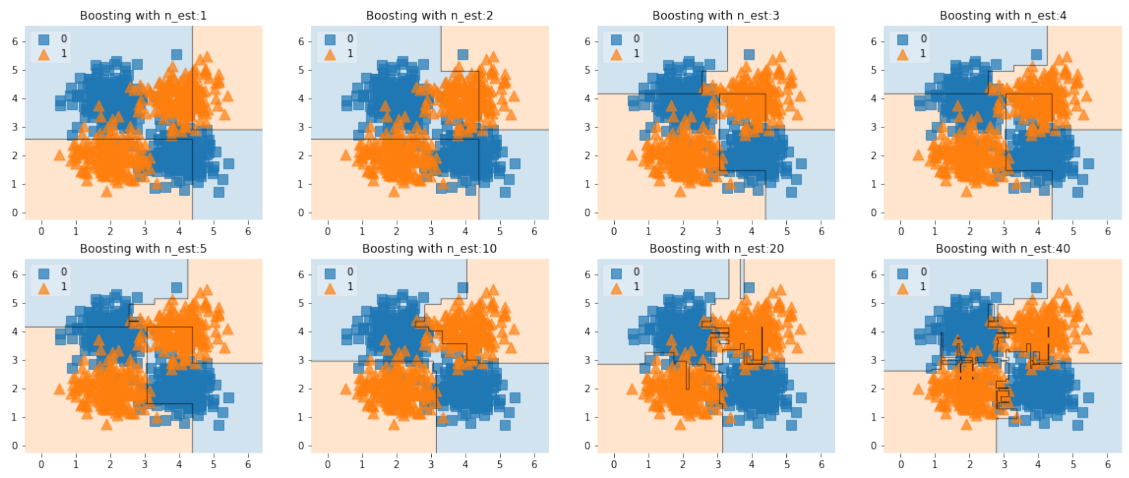
```
[48]: dtn = DecisionTreeClassifier(max_depth=2)
n_est_list = [1,2,3,4,5,10,20,40]
for n_est in n_est_list:
    # create an instance of a boosting classifier with 'n_est' estimators
    boosting = AdaBoostClassifier(base_estimator=dtn, n_estimators=n_est)
    # compute cross-validation accuracy for each bagging classifier
    scores = cross_val_score(boosting, Data2_X, Data2_Y, cv=10,␣
 ↪scoring='accuracy')
    print("Boosting Accuracy: %.2f (+/- %.2f) #estimators: %d" % (scores.
 ↪mean(), scores.std(), n_est))
```

```
Boosting Accuracy: 0.88 (+/- 0.03) #estimators: 1
Boosting Accuracy: 0.88 (+/- 0.03) #estimators: 2
Boosting Accuracy: 0.90 (+/- 0.04) #estimators: 3
Boosting Accuracy: 0.90 (+/- 0.04) #estimators: 4
Boosting Accuracy: 0.92 (+/- 0.03) #estimators: 5
Boosting Accuracy: 0.92 (+/- 0.04) #estimators: 10
Boosting Accuracy: 0.91 (+/- 0.03) #estimators: 20
Boosting Accuracy: 0.91 (+/- 0.03) #estimators: 40
```

Answer: Boosting helped to increase the accuracy and decrease the standard deviation till number of estimators = 10 and then the accuracy started decreasing and the standard deviation increased for same accuracy as the estimator count increased.

Question 7e: Plot decision regions for above boosting classifiers. Explain your reason for what may have lead to the observations in **Question 7d**.

[49]:
```
fig = plt.figure(figsize=(20, 8))
count = 0;
for n_est in n_est_list:
    count = count + 1;
    boosting = AdaBoostClassifier(base_estimator=dtn, n_estimators=n_est)
    boosting.fit(Data2_X, Data2_Y)
    ax = plt.subplot(2,4,count)
    fig = plot_decision_regions(X=Data2_X, y=Data2_Y, clf=boosting, legend=2,
                          scatter_kwargs=scatter_kwargs,
                          contourf_kwargs=contourf_kwargs,
                          scatter_highlight_kwargs=scatter_highlight_kwargs)
    plt.title('Boosting with n_est:'+str(n_est))

plt.show()
```



Answer: The above plots clearly explains as the estimators count increased the model tried to give more and more weightage to misclassified data and this increased the variance in the data and caused over-fitting and resulted in low accuracy.

1.0.8 8. Classification on a real-world dataset

Real world datasets typically have many attributes making it hard to visualize. This question is about using SVM and Decision Tree algorithms on a real world ‘breast cancer’ dataset.

The following code reads the dataset from the 'datasets' library in sklearn.

[32]:
```
from sklearn import datasets
cancer = datasets.load_breast_cancer()
```

The features are:

[33]:
```
cancer.feature_names
```

[33]:
```
array(['mean radius', 'mean texture', 'mean perimeter', 'mean area',
       'mean smoothness', 'mean compactness', 'mean concavity',
       'mean concave points', 'mean symmetry', 'mean fractal dimension',
       'radius error', 'texture error', 'perimeter error', 'area error',
       'smoothness error', 'compactness error', 'concavity error',
       'concave points error', 'symmetry error',
       'fractal dimension error', 'worst radius', 'worst texture',
       'worst perimeter', 'worst area', 'worst smoothness',
       'worst compactness', 'worst concavity', 'worst concave points',
       'worst symmetry', 'worst fractal dimension'], dtype='<U23')
```

Class labels are:

[34]:
```
cancer.target_names
```

[34]:
```
array(['malignant', 'benign'], dtype='<U9')
```

Create dataset for classification

[35]:
```
X = cancer.data
Y = cancer.target
```

Number of samples are:

[54]:
```
X.shape
```

[54]:
```
(569, 30)
```

Question 8a: Of all the SVM classifiers you explored in this hands-on exercise (i.e., linear SVM, SVM with a polynomial kernel and RBF kernel), which SVM results in a highest 10-fold cross-validation accuracy on this dataset? Explore the possible parameters for each SVM to determine the best performance for that SVM. For example, when studying linear SVM, explore a range of C values [0.001, 0.01, 0.1, 1]. Similarly for degree consider [1,2]. For gamma, consider [0.001, 0.01, 0.1, 1, 10, 100].

[36]:
```
#Linear SVM
svm_linearc1 = SVC(C=0.001, kernel='linear')
svm_linear_scoresc1 = cross_val_score(svm_linearc1, X, Y, cv=10,␣
 ↪scoring='accuracy')
svm_linearc2 = SVC(C=0.01, kernel='linear')
```

```
svm_linear_scoresc2 = cross_val_score(svm_linearc2, X, Y, cv=10,␣
 ↪scoring='accuracy')
svm_linearc3 = SVC(C=0.1, kernel='linear')
svm_linear_scoresc3 = cross_val_score(svm_linearc3, X, Y, cv=10,␣
 ↪scoring='accuracy')
svm_linearc4 = SVC(C=1, kernel='linear')
svm_linear_scoresc4 = cross_val_score(svm_linearc4, X, Y, cv=10,␣
 ↪scoring='accuracy')
svmc1 = 0
svmc2 = 0
svmc3 = 0
svmc4 = 0
for i in range(len(svm_linear_scoresc1)):
    svmc1 = svmc1 + svm_linear_scoresc1[i]
    svmc2 = svmc2 + svm_linear_scoresc2[i]
    svmc3 = svmc3 + svm_linear_scoresc3[i]
    svmc4 = svmc4 + svm_linear_scoresc4[i]
print("Mean Accuracy for Linear SVM with C=0.001: ", svmc1/
 ↪len(svm_linear_scoresc1), "\n")
print("Mean Accuracy for Linear SVM with C=0.01: ", svmc2/
 ↪len(svm_linear_scoresc2), "\n")
print("Mean Accuracy for Linear SVM with C=0.1: ", svmc3/
 ↪len(svm_linear_scoresc3), "\n")
print("Mean Accuracy for Linear SVM with C=1: ", svmc4/
 ↪len(svm_linear_scoresc4), "\n")
```

```
Mean Accuracy for Linear SVM with C=0.001:  0.9402568922305763

Mean Accuracy for Linear SVM with C=0.01:  0.9472744360902254

Mean Accuracy for Linear SVM with C=0.1:  0.9472744360902257

Mean Accuracy for Linear SVM with C=1:  0.9543233082706767
```

[37]:
```
#Polynomial varying C and degree
c_val_list = [0.001,0.01,0.1,1]
degree_list = [1, 2]
for c_val in c_val_list:
    for degree in degree_list:
        svm_rbf = SVC(C=c_val, kernel='poly',degree=degree, gamma = 'auto')
        # compute cross-validation accuracy for each Polynomial SVM classifier
        scores = cross_val_score(svm_rbf, X, Y, cv=10, scoring='accuracy')
        print("Poly SVM Accuracy: %.2f (+/- %.4f) #c_val: %.3f #degree_val: %.
 ↪3f" % (scores.mean(), scores.std(), c_val, degree))
```

```
Poly SVM Accuracy: 0.93 (+/- 0.0297) #c_val: 0.001 #degree_val: 1.000
Poly SVM Accuracy: 0.95 (+/- 0.0253) #c_val: 0.001 #degree_val: 2.000
Poly SVM Accuracy: 0.94 (+/- 0.0224) #c_val: 0.010 #degree_val: 1.000
Poly SVM Accuracy: 0.96 (+/- 0.0251) #c_val: 0.010 #degree_val: 2.000
Poly SVM Accuracy: 0.95 (+/- 0.0248) #c_val: 0.100 #degree_val: 1.000
Poly SVM Accuracy: 0.96 (+/- 0.0272) #c_val: 0.100 #degree_val: 2.000
Poly SVM Accuracy: 0.95 (+/- 0.0222) #c_val: 1.000 #degree_val: 1.000
Poly SVM Accuracy: 0.96 (+/- 0.0238) #c_val: 1.000 #degree_val: 2.000
```

[38]:
```
#RBF varying C and gamma
c_val_list = [0.001,0.01,0.1,1]
gamma_val_list = [0.001,0.01,0.1,1,10,100]
for c_val in c_val_list:
    for gamma_val in gamma_val_list:
        svm_rbf = SVC(C=c_val,kernel='rbf',gamma=gamma_val)
        # compute cross-validation accuracy for each SVM classifier
        scores = cross_val_score(svm_rbf, X, Y, cv=10, scoring='accuracy')
        print("RBF SVM Accuracy: %.2f (+/- %.2f) #c_val: %.3f #gamma_val: %.3f"␣
 ↪% (scores.mean(), scores.std(), c_val,gamma_val))
```

```
RBF SVM Accuracy: 0.63 (+/- 0.01) #c_val: 0.001 #gamma_val: 0.001
RBF SVM Accuracy: 0.63 (+/- 0.01) #c_val: 0.001 #gamma_val: 0.010
RBF SVM Accuracy: 0.63 (+/- 0.01) #c_val: 0.001 #gamma_val: 0.100
RBF SVM Accuracy: 0.63 (+/- 0.01) #c_val: 0.001 #gamma_val: 1.000
RBF SVM Accuracy: 0.63 (+/- 0.01) #c_val: 0.001 #gamma_val: 10.000
RBF SVM Accuracy: 0.63 (+/- 0.01) #c_val: 0.001 #gamma_val: 100.000
RBF SVM Accuracy: 0.63 (+/- 0.01) #c_val: 0.010 #gamma_val: 0.001
RBF SVM Accuracy: 0.63 (+/- 0.01) #c_val: 0.010 #gamma_val: 0.010
RBF SVM Accuracy: 0.63 (+/- 0.01) #c_val: 0.010 #gamma_val: 0.100
RBF SVM Accuracy: 0.63 (+/- 0.01) #c_val: 0.010 #gamma_val: 1.000
RBF SVM Accuracy: 0.63 (+/- 0.01) #c_val: 0.010 #gamma_val: 10.000
RBF SVM Accuracy: 0.63 (+/- 0.01) #c_val: 0.010 #gamma_val: 100.000
RBF SVM Accuracy: 0.63 (+/- 0.01) #c_val: 0.100 #gamma_val: 0.001
RBF SVM Accuracy: 0.63 (+/- 0.01) #c_val: 0.100 #gamma_val: 0.010
RBF SVM Accuracy: 0.63 (+/- 0.01) #c_val: 0.100 #gamma_val: 0.100
RBF SVM Accuracy: 0.63 (+/- 0.01) #c_val: 0.100 #gamma_val: 1.000
RBF SVM Accuracy: 0.63 (+/- 0.01) #c_val: 0.100 #gamma_val: 10.000
RBF SVM Accuracy: 0.63 (+/- 0.01) #c_val: 0.100 #gamma_val: 100.000
RBF SVM Accuracy: 0.92 (+/- 0.02) #c_val: 1.000 #gamma_val: 0.001
RBF SVM Accuracy: 0.63 (+/- 0.01) #c_val: 1.000 #gamma_val: 0.010
RBF SVM Accuracy: 0.63 (+/- 0.01) #c_val: 1.000 #gamma_val: 0.100
RBF SVM Accuracy: 0.63 (+/- 0.01) #c_val: 1.000 #gamma_val: 1.000
RBF SVM Accuracy: 0.63 (+/- 0.01) #c_val: 1.000 #gamma_val: 10.000
RBF SVM Accuracy: 0.63 (+/- 0.01) #c_val: 1.000 #gamma_val: 100.000
```

Answer: Polynomial SVM with Cvalue as 2 results in highest accuracy i.e. 96%.

Question 8b: Similar to **Question 8a** explore decision trees with different max_depth to determine which values returns the best classifier.

[40]:
```
dep_list = [2,4,6,8,10,50]
for dep in dep_list:
    dt = DecisionTreeClassifier(max_depth = dep)
    scores = cross_val_score(dt, X, Y, cv=10, scoring='accuracy')
    print("Decision Tree Accuracy: %.4f (+/- %.2f) #Max Depth: %d" % (scores.
 ↪mean(), scores.std(), dep))
```

```
Decision Tree Accuracy: 0.9210 (+/- 0.03) #Max Depth: 2
Decision Tree Accuracy: 0.9192 (+/- 0.04) #Max Depth: 4
Decision Tree Accuracy: 0.9263 (+/- 0.04) #Max Depth: 6
Decision Tree Accuracy: 0.9157 (+/- 0.04) #Max Depth: 8
Decision Tree Accuracy: 0.9104 (+/- 0.03) #Max Depth: 10
Decision Tree Accuracy: 0.9122 (+/- 0.03) #Max Depth: 50
```

Answer: Decision Tree with max depth as 2 results in highest accuracy for the give dataset.

Question 8c: Imagine a scenario where you are working at a cancer center as a data scientist tasked with identifying the characteristics that distinguish malignant tumors from benign tumors. Based on your knowledge of classification techniques which approach would you use and why?

Answer: We know that SVM gives better perfomance for high dimensional dataset and its effective when the number of attributes or dimensions are higher than the number of data points. It uses a subset of training points called support vectors, so it is also memory efficient. Also, we have an advantage of using different Kernel functions such as linear, polynomial and rbf for the same algorithm. So, I would like to use Support Vector Machines for the above mentioned screnario to classify the malignant and benign tumors.