Hands-on Exercise CLASS Module

In [4]:
```
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
```

In [2]:
```
#Mlxtend for visualizing classification decision boundaries
!python -m pip install --user --upgrade pip
```

```
Collecting pip
  Downloading https://files.pythonhosted.org/packages/00/b6/9cfa56b4081ad1387
4b0c6f96af8ce16cfbc1cb06bedf8e9164ce5551ec1/pip-19.3.1-py2.py3-none-any.whl
(1.4MB)
    100% |████████████████████████████████| 1.4MB 92kB/s  eta 0:00:01
Installing collected packages: pip
Successfully installed pip-19.3.1
```

In [3]:
```
!python -m pip install --user mlxtend
```

```
Requirement already satisfied: mlxtend in ./.local/lib/python3.6/site-package
s (0.17.0)
Requirement already satisfied: matplotlib>=3.0.0 in ./.local/lib/python3.6/si
te-packages (from mlxtend) (3.1.1)
Requirement already satisfied: setuptools in /usr/local/anaconda5/lib/python
3.6/site-packages (from mlxtend) (36.5.0.post20170921)
Requirement already satisfied: scikit-learn>=0.20.3 in ./.local/lib/python3.
6/site-packages (from mlxtend) (0.21.3)
Requirement already satisfied: scipy>=1.2.1 in ./.local/lib/python3.6/site-pa
ckages (from mlxtend) (1.3.1)
Requirement already satisfied: joblib>=0.13.2 in ./.local/lib/python3.6/site-
packages (from mlxtend) (0.14.0)
Requirement already satisfied: numpy>=1.16.2 in ./.local/lib/python3.6/site-p
ackages (from mlxtend) (1.17.3)
Requirement already satisfied: pandas>=0.24.2 in ./.local/lib/python3.6/site-
packages (from mlxtend) (0.25.3)
Requirement already satisfied: kiwisolver>=1.0.1 in ./.local/lib/python3.6/si
te-packages (from matplotlib>=3.0.0->mlxtend) (1.1.0)
Requirement already satisfied: cycler>=0.10 in /usr/local/anaconda5/lib/pytho
n3.6/site-packages (from matplotlib>=3.0.0->mlxtend) (0.10.0)
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/anaconda5/l
ib/python3.6/site-packages (from matplotlib>=3.0.0->mlxtend) (2.6.1)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /u
sr/local/anaconda5/lib/python3.6/site-packages (from matplotlib>=3.0.0->mlxte
nd) (2.2.0)
Requirement already satisfied: pytz>=2017.2 in /usr/local/anaconda5/lib/pytho
n3.6/site-packages (from pandas>=0.24.2->mlxtend) (2017.2)
Requirement already satisfied: six in /usr/local/anaconda5/lib/python3.6/site
-packages (from cycler>=0.10->matplotlib>=3.0.0->mlxtend) (1.11.0)
```

In [5]:
```
# 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],Data
4_X[:,1]) - 100 < 0 ] = 0
```

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.

In [12]:
```
dt_2 = DecisionTreeClassifier(max_depth=2)
dt_2_scores = cross_val_score(dt_2, Data3_X, Data3_Y, cv=10, scoring='accurac
y')
print (dt_2_scores)
```

```
[0.8   0.9   0.9   0.875 0.925 0.875 0.9   0.85  0.85  0.875]
```

In [19]:
```
[dt_2_scores.mean(), dt_2_scores.std()]
```

Out[19]: [0.875, 0.03354101966249685]

In [13]:
```
dt_4 = DecisionTreeClassifier(max_depth=4)
dt_4_scores = cross_val_score(dt_4, Data3_X, Data3_Y, cv=10, scoring='accurac
y')
print (dt_4_scores)
```

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

In [20]:
```
[dt_4_scores.mean(), dt_4_scores.std()]
```

Out[20]: [0.97, 0.021794494717703363]

In [14]:
```
dt_6 = DecisionTreeClassifier(max_depth=6)
dt_6_scores = cross_val_score(dt_6, Data3_X, Data3_Y, cv=10, scoring='accurac
y')
print (dt_6_scores)
```

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

In [21]:
```
[dt_6_scores.mean(), dt_6_scores.std()]
```

Out[21]: [0.9674999999999999, 0.0225]

In [15]:
```
dt_8 = DecisionTreeClassifier(max_depth=8)
dt_8_scores = cross_val_score(dt_8, Data3_X, Data3_Y, cv=10, scoring='accurac
y')
print (dt_8_scores)
```

```
[0.95  0.975 0.95  0.95  1.    0.975 0.9   0.875 0.975 0.975]
```

In [22]:
```
[dt_8_scores.mean(), dt_8_scores.std()]
```

Out[22]: [0.9524999999999999, 0.03614208073700239]

In [16]:
```
dt_10 = DecisionTreeClassifier(max_depth=10)
dt_10_scores = cross_val_score(dt_10, Data3_X, Data3_Y, cv=10, scoring='accura
cy')
print (dt_10_scores)
```

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

In [23]:
```
[dt_10_scores.mean(), dt_10_scores.std()]
```

Out[23]: [0.95, 0.024999999999999977]

In [17]:
```
dt_50 = DecisionTreeClassifier(max_depth=2)
dt_50_scores = cross_val_score(dt_50, Data3_X, Data3_Y, cv=10, scoring='accura
cy')
print (dt_50_scores)
```

```
[0.8   0.9   0.9   0.875 0.925 0.875 0.9   0.85  0.85  0.9  ]
```

In [24]:
```
[dt_50_scores.mean(), dt_50_scores.std()]
```

Out[24]:
```
[0.8775000000000001, 0.034369317712168806]
```

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

Answer: Looking at the mean values, it can be observed that the accuracy is low when max_depth is either 2 (small) or 50 (large).
Choosing a large max_depth value can cause the problem of over-fitting and small value of max_depth can make the model a little too flexible and generic making it under-fitting

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: The mean accuracy for max_depth = 50 is around 87.75% while the highest accuracy of 97% is observed when max_depth is 4.
Difference= 9.25
This phenomenon is called over-fitting problem

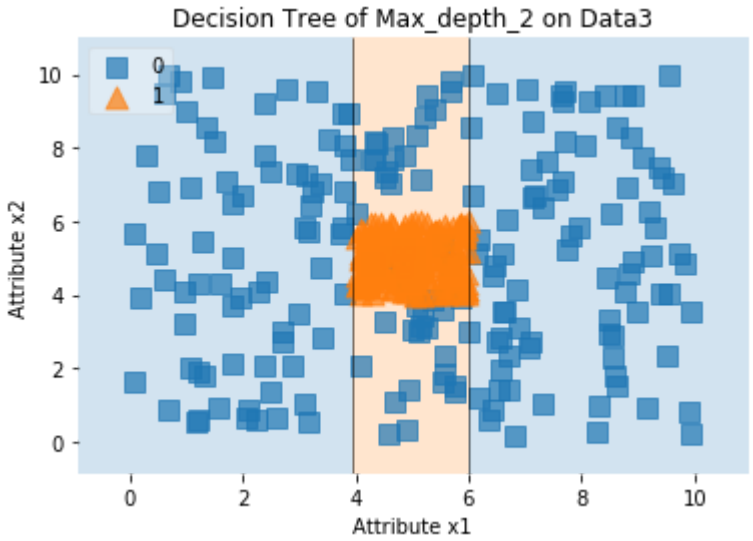
Question 1d: Plot decision regions for the above decision tree models

In [6]:
```
# Training a classifier
DT_2=DecisionTreeClassifier(max_depth=2)
DT_2.fit(Data3_X, Data3_Y)

# parameters to set size or markers, contours, and transparency
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}

# passing above parameters to the plotting function
plot_decision_regions(X=Data3_X, y=Data3_Y, clf=DT_2, legend=2,
                      scatter_kwargs=scatter_kwargs,
                      contourf_kwargs=contourf_kwargs,
                      scatter_highlight_kwargs=scatter_highlight_kwargs)

# Adding axes annotations
plt.xlabel('Attribute x1')
plt.ylabel('Attribute x2')
plt.title('Decision Tree of Max_depth_2 on Data3')
plt.show()
```

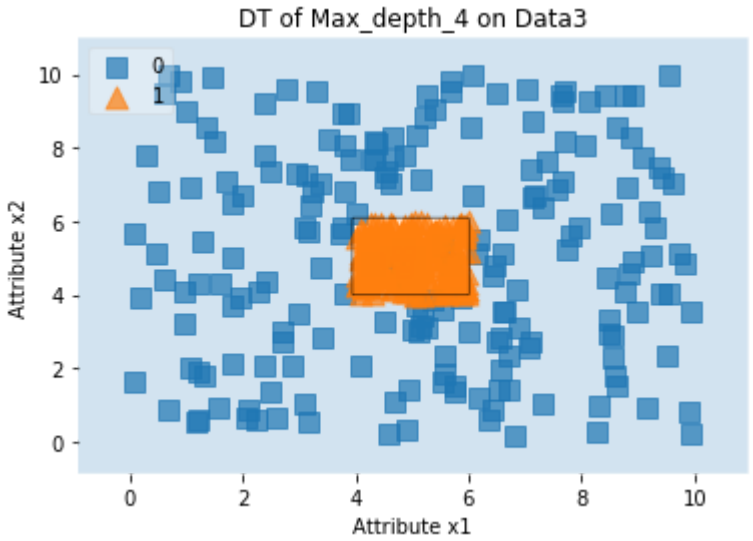


In [5]:
```
# Training a classifier
DT_4=DecisionTreeClassifier(max_depth=4)
DT_4.fit(Data3_X, Data3_Y)

# parameters to set size or markers, contours, and transparency
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}

# passing above parameters to the plotting function
plot_decision_regions(X=Data3_X, y=Data3_Y, clf=DT_4, legend=2,
                      scatter_kwargs=scatter_kwargs,
                      contourf_kwargs=contourf_kwargs,
                      scatter_highlight_kwargs=scatter_highlight_kwargs)

# Adding axes annotations
plt.xlabel('Attribute x1')
plt.ylabel('Attribute x2')
plt.title('Decision Tree of Max_depth_4 on Data3')
plt.show()
```

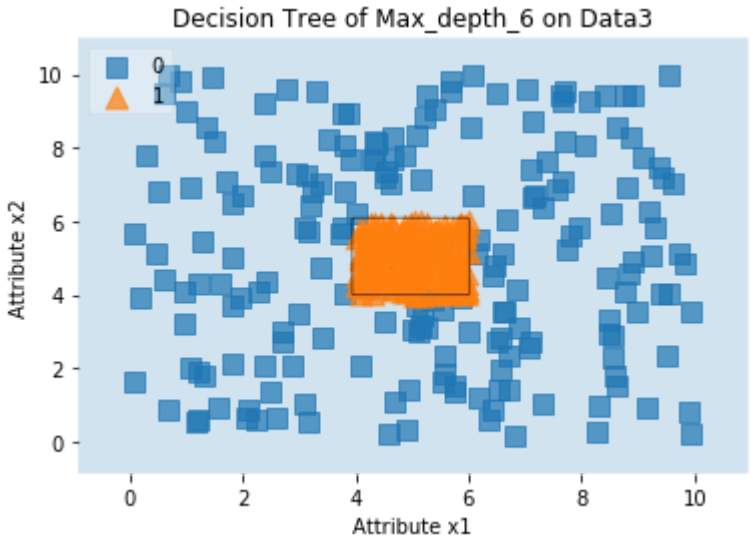


In [7]:
```
# Training a classifier
DT_6=DecisionTreeClassifier(max_depth=6)
DT_6.fit(Data3_X, Data3_Y)

# parameters to set size or markers, contours, and transparency
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}

# passing above parameters to the plotting function
plot_decision_regions(X=Data3_X, y=Data3_Y, clf=DT_6, legend=2,
                      scatter_kwargs=scatter_kwargs,
                      contourf_kwargs=contourf_kwargs,
                      scatter_highlight_kwargs=scatter_highlight_kwargs)

# Adding axes annotations
plt.xlabel('Attribute x1')
plt.ylabel('Attribute x2')
plt.title('Decision Tree of Max_depth_6 on Data3')
plt.show()
```

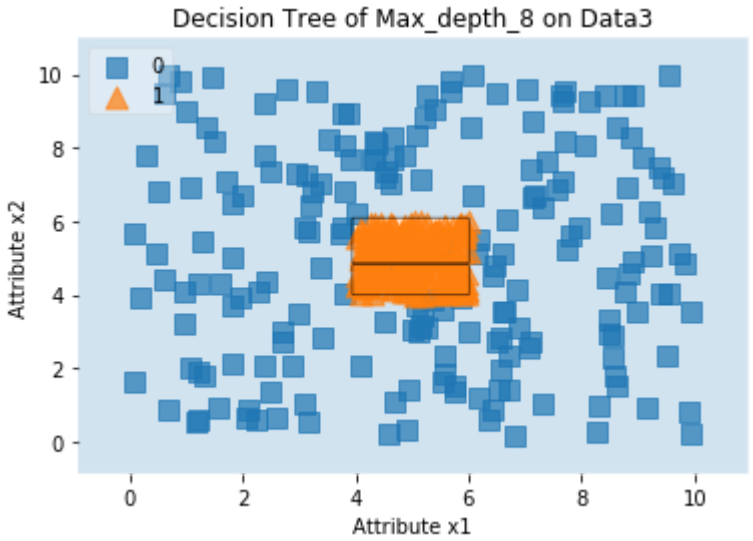


In [8]:
```
# Training a classifier
DT_8=DecisionTreeClassifier(max_depth=8)
DT_8.fit(Data3_X, Data3_Y)

# parameters to set size or markers, contours, and transparency
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}

# passing above parameters to the plotting function
plot_decision_regions(X=Data3_X, y=Data3_Y, clf=DT_8, legend=2,
                      scatter_kwargs=scatter_kwargs,
                      contourf_kwargs=contourf_kwargs,
                      scatter_highlight_kwargs=scatter_highlight_kwargs)

# Adding axes annotations
plt.xlabel('Attribute x1')
plt.ylabel('Attribute x2')
plt.title('Decision Tree of Max_depth_8 on Data3')
plt.show()
```

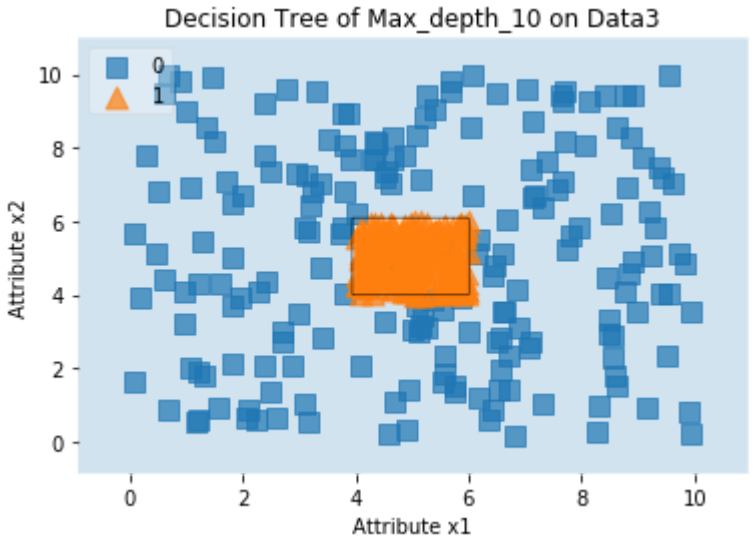


In [9]:
```
# Training a classifier
DT_10=DecisionTreeClassifier(max_depth=10)
DT_10.fit(Data3_X, Data3_Y)

# parameters to set size or markers, contours, and transparency
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}

# passing above parameters to the plotting function
plot_decision_regions(X=Data3_X, y=Data3_Y, clf=DT_10, legend=2,
                      scatter_kwargs=scatter_kwargs,
                      contourf_kwargs=contourf_kwargs,
                      scatter_highlight_kwargs=scatter_highlight_kwargs)

# Adding axes annotations
plt.xlabel('Attribute x1')
plt.ylabel('Attribute x2')
plt.title('Decision Tree of Max_depth_10 on Data3')
plt.show()
```

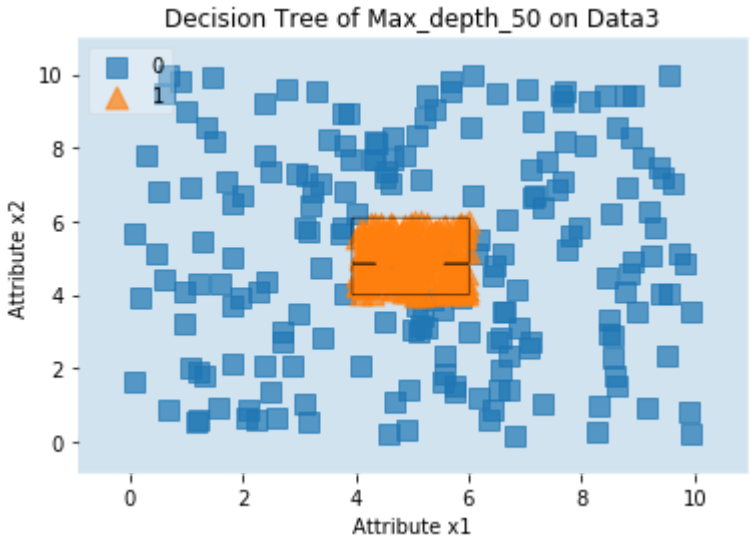


In [10]:
```
# Training a classifier
DT_50=DecisionTreeClassifier(max_depth=50)
DT_50.fit(Data3_X, Data3_Y)

# parameters to set size or markers, contours, and transparency
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}

# passing above parameters to the plotting function
plot_decision_regions(X=Data3_X, y=Data3_Y, clf=DT_50, legend=2,
                      scatter_kwargs=scatter_kwargs,
                      contourf_kwargs=contourf_kwargs,
                      scatter_highlight_kwargs=scatter_highlight_kwargs)

# Adding axes annotations
plt.xlabel('Attribute x1')
plt.ylabel('Attribute x2')
plt.title('Decision Tree of Max_depth_50 on Data3')
plt.show()
```



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

Answer: Analysing the decision boundaries and mean accuracy, it can observed that boundaries max depth 4 is better suited for this data

2. k Nearest Neighbor

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

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

In [12]:
```
knn_1 = KNeighborsClassifier(n_neighbors=1)
knn_1_scores = cross_val_score(knn_1, Data2_X, Data2_Y, cv=10, scoring='accura
cy')
print(knn_1_scores)
```

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

In [16]:
```
[knn_1_scores.mean(), knn_1_scores.std()]
```

Out[16]: [0.9125, 0.02091650066335192]

In [13]:
```
knn_5 = KNeighborsClassifier(n_neighbors=5)
knn_5_scores = cross_val_score(knn_5, Data2_X, Data2_Y, cv=10, scoring='accura
cy')
print(knn_5_scores)
```

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

In [17]:
```
[knn_5_scores.mean(), knn_5_scores.std()]
```

Out[17]: [0.9349999999999999, 0.0325]

In [14]:
```
knn_10 = KNeighborsClassifier(n_neighbors=10)
knn_10_scores = cross_val_score(knn_10, Data2_X, Data2_Y, cv=10, scoring='accu
racy')
print(knn_10_scores)
```

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

In [18]:
```
[knn_10_scores.mean(), knn_10_scores.std()]
```

Out[18]: [0.9400000000000001, 0.03436931771216879]

In [15]:
```
knn_50 = KNeighborsClassifier(n_neighbors=50)
knn_50_scores = cross_val_score(knn_50, Data2_X, Data2_Y, cv=10, scoring='accu
racy')
print(knn_50_scores)
```

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

In [19]:
```
[knn_50_scores.mean(), knn_50_scores.std()]
```

Out[19]: [0.9412499999999999, 0.030644126680328173]

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

Answer: Lowest accuracy is observed when n_neighbors value is 1. i.e we consider one nearest neighbor. This phenomenon is called Over-fitting as it has high error rate

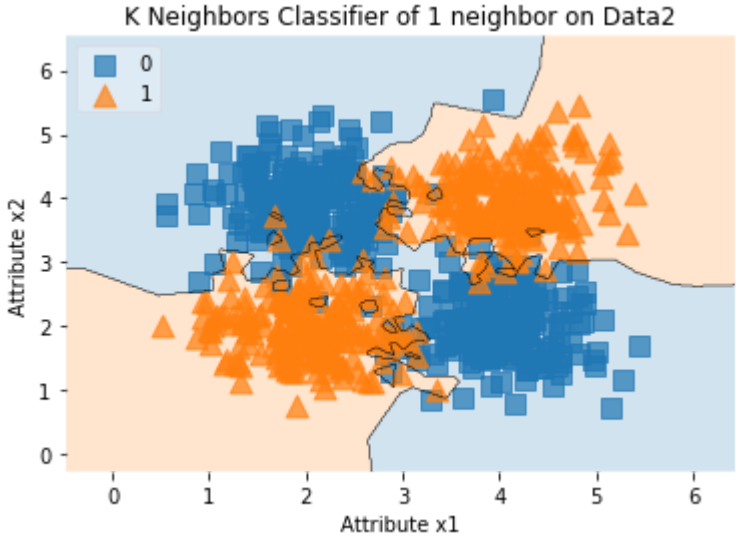
Question 2c: Plot decision regions for a kNN classifier with n_neighbors = 1, 5, 10, 50

In [21]:
```
# Training a classifier
knn_1 = KNeighborsClassifier(n_neighbors=1)
knn_1.fit(Data2_X, Data2_Y)

# parameters to set size or markers, contours, and transparency
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}

# passing above parameters to the plotting function
plot_decision_regions(X=Data2_X, y=Data2_Y, clf=knn_1, legend=2,
                      scatter_kwargs=scatter_kwargs,
                      contourf_kwargs=contourf_kwargs,
                      scatter_highlight_kwargs=scatter_highlight_kwargs)

# Adding axes annotations
plt.xlabel('Attribute x1')
plt.ylabel('Attribute x2')
plt.title('K Neighbors Classifier of 1 neighbor on Data2')
plt.show()
```

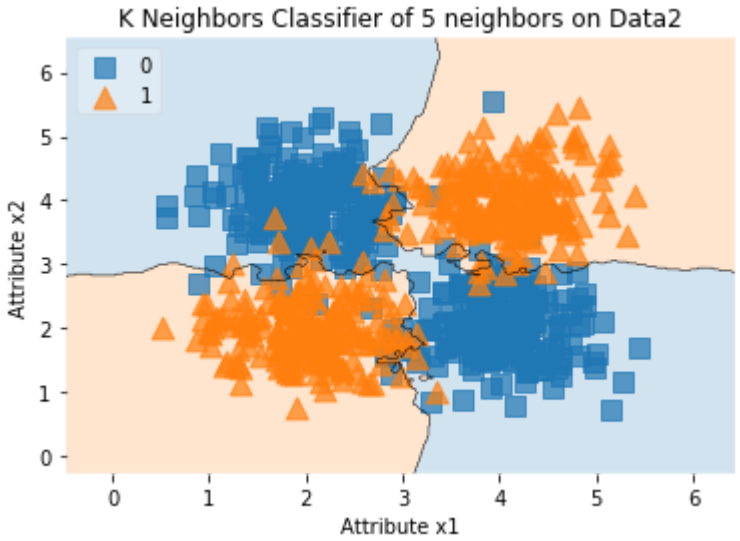


In [23]:
```
# Training a classifier
knn_5 = KNeighborsClassifier(n_neighbors=5)
knn_5.fit(Data2_X, Data2_Y)

# parameters to set size or markers, contours, and transparency
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}

# passing above parameters to the plotting function
plot_decision_regions(X=Data2_X, y=Data2_Y, clf=knn_5, legend=2,
                      scatter_kwargs=scatter_kwargs,
                      contourf_kwargs=contourf_kwargs,
                      scatter_highlight_kwargs=scatter_highlight_kwargs)

# Adding axes annotations
plt.xlabel('Attribute x1')
plt.ylabel('Attribute x2')
plt.title('K Neighbors Classifier of 5 neighbors on Data2')
plt.show()
```

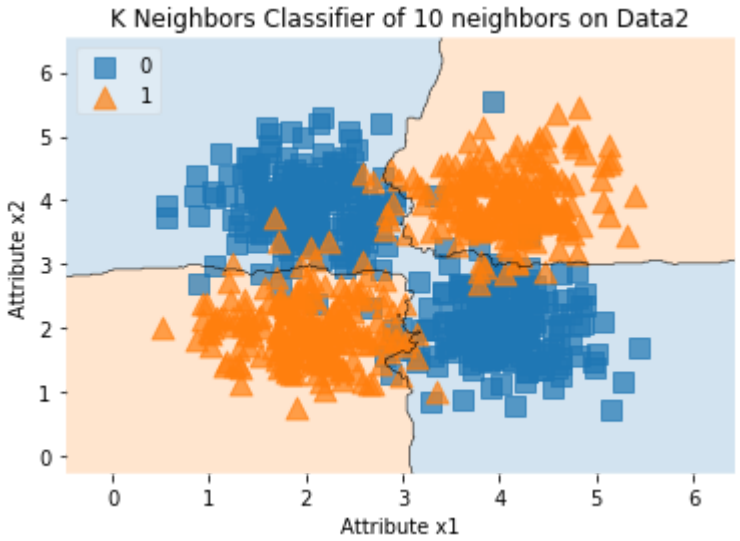


In [25]:
```
# Training a classifier
knn_10 = KNeighborsClassifier(n_neighbors=10)
knn_10.fit(Data2_X, Data2_Y)

# parameters to set size or markers, contours, and transparency
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}

# passing above parameters to the plotting function
plot_decision_regions(X=Data2_X, y=Data2_Y, clf=knn_10, legend=2,
                      scatter_kwargs=scatter_kwargs,
                      contourf_kwargs=contourf_kwargs,
                      scatter_highlight_kwargs=scatter_highlight_kwargs)

# Adding axes annotations
plt.xlabel('Attribute x1')
plt.ylabel('Attribute x2')
plt.title('K Neighbors Classifier of 10 neighbors on Data2')
plt.show()
```

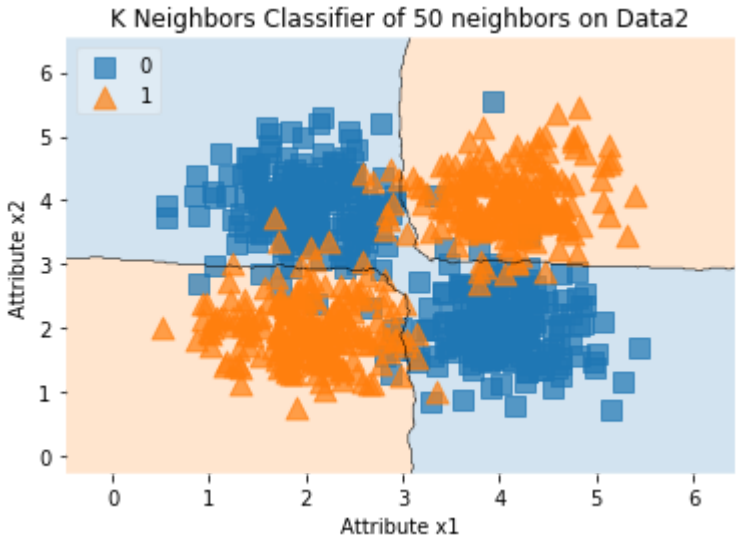


In [26]:
```
# Training a classifier
knn_50 = KNeighborsClassifier(n_neighbors=50)
knn_50.fit(Data2_X, Data2_Y)

# parameters to set size or markers, contours, and transparency
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}

# passing above parameters to the plotting function
plot_decision_regions(X=Data2_X, y=Data2_Y, clf=knn_50, legend=2,
                      scatter_kwargs=scatter_kwargs,
                      contourf_kwargs=contourf_kwargs,
                      scatter_highlight_kwargs=scatter_highlight_kwargs)

# Adding axes annotations
plt.xlabel('Attribute x1')
plt.ylabel('Attribute x2')
plt.title('K Neighbors Classifier of 50 neighbors on Data2')
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: With the increase in the number of n_neighbors, the decision boundary becomes smoother and well defined

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

In [33]:
```
nb_data1 = GaussianNB()
nb_data1_scores = cross_val_score(nb_data1, Data1_X, Data1_Y, cv=10, scoring=
'accuracy')
print (nb_data1_scores)
[nb_data1_scores.mean(),nb_data1_scores.std()]
```

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

Out[33]:
```
[0.9675, 0.03172144385112379]
```

In [34]:
```
nb_data2 = GaussianNB()
nb_data2_scores = cross_val_score(nb_data2, Data2_X, Data2_Y, cv=10, scoring=
'accuracy')
print (nb_data2_scores)
[nb_data2_scores.mean(),nb_data2_scores.std()]
```

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

Out[34]:
```
[0.049999999999999996, 0.026809513236909017]
```

In [35]:
```
nb_data3 = GaussianNB()
nb_data3_scores = cross_val_score(nb_data3, Data3_X, Data3_Y, cv=10, scoring=
'accuracy')
print (nb_data3_scores)
[nb_data3_scores.mean(),nb_data3_scores.std()]
```

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

Out[35]:
```
[0.96, 0.027838821814150098]
```

In [36]:
```
nb_data4 = GaussianNB()
nb_data4_scores = cross_val_score(nb_data4, Data4_X, Data4_Y, cv=10, scoring=
'accuracy')
print (nb_data4_scores)
[nb_data4_scores.mean(),nb_data4_scores.std()]
```

```
[0.90196078 1.         0.98       0.98       0.98       0.96
 0.94       0.96       0.97959184 0.95918367]
```

Out[36]:
```
[0.9640736294517807, 0.026052087140989725]
```

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

Answer: Naive Bayesian algorithm performed poorly on data 2 as it has lowest accuracy of 50% compared to other datasets.

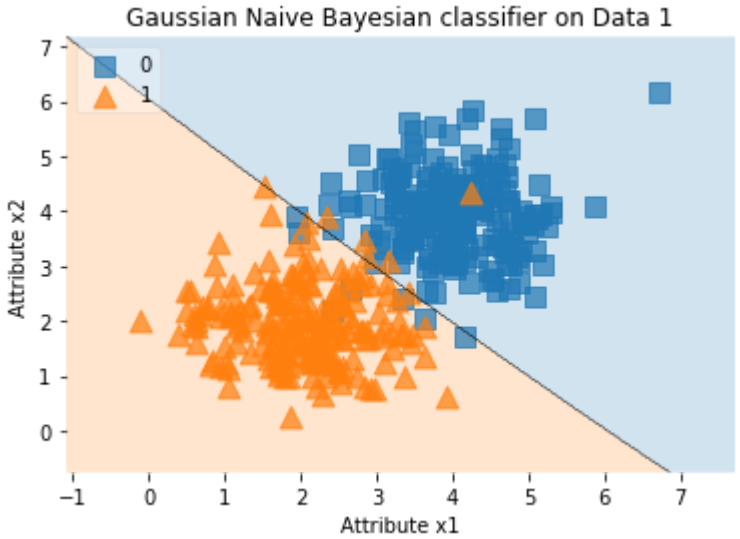
Question 3c: Plot decision regions for a NB classifier on each of the four datasets

In [38]:
```
# Training a classifier
nb_data1 = GaussianNB()
nb_data1.fit(Data1_X, Data1_Y)

# parameters to set size or markers, contours, and transparency
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}

# passing above parameters to the plotting function
plot_decision_regions(X=Data1_X, y=Data1_Y, clf=nb_data1, legend=2,
                      scatter_kwargs=scatter_kwargs,
                      contourf_kwargs=contourf_kwargs,
                      scatter_highlight_kwargs=scatter_highlight_kwargs)

# Adding axes annotations
plt.xlabel('Attribute x1')
plt.ylabel('Attribute x2')
plt.title('Gaussian Naive Bayesian classifier on Data 1')
plt.show()
```

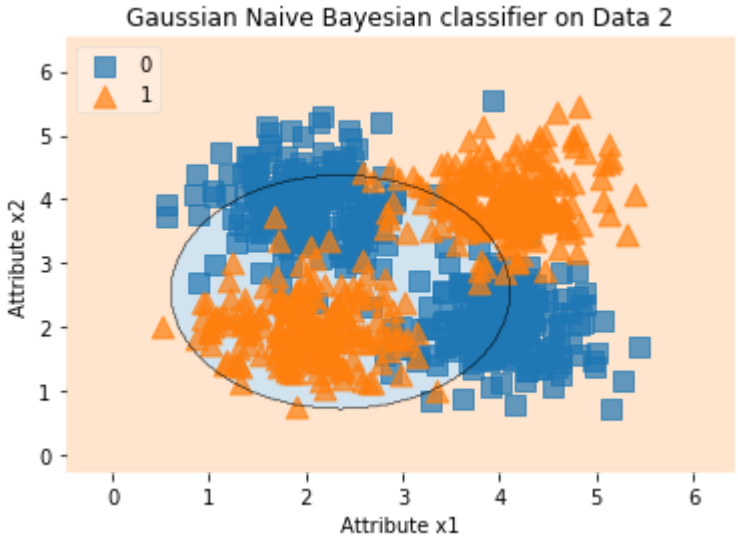


In [39]:
```
# Training a classifier
nb_data2 = GaussianNB()
nb_data2.fit(Data2_X, Data2_Y)

# parameters to set size or markers, contours, and transparency
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}

# passing above parameters to the plotting function
plot_decision_regions(X=Data2_X, y=Data2_Y, clf=nb_data2, legend=2,
                      scatter_kwargs=scatter_kwargs,
                      contourf_kwargs=contourf_kwargs,
                      scatter_highlight_kwargs=scatter_highlight_kwargs)

# Adding axes annotations
plt.xlabel('Attribute x1')
plt.ylabel('Attribute x2')
plt.title('Gaussian Naive Bayesian classifier on Data 2')
plt.show()
```

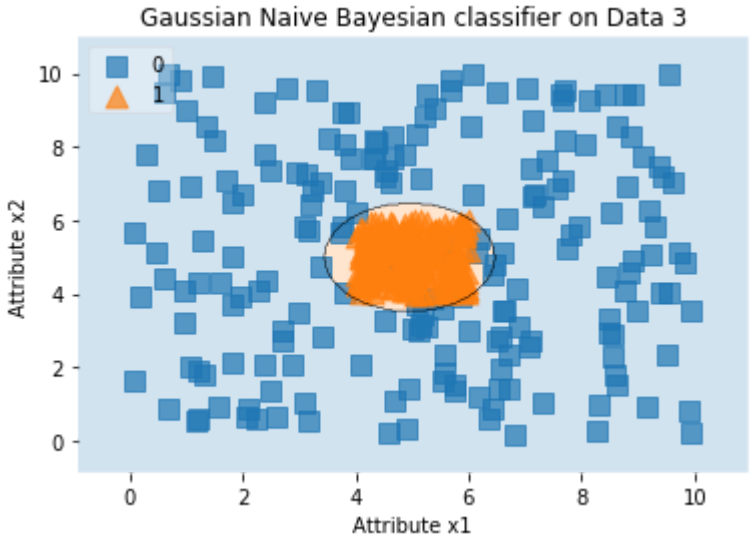


In [40]:
```
# Training a classifier
nb_data3 = GaussianNB()
nb_data3.fit(Data3_X, Data3_Y)

# parameters to set size or markers, contours, and transparency
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}

# passing above parameters to the plotting function
plot_decision_regions(X=Data3_X, y=Data3_Y, clf=nb_data3, legend=2,
                      scatter_kwargs=scatter_kwargs,
                      contourf_kwargs=contourf_kwargs,
                      scatter_highlight_kwargs=scatter_highlight_kwargs)

# Adding axes annotations
plt.xlabel('Attribute x1')
plt.ylabel('Attribute x2')
plt.title('Gaussian Naive Bayesian classifier on Data 3')
plt.show()
```

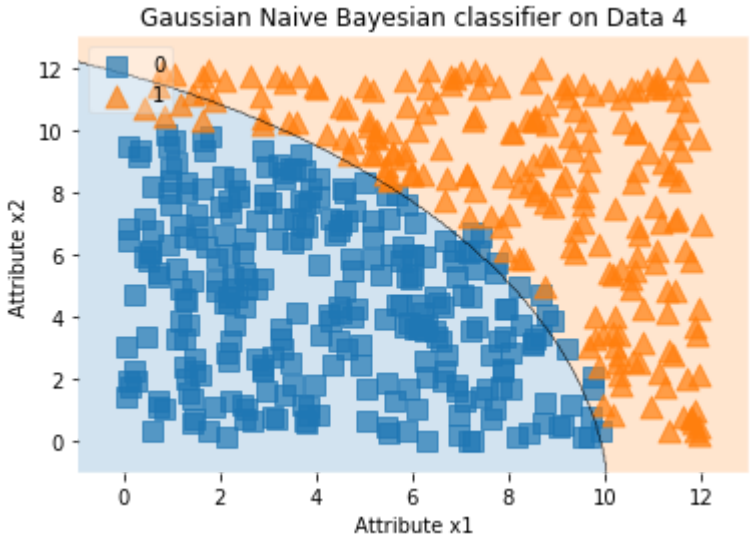


In [41]:
```
# Training a classifier
nb_data4 = GaussianNB()
nb_data4.fit(Data4_X, Data4_Y)

# parameters to set size or markers, contours, and transparency
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}

# passing above parameters to the plotting function
plot_decision_regions(X=Data4_X, y=Data4_Y, clf=nb_data4, legend=2,
                      scatter_kwargs=scatter_kwargs,
                      contourf_kwargs=contourf_kwargs,
                      scatter_highlight_kwargs=scatter_highlight_kwargs)

# Adding axes annotations
plt.xlabel('Attribute x1')
plt.ylabel('Attribute x2')
plt.title('Gaussian Naive Bayesian classifier on Data 4')
plt.show()
```



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

Answer: The decision boundary is a linear stright line for Data q, circular/elliptical for data 2 and 3 whereas it's a curve for data 4. The boundaries were created based on the structure and order of the data

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

Answer: NB classifier works best on neat data which is seen in datasets 1 and 4. Datasets 2 and 3 are in unordered form(scattered and overlapping) and hence NB performs poorly on them

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: Visually examining the data, it can be observed that the linear SVM would perform the best for data1 and then to data4 . Data 2 and data 3 are scattered/overlapping and hence linear SVM may not produce accurate results

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

In [48]:
```
svm_linear_data1 = SVC(C=0.5, kernel='linear')
svm_linear_data1_scores = cross_val_score(svm_linear_data1, Data1_X, Data1_Y,
cv=10, scoring='accuracy')
print(svm_linear_data1_scores)
[svm_linear_data1_scores.mean(), svm_linear_data1_scores.std()]
```

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

Out[48]:
```
[0.9674999999999999, 0.02968585521759479]
```

In [49]:
```
svm_linear_data2 = SVC(C=0.5, kernel='linear')
svm_linear_data2_scores = cross_val_score(svm_linear_data2, Data2_X, Data2_Y,
cv=10, scoring='accuracy')
print(svm_linear_data2_scores)
[svm_linear_data2_scores.mean(), svm_linear_data2_scores.std()]
```

```
[0.125  0.1375 0.0125 0.0875 0.2    0.2375 0.1    0.15   0.1875 0.175 ]
```

Out[49]:
```
[0.14125000000000001, 0.06124999999999999]
```

In [50]:
```
svm_linear_data3 = SVC(C=0.5, kernel='linear')
svm_linear_data3_scores = cross_val_score(svm_linear_data3, Data3_X, Data3_Y,
cv=10, scoring='accuracy')
print(svm_linear_data3_scores)
[svm_linear_data3_scores.mean(), svm_linear_data3_scores.std()]
```

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

Out[50]:
```
[0.6425000000000001, 0.0275]
```

In [51]:
```
svm_linear_data4 = SVC(C=0.5, kernel='linear')
svm_linear_data4_scores = cross_val_score(svm_linear_data4, Data4_X, Data4_Y,
cv=10, scoring='accuracy')
print(svm_linear_data4_scores)
[svm_linear_data4_scores.mean(), svm_linear_data4_scores.std()]
```

```
[0.94117647 0.90196078 0.92       0.92       0.98       0.92
 0.94       0.92       0.95918367 0.85714286]
```

Out[51]:
```
[0.9259463785514207, 0.03147481962633185]
```

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

Answer: Based on the mean of the accuracies, below is the decreasing order:
Data1(96.7%) > Data4(92.5%) > Data3(64.2%) > Data2(14.1%)

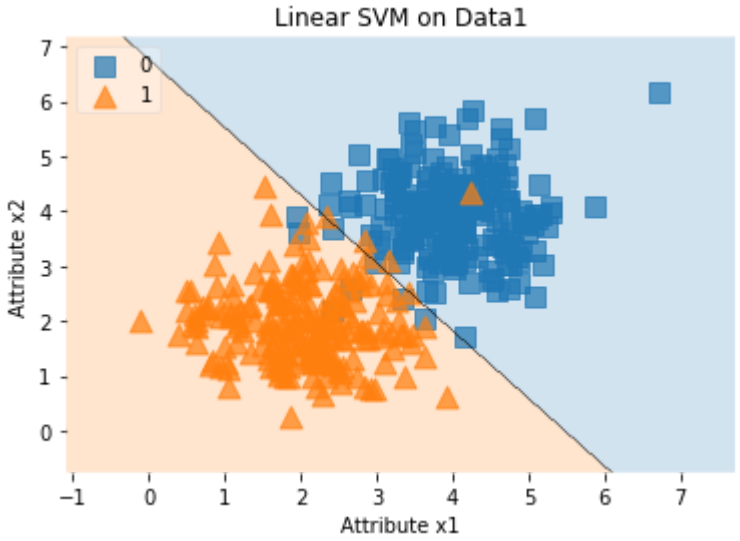
Question 4d: Plot decision regions for a linear SVM classifier on each of the four datasets

In [56]:
```
# Training a classifier
svm_linear_data1 = SVC(kernel='linear')
svm_linear_data1.fit(Data1_X, Data1_Y)

# parameters to set size or markers, contours, and transparency
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}

# passing above parameters to the plotting function
plot_decision_regions(X=Data1_X, y=Data1_Y, clf=svm_linear_data1, legend=2,
                      scatter_kwargs=scatter_kwargs,
                      contourf_kwargs=contourf_kwargs,
                      scatter_highlight_kwargs=scatter_highlight_kwargs)

# Adding axes annotations
plt.xlabel('Attribute x1')
plt.ylabel('Attribute x2')
plt.title('Linear SVM on Data1')
plt.show()
```

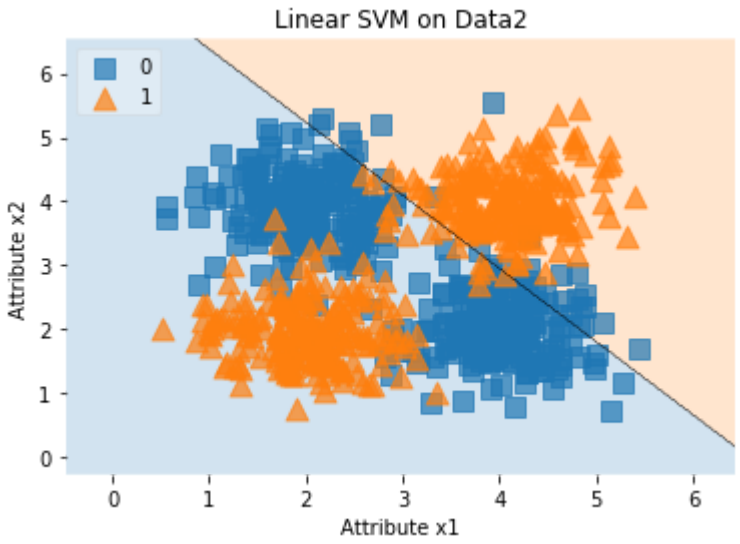


In [57]:
```
# Training a classifier
svm_linear_data2 = SVC(kernel='linear')
svm_linear_data2.fit(Data2_X, Data2_Y)

# parameters to set size or markers, contours, and transparency
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}

# passing above parameters to the plotting function
plot_decision_regions(X=Data2_X, y=Data2_Y, clf=svm_linear_data2, legend=2,
                      scatter_kwargs=scatter_kwargs,
                      contourf_kwargs=contourf_kwargs,
                      scatter_highlight_kwargs=scatter_highlight_kwargs)

# Adding axes annotations
plt.xlabel('Attribute x1')
plt.ylabel('Attribute x2')
plt.title('Linear SVM on Data2')
plt.show()
```

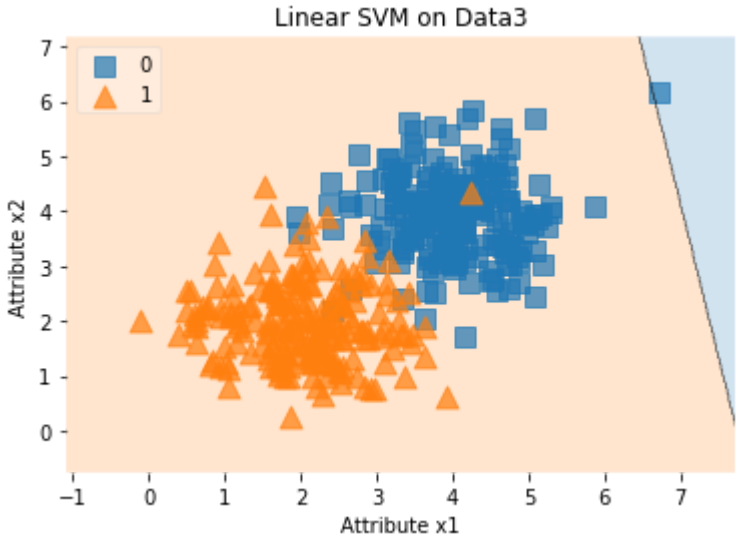


In [58]:
```
# Training a classifier
svm_linear_data3 = SVC(kernel='linear')
svm_linear_data3.fit(Data3_X, Data3_Y)

# parameters to set size or markers, contours, and transparency
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}

# passing above parameters to the plotting function
plot_decision_regions(X=Data1_X, y=Data1_Y, clf=svm_linear_data3, legend=2,
                      scatter_kwargs=scatter_kwargs,
                      contourf_kwargs=contourf_kwargs,
                      scatter_highlight_kwargs=scatter_highlight_kwargs)

# Adding axes annotations
plt.xlabel('Attribute x1')
plt.ylabel('Attribute x2')
plt.title('Linear SVM on Data3')
plt.show()
```

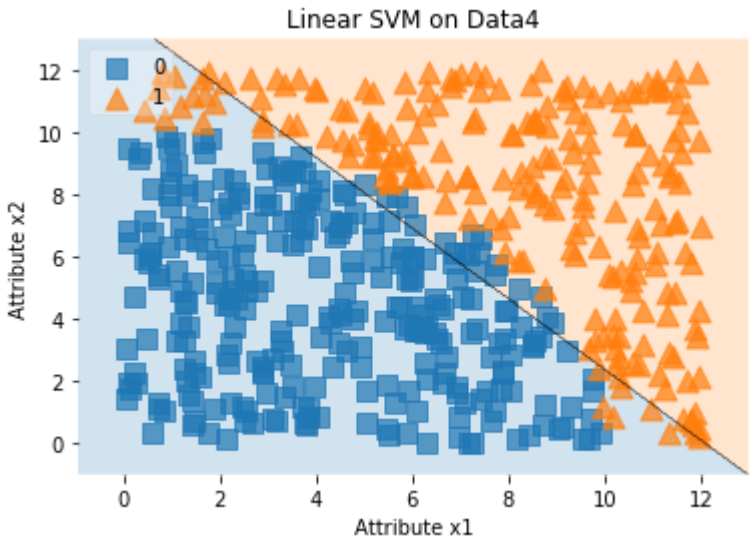


In [59]:
```
# Training a classifier
svm_linear_data4 = SVC(kernel='linear')
svm_linear_data4.fit(Data4_X, Data4_Y)

# parameters to set size or markers, contours, and transparency
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}

# passing above parameters to the plotting function
plot_decision_regions(X=Data4_X, y=Data4_Y, clf=svm_linear_data4, legend=2,
                      scatter_kwargs=scatter_kwargs,
                      contourf_kwargs=contourf_kwargs,
                      scatter_highlight_kwargs=scatter_highlight_kwargs)

# Adding axes annotations
plt.xlabel('Attribute x1')
plt.ylabel('Attribute x2')
plt.title('Linear SVM on Data4')
plt.show()
```



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

Answer: The mean accuracy values in question 4c and decision boundaraies in question 4d are symmetrical and inline with their outputs.
A clear decision boudnary is observed for data1 and data 4, where as the decision boundary is not very accurate for data3 and data2

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.

In [63]:
```
svm_poly_degree1 = SVC(kernel='poly',degree=1, gamma = 'auto')
svm_poly_degree1_scores = cross_val_score(svm_poly_degree1, Data2_X, Data2_Y,
cv=10, scoring='accuracy')
print(svm_poly_degree1_scores)
[svm_poly_degree1_scores.mean(), svm_poly_degree1_scores.std()]
```

```
[0.125  0.1375 0.0125 0.0875 0.2    0.2375 0.1    0.15   0.1875 0.175 ]
```

Out[63]:
```
[0.14125000000000001, 0.06124999999999999]
```

In [64]:
```
svm_poly_degree2 = SVC(kernel='poly',degree=2, gamma = 'auto')
svm_poly_degree2_scores = cross_val_score(svm_poly_degree2, Data2_X, Data2_Y,
cv=10, scoring='accuracy')
print(svm_poly_degree2_scores)
[svm_poly_degree2_scores.mean(), svm_poly_degree2_scores.std()]
```

```
[0.8    0.8375 0.8875 0.85   0.9125 0.9    0.8625 0.8875 0.9125 0.8625]
```

Out[64]:
```
[0.8712500000000001, 0.03402664397204048]
```

In [65]:
```
svm_poly_degree3 = SVC(kernel='poly',degree=3, gamma = 'auto')
svm_poly_degree3_scores = cross_val_score(svm_poly_degree3, Data2_X, Data2_Y,
cv=10, scoring='accuracy')
print(svm_poly_degree3_scores)
[svm_poly_degree3_scores.mean(), svm_poly_degree3_scores.std()]
```

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

Out[65]:
```
[0.8762500000000001, 0.026487025125521375]
```

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

Answer: Based on the mean value of accuracies, below is the ranking order of polynomial kernals of various degrees:
Degree 3 (87.6%) > Degree 2 (87.1%) > Degree 1 (14.1%)

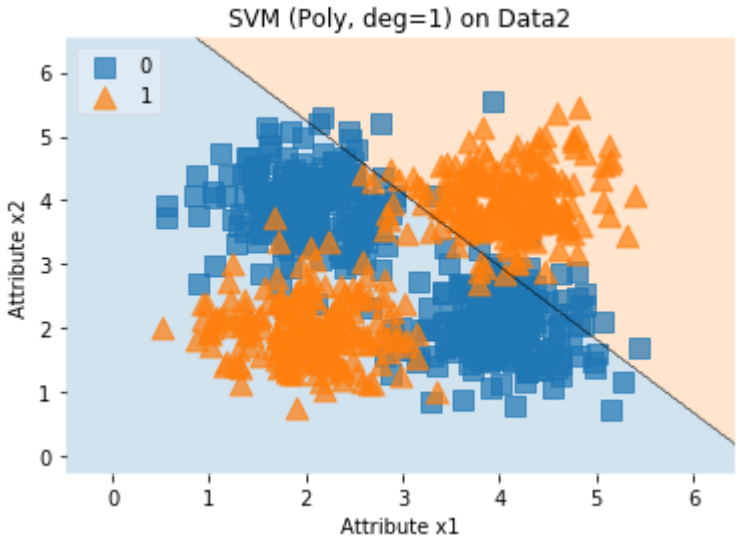
Question 5c: Plot decision regions for a polynomial kernel SVM with degree values 1, 2, and 3.

In [66]:
```
# Training a classifier
svm_poly_degree1 = SVC(kernel='poly', degree = 1, gamma = 'auto')
svm_poly_degree1.fit(Data2_X, Data2_Y)

# parameters to set size or markers, contours, and transparency
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}

# passing above parameters to the plotting function
plot_decision_regions(X=Data2_X, y=Data2_Y, clf=svm_poly_degree1, legend=2,
                      scatter_kwargs=scatter_kwargs,
                      contourf_kwargs=contourf_kwargs,
                      scatter_highlight_kwargs=scatter_highlight_kwargs)

# Adding axes annotations
plt.xlabel('Attribute x1')
plt.ylabel('Attribute x2')
plt.title('SVM (Poly, deg=1) on Data2')
plt.show()
```

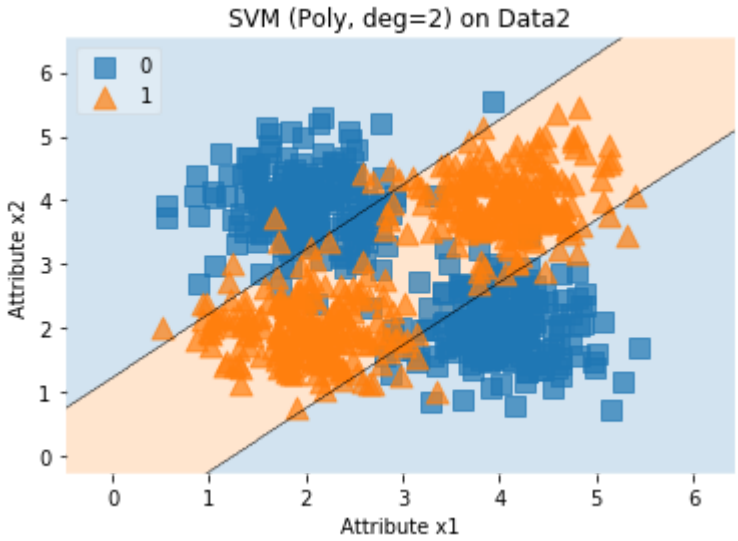


In [67]:
```
# Training a classifier
svm_poly_degree2 = SVC(kernel='poly', degree = 2, gamma = 'auto')
svm_poly_degree2.fit(Data2_X, Data2_Y)

# parameters to set size or markers, contours, and transparency
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}

# passing above parameters to the plotting function
plot_decision_regions(X=Data2_X, y=Data2_Y, clf=svm_poly_degree2, legend=2,
                      scatter_kwargs=scatter_kwargs,
                      contourf_kwargs=contourf_kwargs,
                      scatter_highlight_kwargs=scatter_highlight_kwargs)

# Adding axes annotations
plt.xlabel('Attribute x1')
plt.ylabel('Attribute x2')
plt.title('SVM (Poly, deg=2) on Data2')
plt.show()
```

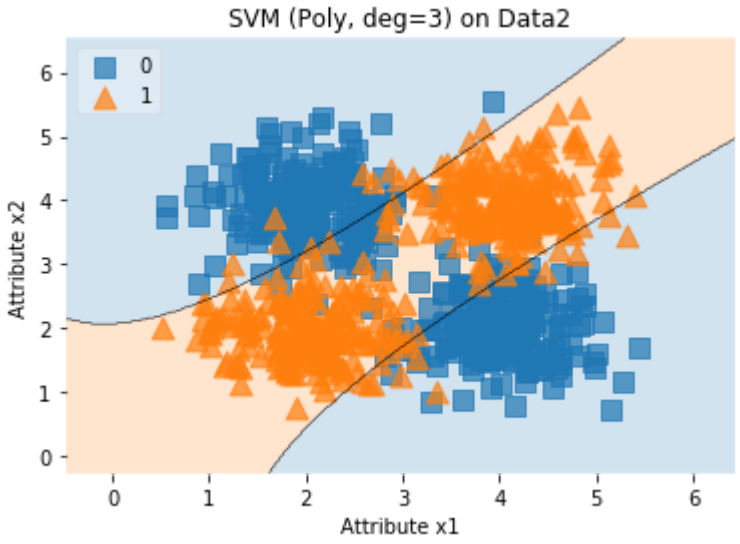


In [68]:
```
# Training a classifier
svm_poly_degree3 = SVC(kernel='poly', degree = 3, gamma = 'auto')
svm_poly_degree3.fit(Data2_X, Data2_Y)

# parameters to set size or markers, contours, and transparency
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}

# passing above parameters to the plotting function
plot_decision_regions(X=Data2_X, y=Data2_Y, clf=svm_poly_degree3, legend=2,
                      scatter_kwargs=scatter_kwargs,
                      contourf_kwargs=contourf_kwargs,
                      scatter_highlight_kwargs=scatter_highlight_kwargs)

# Adding axes annotations
plt.xlabel('Attribute x1')
plt.ylabel('Attribute x2')
plt.title('SVM (Poly, deg=3) on Data2')
plt.show()
```



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

Answer:

The mean accuracy values in question 5b and decision boundaraies in question 5c are symmetrical and inline with their outputs.
A clear decision boundary is observed when Degree is choosed as 3 (87.6%), followed by Degree 2 (87.1%) and Degree 1 (14.1%)

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.

In [70]:
```
svm_rbf_1 = SVC(kernel='rbf', gamma=0.01)
svm_rbf_1_scores = cross_val_score(svm_rbf_1, Data2_X, Data2_Y, cv=10, scoring
='accuracy')
print(svm_rbf_1_scores)
[svm_rbf_1_scores.mean(), svm_rbf_1_scores.std()]
```

```
[0.5    0.475  0.4125 0.475  0.4875 0.4375 0.4625 0.5    0.475  0.4375]
```

Out[70]: [0.46624999999999994, 0.027414640249326636]

In [71]:
```
svm_rbf_2 = SVC(kernel='rbf', gamma=0.1)
svm_rbf_2_scores = cross_val_score(svm_rbf_2, Data2_X, Data2_Y, cv=10, scoring
='accuracy')
print(svm_rbf_2_scores)
[svm_rbf_2_scores.mean(), svm_rbf_2_scores.std()]
```

```
[0.9875 0.9125 0.9375 0.9125 0.9625 0.9375 0.875  0.9375 0.9625 0.9625]
```

Out[71]: [0.9387500000000001, 0.03084740669813268]

In [72]:
```
svm_rbf_3 = SVC(kernel='rbf', gamma=1)
svm_rbf_3_scores = cross_val_score(svm_rbf_3, Data2_X, Data2_Y, cv=10, scoring
='accuracy')
print(svm_rbf_3_scores)
[svm_rbf_3_scores.mean(), svm_rbf_3_scores.std()]
```

```
[1.     0.9125 0.9375 0.9375 0.9625 0.9625 0.8625 0.9375 0.9625 0.95  ]
```

Out[72]: [0.9424999999999999, 0.03455068740271312]

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

Answer: Based on the mean value of accuracies, RBF kernals are ordered as below:
Gamma 1(94.2%) > Gamma 0.1(93.8%) > Gamma 0.01(46.6%)

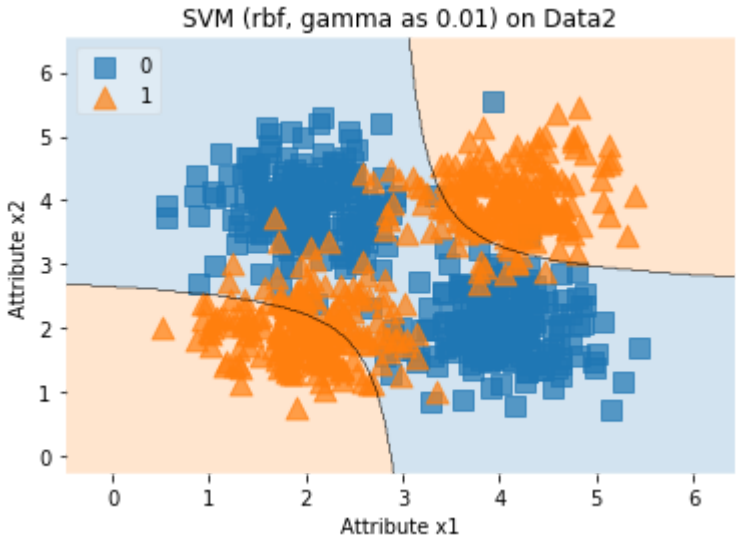
Question 5g: Plot decision regions for the above RBF Kernels

In [74]:
```
# Training a classifier
svm_rbf_1 = SVC(kernel='rbf', gamma = 0.01)
svm_rbf_1.fit(Data2_X, Data2_Y)

# parameters to set size or markers, contours, and transparency
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}

# passing above parameters to the plotting function
plot_decision_regions(X=Data2_X, y=Data2_Y, clf=svm_rbf_1, legend=2,
                      scatter_kwargs=scatter_kwargs,
                      contourf_kwargs=contourf_kwargs,
                      scatter_highlight_kwargs=scatter_highlight_kwargs)

# Adding axes annotations
plt.xlabel('Attribute x1')
plt.ylabel('Attribute x2')
plt.title('SVM (rbf, gamma as 0.01) on Data2')
plt.show()
```

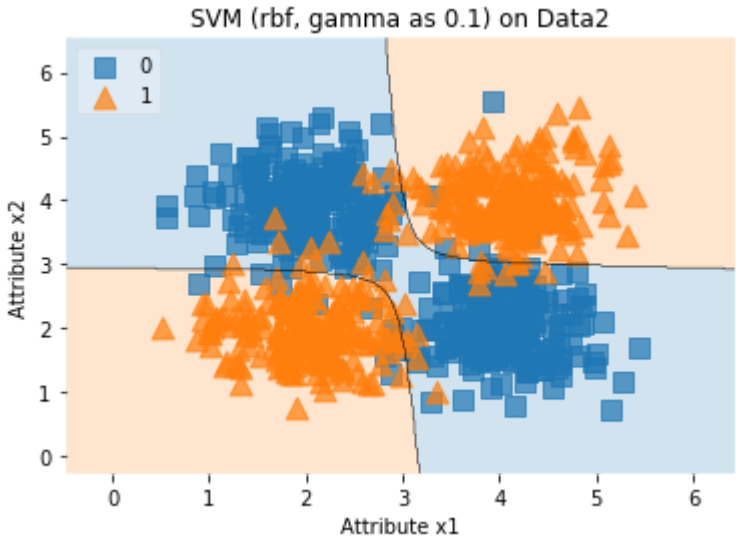


In [75]:
```
# Training a classifier
svm_rbf_2 = SVC(kernel='rbf', gamma = 0.1)
svm_rbf_2.fit(Data2_X, Data2_Y)

# parameters to set size or markers, contours, and transparency
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}

# passing above parameters to the plotting function
plot_decision_regions(X=Data2_X, y=Data2_Y, clf=svm_rbf_2, legend=2,
                      scatter_kwargs=scatter_kwargs,
                      contourf_kwargs=contourf_kwargs,
                      scatter_highlight_kwargs=scatter_highlight_kwargs)

# Adding axes annotations
plt.xlabel('Attribute x1')
plt.ylabel('Attribute x2')
plt.title('SVM (rbf, gamma as 0.1) on Data2')
plt.show()
```

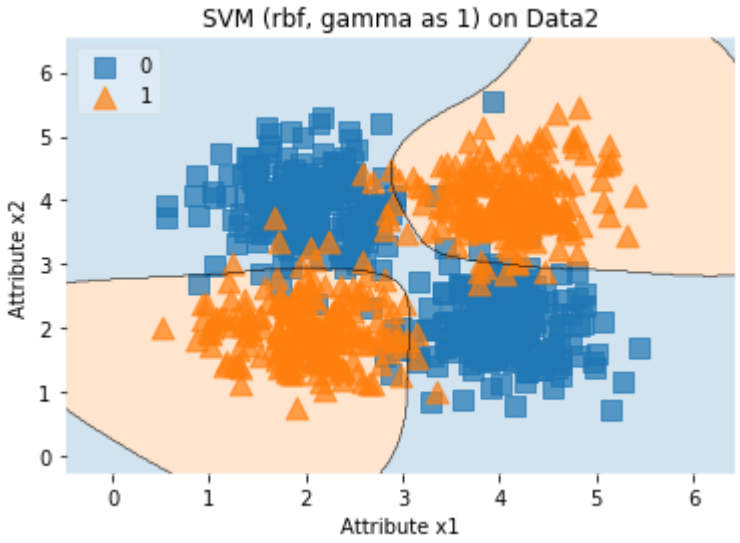


In [76]:
```
# Training a classifier
svm_rbf_3 = SVC(kernel='rbf', gamma = 1)
svm_rbf_3.fit(Data2_X, Data2_Y)

# parameters to set size or markers, contours, and transparency
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}

# passing above parameters to the plotting function
plot_decision_regions(X=Data2_X, y=Data2_Y, clf=svm_rbf_3, legend=2,
                      scatter_kwargs=scatter_kwargs,
                      contourf_kwargs=contourf_kwargs,
                      scatter_highlight_kwargs=scatter_highlight_kwargs)

# Adding axes annotations
plt.xlabel('Attribute x1')
plt.ylabel('Attribute x2')
plt.title('SVM (rbf, gamma as 1) on Data2')
plt.show()
```



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

Answer: Gamma is defined as the spread of the kernal and therefore the decision boundary. When gamma is low, the 'curve' of the decision boundary is very low and thus the decision region is very broad.
Hence we can observe that Gamma 1(94.2%) > Gamma 0.1(93.8%) > Gamma 0.01(46.6%)

Question 5i: Between SVM with a Polynomial kernel and SVM with an RBF kernel, which one is ideally suited of Data3? Explain your reason.

Answer: *Correction for the question as data2, instead of data3 considered*
SVM kernal with RBF is best suited considering the accuracy values

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.

In [79]:
```
svm_rbf_cv_3 = SVC(kernel='rbf', gamma='auto')
svm_rbf_cv_3_scores = cross_val_score(svm_rbf_cv_3, Data2_X, Data2_Y, cv=3, sc
oring='accuracy')
print(svm_rbf_cv_3_scores)
print('Mean for cv=3:')
[svm_rbf_cv_3_scores.mean()]
```

```
[0.91044776 0.93984962 0.92857143]
Mean for cv=3:
```

Out[79]:
```
[0.9262896046085363]
```

In [80]:
```
svm_rbf_cv_4 = SVC(kernel='rbf', gamma='auto')
svm_rbf_cv_4_scores = cross_val_score(svm_rbf_cv_4, Data2_X, Data2_Y, cv=4, sc
oring='accuracy')
print(svm_rbf_cv_4_scores)
print('Mean for cv=4:')
[svm_rbf_cv_4_scores.mean()]
```

```
[0.93  0.935 0.9   0.955]
Mean for cv=4:
```

Out[80]:
```
[0.93]
```

In [81]:
```
svm_rbf_cv_5 = SVC(kernel='rbf', gamma='auto')
svm_rbf_cv_5_scores = cross_val_score(svm_rbf_cv_5, Data2_X, Data2_Y, cv=5, sc
oring='accuracy')
print(svm_rbf_cv_5_scores)
print('Mean for cv=5:')
[svm_rbf_cv_5_scores.mean()]
```

```
[0.94375 0.9125  0.95    0.90625 0.95625]
Mean for cv=5:
```

Out[81]:
```
[0.9337500000000001]
```

In [82]:
```
svm_rbf_cv_6 = SVC(kernel='rbf', gamma='auto')
svm_rbf_cv_6_scores = cross_val_score(svm_rbf_cv_6, Data2_X, Data2_Y, cv=6, sc
oring='accuracy')
print(svm_rbf_cv_6_scores)
print('Mean for cv=6:')
[svm_rbf_cv_6_scores.mean()]
```

```
[0.97014925 0.91791045 0.95522388 0.90298507 0.93939394 0.96212121]
Mean for cv=6:
```

Out[82]:
```
[0.9412973013719282]
```

Answer:
Mean for cv=3: **92%**
Mean for cv=4: **93%**
Mean for cv=5: **93.3%**
Mean for cv=6: **94.1%**
The difference in the mean accuracies is because of the different value of the cv choosen. With the increase in the number of folds, accuracy of the classifier increases

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.

In [31]:
```
#DT:

dt_4 = DecisionTreeClassifier(max_depth=4)
dt_4_accuracy_scores = cross_val_score(dt_4, Data3_X, Data3_Y, cv=30, scoring=
'accuracy')
dt_4_precision_scores = cross_val_score(dt_4, Data3_X, Data3_Y, cv=30, scoring
='precision')
print('\n')
print('Decision Tree:')
print ('Accuracy:',dt_4_accuracy_scores.mean())
print ('Precision:',dt_4_precision_scores.mean())
#[dt_2_scores.mean(), dt_2_scores.std()]

#KNN:

knn_5 = KNeighborsClassifier(n_neighbors=5)
knn_5_accuracy_scores = cross_val_score(knn_5, Data2_X, Data2_Y, cv=10, scorin
g='accuracy')
knn_5_precision_scores = cross_val_score(knn_5, Data2_X, Data2_Y, cv=10, scori
ng='precision')
print('\n')
print('KNN:')
print ('Accuracy:',dt_4_accuracy_scores.mean())
print ('Precision:',dt_4_precision_scores.mean())
#print(knn_5_scores)

#Naive Bayes:

nb_data1 = GaussianNB()
nb_data1_accuracy_scores = cross_val_score(nb_data1, Data1_X, Data1_Y, cv=10,
scoring='accuracy')
nb_data1_precision_scores = cross_val_score(nb_data1, Data1_X, Data1_Y, cv=10,
scoring='precision')
print('\n')
print('Naive Bayesian:')
print ('Accuracy:',nb_data1_accuracy_scores.mean())
print ('Precision:',nb_data1_precision_scores.mean())

#Linear SVM:

svm_linear_data1 = SVC(C=0.5, kernel='linear')
svm_linear_data1_accuracy_scores = cross_val_score(svm_linear_data1, Data1_X,
Data1_Y, cv=10, scoring='accuracy')
svm_linear_data1_precision_scores = cross_val_score(svm_linear_data1, Data1_X,
Data1_Y, cv=10, scoring='precision')
print('\n')
print('Linear SVM:')
print ('Accuracy:',svm_linear_data1_accuracy_scores.mean())
print ('Precision:',svm_linear_data1_precision_scores.mean())

#Kernal poly SVM:
svm_poly_degree1 = SVC(kernel='poly',degree=1, gamma = 'auto')
svm_poly_degree1_accuracy_scores = cross_val_score(svm_poly_degree1, Data2_X,
Data2_Y, cv=10, scoring='accuracy')
svm_poly_degree1_precision_scores = cross_val_score(svm_poly_degree1, Data2_X,
Data2_Y, cv=10, scoring='precision')
```

```
print('\n')
print('Poly SVM:')
print ('Accuracy:',svm_poly_degree1_accuracy_scores.mean())
print ('Precision:',svm_poly_degree1_precision_scores.mean())

#SVM RBF:

svm_rbf_3 = SVC(kernel='rbf', gamma=1)
svm_rbf_3_accuracy_scores = cross_val_score(svm_rbf_3, Data2_X, Data2_Y, cv=10
, scoring='accuracy')
svm_rbf_3_precision_scores = cross_val_score(svm_rbf_3, Data2_X, Data2_Y, cv=1
0, scoring='precision')

print('\n')
print('SVM RBF:')
print ('Accuracy:',svm_rbf_3_accuracy_scores.mean())
print ('Precision:',svm_rbf_3_precision_scores.mean())
```

```
Decision Tree:
Accuracy: 0.9718253968253968
Precision: 0.9604497354497356


KNN:
Accuracy: 0.9718253968253968
Precision: 0.9604497354497356


Naive Bayesian:
Accuracy: 0.9675
Precision: 0.9668771566597654


Linear SVM:
Accuracy: 0.9674999999999999
Precision: 0.9658297258297258


Poly SVM:
Accuracy: 0.14125000000000001
Precision: 0.135276964855303


SVM RBF:
Accuracy: 0.9424999999999999
Precision: 0.9528976725318188
```

Answer: Ranking based on accuracy:DT,NB,KNN,RBF,Linear SVM,Poly SVM
Ranking based on precision: DT,NB,KNN,RBF,Linear SVM,Poly SVM

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.

In [8]:
```
svm_linear = SVC(C=0.5, kernel='linear', gamma = 'auto')
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_linear, n_estimators=n_est)
    # compute cross-validation accuracy for each bagging classifier
    scores = cross_val_score(bagging, Data3_X, Data3_Y, cv=5, scoring='accurac
y')
    print("Mean Bagging Accuracy: %.2f Standard Deviation (+/-): %.2f for esti
mators: %d" % (scores.mean(), scores.std(), n_est))
```

```
Mean Bagging Accuracy: 0.64 Standard Deviation (+/-): 0.03 for estimators: 1
Mean Bagging Accuracy: 0.62 Standard Deviation (+/-): 0.12 for estimators: 2
Mean Bagging Accuracy: 0.65 Standard Deviation (+/-): 0.03 for estimators: 3
Mean Bagging Accuracy: 0.62 Standard Deviation (+/-): 0.08 for estimators: 4
Mean Bagging Accuracy: 0.56 Standard Deviation (+/-): 0.08 for estimators: 5
Mean Bagging Accuracy: 0.68 Standard Deviation (+/-): 0.04 for estimators: 10
Mean Bagging Accuracy: 0.71 Standard Deviation (+/-): 0.09 for estimators: 20
```

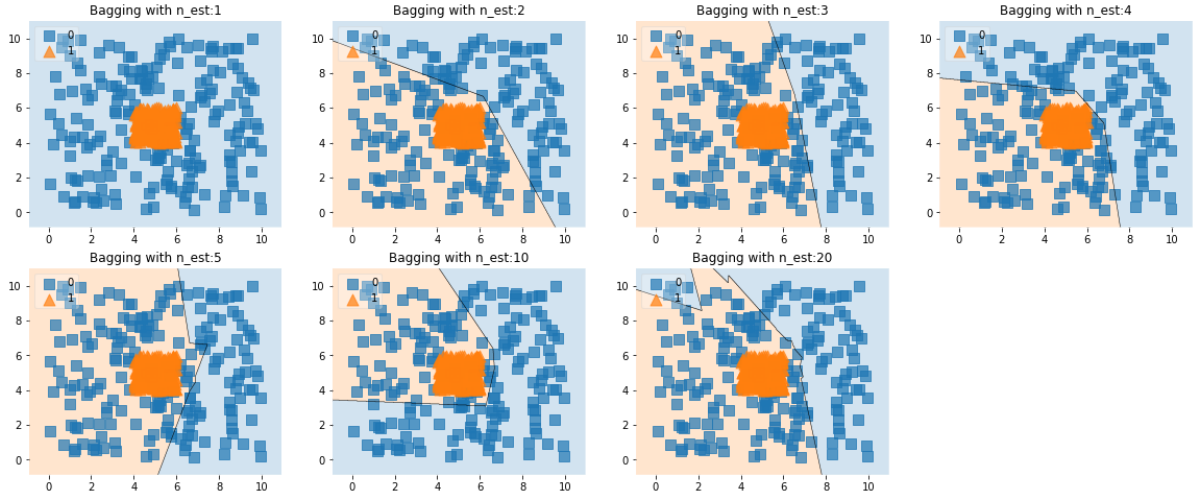
Answer: Bagging estimator reaches its highest accuracy value with increase in the number of estimators untill it reaches its threshold. In this case, when number of estimators is 20, the mean bagging accuracy is the highest and standard deviation is less (not minimum).

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

In [93]:
```
svm_linear = SVC(C=0.5, kernel='linear', gamma = 'auto')
n_est_list = [1,2,3,4,5,10,20]
fig = plt.figure(figsize=(20, 8))
count = 0;
for n_est in n_est_list:
    count = count + 1;
    bagging = BaggingClassifier(base_estimator=svm_linear, 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()
```

```
C:\Users\15139\AppData\Roaming\Python\Python37\site-packages\mlxtend\plotting
\decision_regions.py:247: UserWarning: No contour levels were found within th
e 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: When one estimator is used, the decsion regions are not clearly marked where as when many estimators are used, the decision regions became clearer and well separated

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.

In [92]:
```
dt = 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=dt, n_estimators=n_est)
    # compute cross-validation accuracy for each bagging classifier
    scores = cross_val_score(boosting, Data2_X, Data2_Y, cv=10, scoring='accur
acy')
    print("Mean Boosting Accuracy: %.2f Standard Deviation(+/-): %.2f for esti
mators: %d" % (scores.mean(), scores.std(), n_est))
```

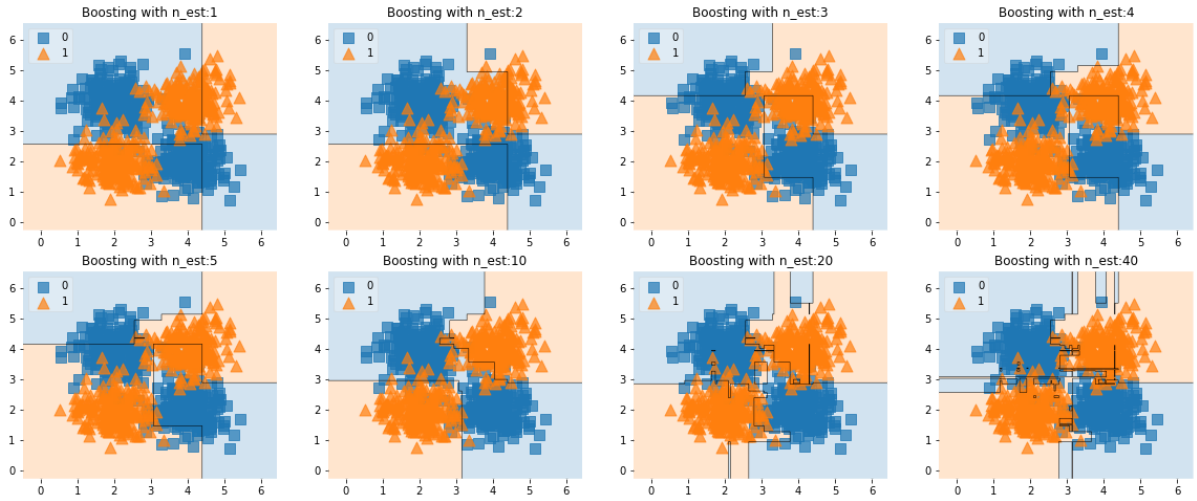
```
Mean Boosting Accuracy: 0.88 Standard Deviation(+/-): 0.03 for estimators: 1
Mean Boosting Accuracy: 0.88 Standard Deviation(+/-): 0.03 for estimators: 2
Mean Boosting Accuracy: 0.90 Standard Deviation(+/-): 0.04 for estimators: 3
Mean Boosting Accuracy: 0.90 Standard Deviation(+/-): 0.04 for estimators: 4
Mean Boosting Accuracy: 0.92 Standard Deviation(+/-): 0.03 for estimators: 5
Mean Boosting Accuracy: 0.92 Standard Deviation(+/-): 0.04 for estimators: 10
Mean Boosting Accuracy: 0.91 Standard Deviation(+/-): 0.04 for estimators: 20
Mean Boosting Accuracy: 0.91 Standard Deviation(+/-): 0.02 for estimators: 40
```

Answer: The Mean boosting accuracy is highest and standard deviation is least when number of estimators is 5

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

In [91]:
```
fig = plt.figure(figsize=(20, 8))
count = 0;
for n_est in n_est_list:
    count = count + 1;
    boosting = AdaBoostClassifier(base_estimator=dt, 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:

When the number of estimators is choosen as 5, the decision boundaries are clear and well defined(structured), inline with the theory in 7d

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.

In [11]:
```
from sklearn import datasets
cancer = datasets.load_breast_cancer()
```

The features are:

In [12]:
```
cancer.feature_names
```

Out[12]:
```
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:

In [13]:
```
cancer.target_names
```

Out[13]:
```
array(['malignant', 'benign'], dtype='<U9')
```

Create dataset for classification

In [14]:
```
X = cancer.data
Y = cancer.target
```

Number of samples are:

In [22]:
```
X.shape
```

Out[22]:
```
(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].

In [20]:
```
#Linear SVM
c_list = [0.001,0.01,0.1,1]
for c in c_list:
    # create an instance of a SVM
    svm_linear = SVC(c, kernel='linear')
    #boosting = AdaBoostClassifier(base_estimator=dt, n_estimators=n_est)
    # compute cross-validation accuracy
    scores = cross_val_score(svm_linear,X,Y, cv=10, scoring='accuracy')
    print('C value:',c)
    print("Accuracy: %.2f ; Standard Deviation(+/-) %.2f" % (scores.mean(), sc
ores.std()))
    print('\n')
```

```
C value: 0.001
Accuracy: 0.94 ; Standard Deviation(+/-) 0.03


C value: 0.01
Accuracy: 0.95 ; Standard Deviation(+/-) 0.03


C value: 0.1
Accuracy: 0.95 ; Standard Deviation(+/-) 0.02


C value: 1
Accuracy: 0.95 ; Standard Deviation(+/-) 0.02
```

In [23]:
```
#Polynomial varying C and degree
c_list = [0.001,0.01,0.1,1]
degree_list = [1,2]
for c in c_list:
    for d in degree_list:
        # create an instance of a SVM
        svm_poly = SVC(c,kernel='poly', degree = d, gamma = 'auto')
        # compute cross-validation accuracy
        scores = cross_val_score(svm_poly,X,Y, cv=10, scoring='accuracy')
        print('C value:',c)
        print('Degree:',d)
        print("Accuracy: %.2f ; Standard Deviation(+/-) %.2f " % (scores.mean
(), scores.std()))
        print('\n')
```

```
C value: 0.001
Degree: 1
Accuracy: 0.93 ; Standard Deviation(+/-) 0.03


C value: 0.001
Degree: 2
Accuracy: 0.95 ; Standard Deviation(+/-) 0.03


C value: 0.01
Degree: 1
Accuracy: 0.94 ; Standard Deviation(+/-) 0.02


C value: 0.01
Degree: 2
Accuracy: 0.96 ; Standard Deviation(+/-) 0.03


C value: 0.1
Degree: 1
Accuracy: 0.95 ; Standard Deviation(+/-) 0.02


C value: 0.1
Degree: 2
Accuracy: 0.96 ; Standard Deviation(+/-) 0.02


C value: 1
Degree: 1
Accuracy: 0.95 ; Standard Deviation(+/-) 0.02


C value: 1
Degree: 2
Accuracy: 0.96 ; Standard Deviation(+/-) 0.02
```

In [19]:
```
#RBF varying C and gamma
c_list = [0.001,0.01,0.1,1]
#degree_list = [1,2,3]
gamma_list = [0.001, 0.01, 0.1, 1, 10, 100]
for c in c_list:
    #for d in degree_list:
        for g in gamma_list:
            # create an instance of a SVM
            svm_rbf = SVC(c,kernel='rbf', degree = 1, gamma = g)
            # compute cross-validation accuracy
            scores = cross_val_score(svm_rbf,X,Y, cv=10, scoring='accuracy')
            print('C value:',c)
            print('Gamma value:', g)
            print("Accuracy: %.2f ; Standard Deviation(+/-) %.2f " % (scores.m
ean(), scores.std()))
            print('\n')
```

```
C value: 0.001
Gamma value: 0.001
Accuracy: 0.63 ; Standard Deviation(+/-) 0.00


C value: 0.001
Gamma value: 0.01
Accuracy: 0.63 ; Standard Deviation(+/-) 0.00


C value: 0.001
Gamma value: 0.1
Accuracy: 0.63 ; Standard Deviation(+/-) 0.00


C value: 0.001
Gamma value: 1
Accuracy: 0.63 ; Standard Deviation(+/-) 0.00


C value: 0.001
Gamma value: 10
Accuracy: 0.63 ; Standard Deviation(+/-) 0.00


C value: 0.001
Gamma value: 100
Accuracy: 0.63 ; Standard Deviation(+/-) 0.00


C value: 0.01
Gamma value: 0.001
Accuracy: 0.63 ; Standard Deviation(+/-) 0.00


C value: 0.01
Gamma value: 0.01
Accuracy: 0.63 ; Standard Deviation(+/-) 0.00


C value: 0.01
Gamma value: 0.1
Accuracy: 0.63 ; Standard Deviation(+/-) 0.00


C value: 0.01
Gamma value: 1
Accuracy: 0.63 ; Standard Deviation(+/-) 0.00


C value: 0.01
Gamma value: 10
Accuracy: 0.63 ; Standard Deviation(+/-) 0.00


C value: 0.01
Gamma value: 100
```

```
Accuracy: 0.63 ; Standard Deviation(+/-) 0.00


C value: 0.1
Gamma value: 0.001
Accuracy: 0.63 ; Standard Deviation(+/-) 0.00


C value: 0.1
Gamma value: 0.01
Accuracy: 0.63 ; Standard Deviation(+/-) 0.00


C value: 0.1
Gamma value: 0.1
Accuracy: 0.63 ; Standard Deviation(+/-) 0.00


C value: 0.1
Gamma value: 1
Accuracy: 0.63 ; Standard Deviation(+/-) 0.00


C value: 0.1
Gamma value: 10
Accuracy: 0.63 ; Standard Deviation(+/-) 0.00


C value: 0.1
Gamma value: 100
Accuracy: 0.63 ; Standard Deviation(+/-) 0.00


C value: 1
Gamma value: 0.001
Accuracy: 0.92 ; Standard Deviation(+/-) 0.02


C value: 1
Gamma value: 0.01
Accuracy: 0.63 ; Standard Deviation(+/-) 0.01


C value: 1
Gamma value: 0.1
Accuracy: 0.63 ; Standard Deviation(+/-) 0.00


C value: 1
Gamma value: 1
Accuracy: 0.63 ; Standard Deviation(+/-) 0.00


C value: 1
Gamma value: 10
Accuracy: 0.63 ; Standard Deviation(+/-) 0.00
```

```
C value: 1
Gamma value: 100
Accuracy: 0.63 ; Standard Deviation(+/-) 0.00
```

Answer: Highest accuracy is observed for poly kernal SVM

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

In [26]:
```
dt_list = [2,4,6,8,10,20]
for dt in dt_list:
    # create an instance of a SVM
    dt_est=DecisionTreeClassifier(max_depth=dt)
    # compute cross-validation accuracy
    scores = cross_val_score(dt_est,X,Y, cv=10, scoring='accuracy')
    print('Max_depth:',dt)
    print("Accuracy: %.2f ; Standard Deviation(+/-) %.2f " % (scores.mean(), s
cores.std()))
```

```
Max_depth: 2
Accuracy: 0.92 ; Standard Deviation(+/-) 0.03
Max_depth: 4
Accuracy: 0.91 ; Standard Deviation(+/-) 0.03
Max_depth: 6
Accuracy: 0.91 ; Standard Deviation(+/-) 0.04
Max_depth: 8
Accuracy: 0.92 ; Standard Deviation(+/-) 0.03
Max_depth: 10
Accuracy: 0.92 ; Standard Deviation(+/-) 0.03
Max_depth: 20
Accuracy: 0.91 ; Standard Deviation(+/-) 0.04
```

Answer:

Highest accuracy is observed for max_depth 2,8 and 10

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: SVM classifier is best at calssifying this type of data as it finds the best separating hyperplane based on the dimensionality of the data. In addition, using kernal SVM is efficient compared to lienar SVM as it enables us to model higher dimensional, non linear data using a kernal function