Hands-on Exercise CLASS Module

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

In [2]:
```
import numpy as np
import pandas as pd

#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
```

In [3]:
```
#Mlxtend for visualizing classification decision boundaries
from mlxtend.plotting import plot_decision_regions
```

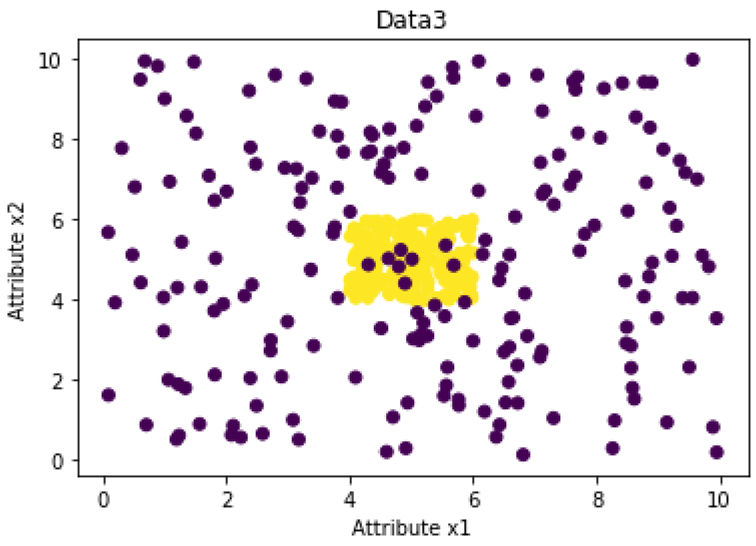
In [4]:
```
# Generating Data1

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 [5]:
```
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()
```



In [6]:
```
dt2 = DecisionTreeClassifier(max_depth=2)
dt4 = DecisionTreeClassifier(max_depth=4)
dt6 = DecisionTreeClassifier(max_depth=6)
dt8 = DecisionTreeClassifier(max_depth=8)
dt10 = DecisionTreeClassifier(max_depth=10)
dt50 = DecisionTreeClassifier(max_depth=50)
```

In [7]:
```
dt2_score = cross_val_score(dt2, Data3_X, Data3_Y, cv=10, scoring='accuracy')
dt4_score = cross_val_score(dt4, Data3_X, Data3_Y, cv=10, scoring='accuracy')
dt6_score = cross_val_score(dt6, Data3_X, Data3_Y, cv=10, scoring='accuracy')
dt8_score = cross_val_score(dt8, Data3_X, Data3_Y, cv=10, scoring='accuracy')
dt10_score = cross_val_score(dt10, Data3_X, Data3_Y, cv=10, scoring='accuracy'
)
dt50_score = cross_val_score(dt50, Data3_X, Data3_Y, cv=10, scoring='accuracy'
)
```

In [8]:
```
dt_accuracy = { 'Depth': [2,4,6,8,10,50],
    'Score': [dt2_score.mean(),dt4_score.mean(),dt6_score.mean(),dt8_score.mea
n(), dt10_score.mean(),
                          dt50_score.mean()]
                   }
dt_accuracy_df = pd.DataFrame.from_dict(dt_accuracy)
print(dt_accuracy_df)
```

```
   Depth   Score
0      2  0.8750
1      4  0.9700
2      6  0.9675
3      8  0.9500
4     10  0.9425
5     50  0.9450
```

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

```
In [9]: print(dt_accuracy_df.loc[dt_accuracy_df.Score.idxmin()])

        Depth    2.000
        Score    0.875
        Name: 0, dtype: float64
```

Answer: For max depth = 2. The accuracy was the lowest. This phenomenon is called underfitting

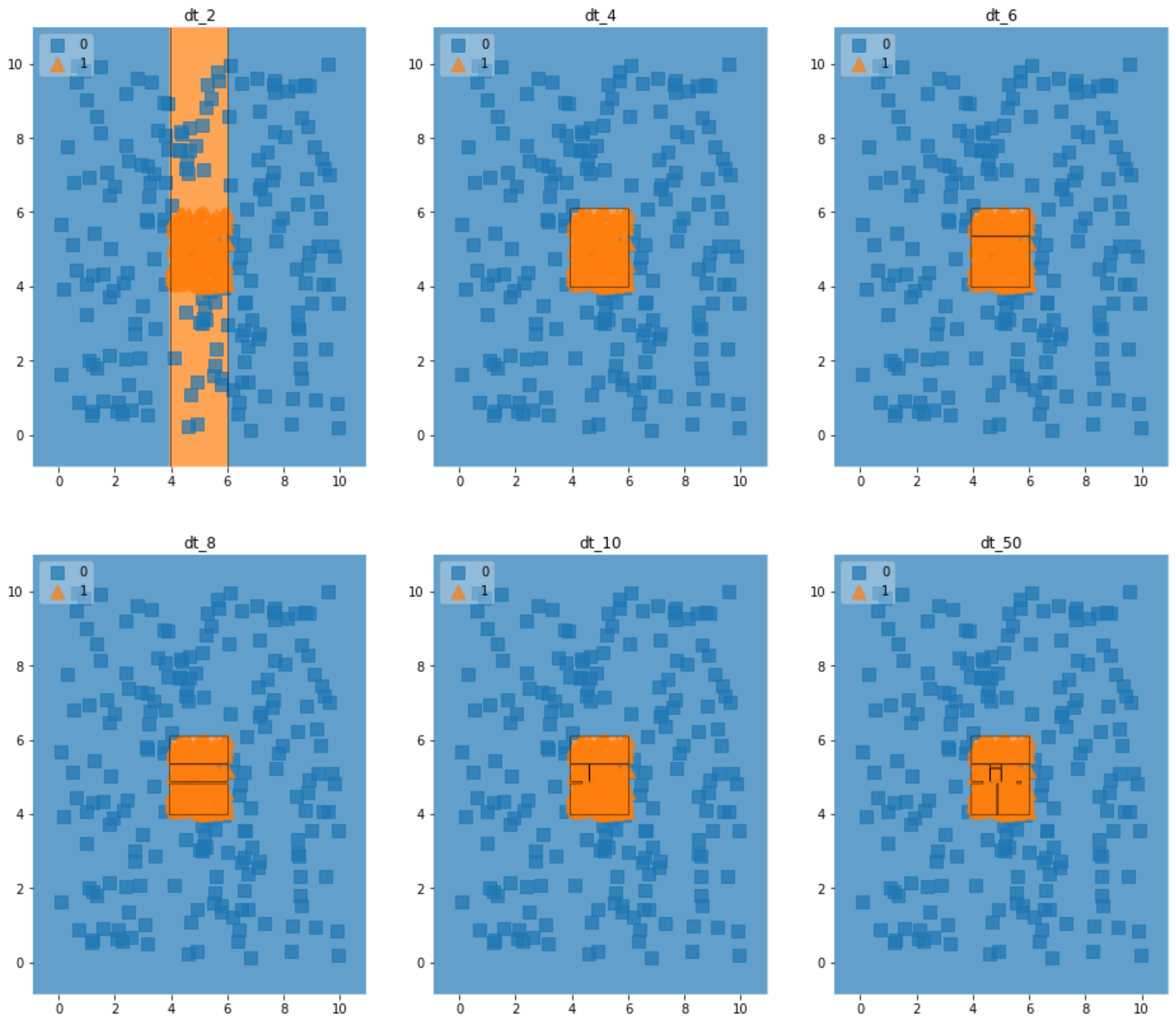
Question 1c: What accuracy did you observe for max depth=50? What is the difference between this accuracy and the highest accuracy? What is this phenomenon called?

```
In [10]: dt_accuracy_df.loc[dt_accuracy_df['Depth'] == 50.000]['Score']

Out[10]: 5    0.945
         Name: Score, dtype: float64

In [11]:  dt_accuracy_df.loc[dt_accuracy_df.Score.idxmax()]

Out[11]: Depth    4.00
         Score    0.97
         Name: 1, dtype: float64

In [12]: dt_accuracy_df.loc[dt_accuracy_df.Score.idxmax()]['Score'] - dt_accuracy_df.lo
         c[dt_accuracy_df['Depth'] == 50.000]['Score']

Out[12]: 5    0.025
         Name: Score, dtype: float64
```

**Answer: ** Accuracy for Max depth 50 is 94.5% whereas the highest accuracy is 97% for maxdepth = 4. The decrease in accuracy is due to overfitting

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

In [13]:
```
# parameters to set size or markers, contours, and transparency
scatter_kwargs = {'s': 120, 'edgecolor': None, 'alpha': 0.7}
contourf_kwargs = {'alpha': 0.7}
scatter_highlight_kwargs = {'s': 120, 'label': 'Test data', 'alpha': 0.7}


# Creating a list of classifiers and their names for plotting
clf_list = [dt2,dt4,dt6,dt8,dt10,dt50]
labels = ['dt_2','dt_4','dt_6','dt_8','dt_10','dt_50']

# Plotting the decision boundaries
fig = plt.figure(figsize=(16,14))
count = 0;

for clf, label in zip(clf_list, labels):
    count = count + 1;
    clf.fit(Data3_X, Data3_Y)
    ax = plt.subplot(2,3,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.title(label)
plt.show()
```



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

Answer: Maxdepth 4 is the better suited for the data based on decision regions, this would only misclassify 0's in the center

- Depth2: Too many 0's wrongly classified as 1's. This is an underfitting model
- Depth6: Creates 2 region in the center which wrongly classifies some of the 1's as 0's due to the presence of 0's in the central region, thus increasing the misclassification
- Depth8: Creates 3 new region in the center which wrongly classifies some of the 1's as 0's due to the presence of 0's in the central region, thus increasing the misclassification
- Depth10: Creates 5 regions in the center which wrongly classifies some of the 1's as 0's due to the presence of 0's in the central region, thus increasing the misclassification
- Depth50: Creates 7 regions in the center to account for the presence of 0's in the central region, thus increasing the misclassification. This is an example of overfitting model

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 [14]:
```
knn1 = KNeighborsClassifier(n_neighbors=1)
knn5 = KNeighborsClassifier(n_neighbors=5)
knn10 = KNeighborsClassifier(n_neighbors=10)
knn50 = KNeighborsClassifier(n_neighbors=50)
```

In [15]:
```
knn1_score = cross_val_score(knn1, Data2_X, Data2_Y, cv=10, scoring='accuracy'
)
knn5_score = cross_val_score(knn5, Data2_X, Data2_Y, cv=10, scoring='accuracy'
)
knn10_score = cross_val_score(knn10, Data2_X, Data2_Y, cv=10, scoring='accurac
y')
knn50_score = cross_val_score(knn50, Data2_X, Data2_Y, cv=10, scoring='accurac
y')
```

In [16]:
```
knn_accuracy_df = pd.DataFrame.from_dict({ 'Neighbours': [1,5,10,50],
    'Score': [knn1_score.mean(),knn5_score.mean(),knn10_score.mean(),knn50_sco
re.mean()]})
print(knn_accuracy_df)
```

```
   Neighbours    Score
0           1  0.91250
1           5  0.93500
2          10  0.94000
3          50  0.94125
```

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

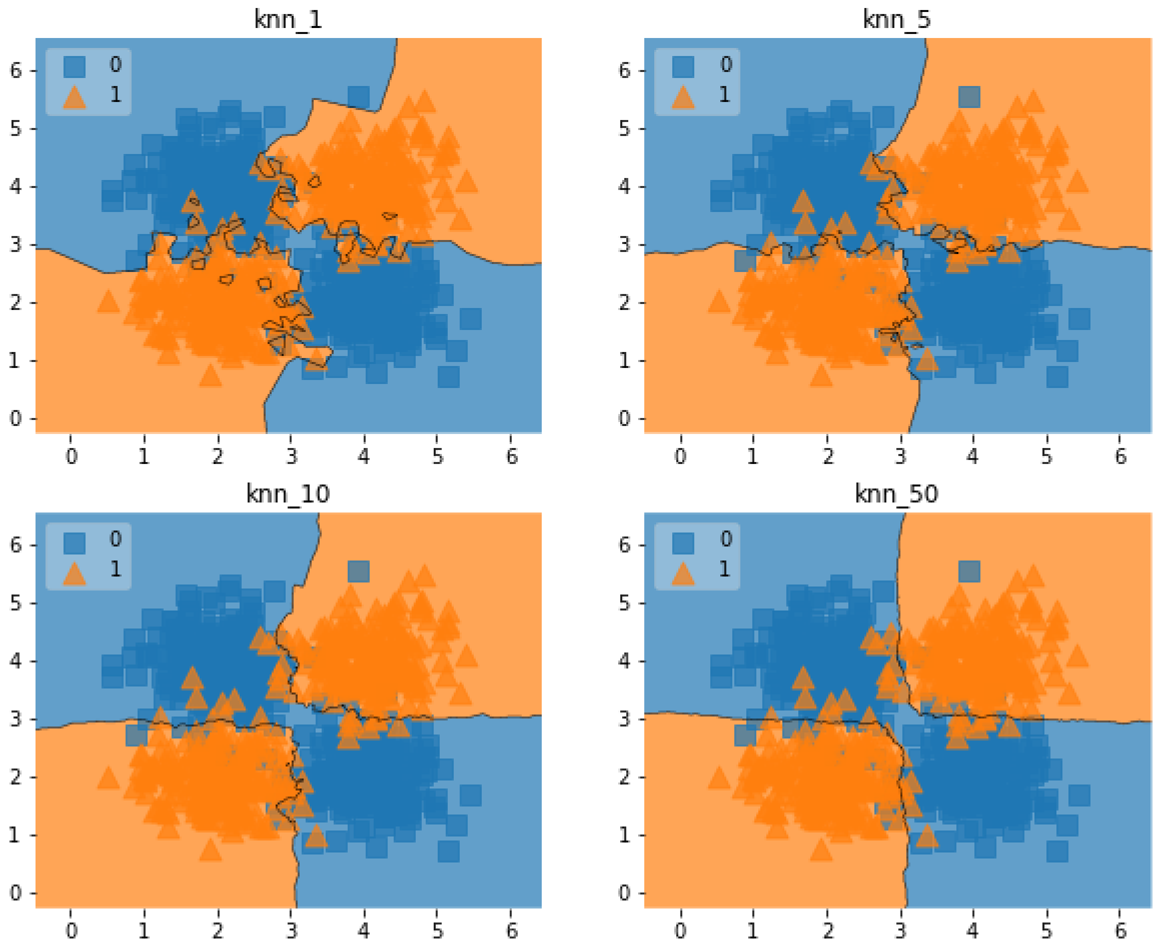
In [17]:
```
print(knn_accuracy_df.loc[knn_accuracy_df.Score.idxmin()])
```

```
Neighbours    1.0000
Score         0.9125
Name: 0, dtype: float64
```

Answer: Neighbour 1 has the lowest accuracy. This phenomenon is called overfitting

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

In [18]:
```
# Creating a list of classifiers and their names for plotting
clf_list = [knn1,knn5,knn10,knn50]
labels = ['knn_1','knn_5','knn_10','knn_50']

# Plotting the decision boundaries
fig = plt.figure(figsize=(10,8))
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.title(label)
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 neighbours are increasing the decision region is becoming more and more generalized.

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

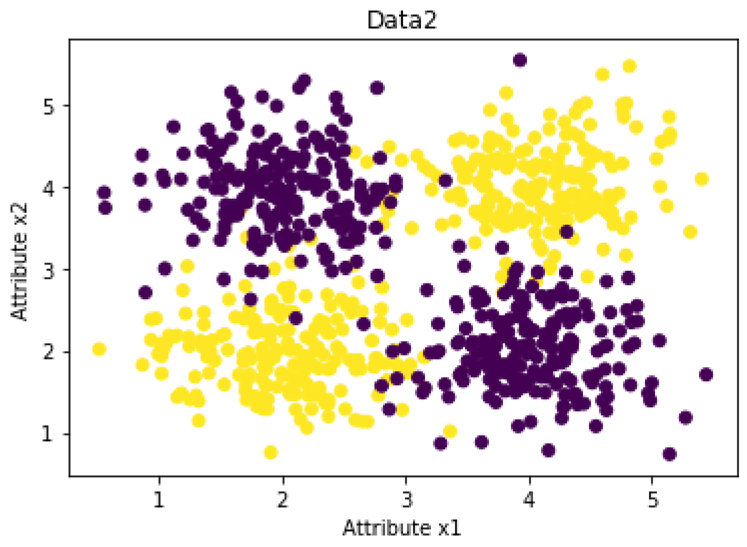
In [19]:
```
nb = GaussianNB()
```

In [20]:
```
nb_scores_1 = cross_val_score(nb, Data1_X, Data1_Y, cv=10, scoring='accuracy')
nb_scores_2 = cross_val_score(nb, Data2_X, Data2_Y, cv=10, scoring='accuracy')
nb_scores_3 = cross_val_score(nb, Data3_X, Data3_Y, cv=10, scoring='accuracy')
nb_scores_4 = cross_val_score(nb, Data4_X, Data4_Y, cv=10, scoring='accuracy')
```

In [21]:
```
nb_accuracy_df = pd.DataFrame.from_dict({ 'Dataset': [1,2,3,4],
    'Score': [nb_scores_1.mean(),nb_scores_2.mean(),nb_scores_3.mean(),nb_scor
es_4.mean()]})
print(nb_accuracy_df)
```

```
   Dataset     Score
0        1  0.967500
1        2  0.050000
2        3  0.960000
3        4  0.964074
```

In [22]:
```
plt.scatter(Data2_X[:,0],Data2_X[:,1], c= Data2_Y)
plt.xlabel('Attribute x1')
plt.ylabel('Attribute x2')
plt.title('Data2')
plt.show()
```

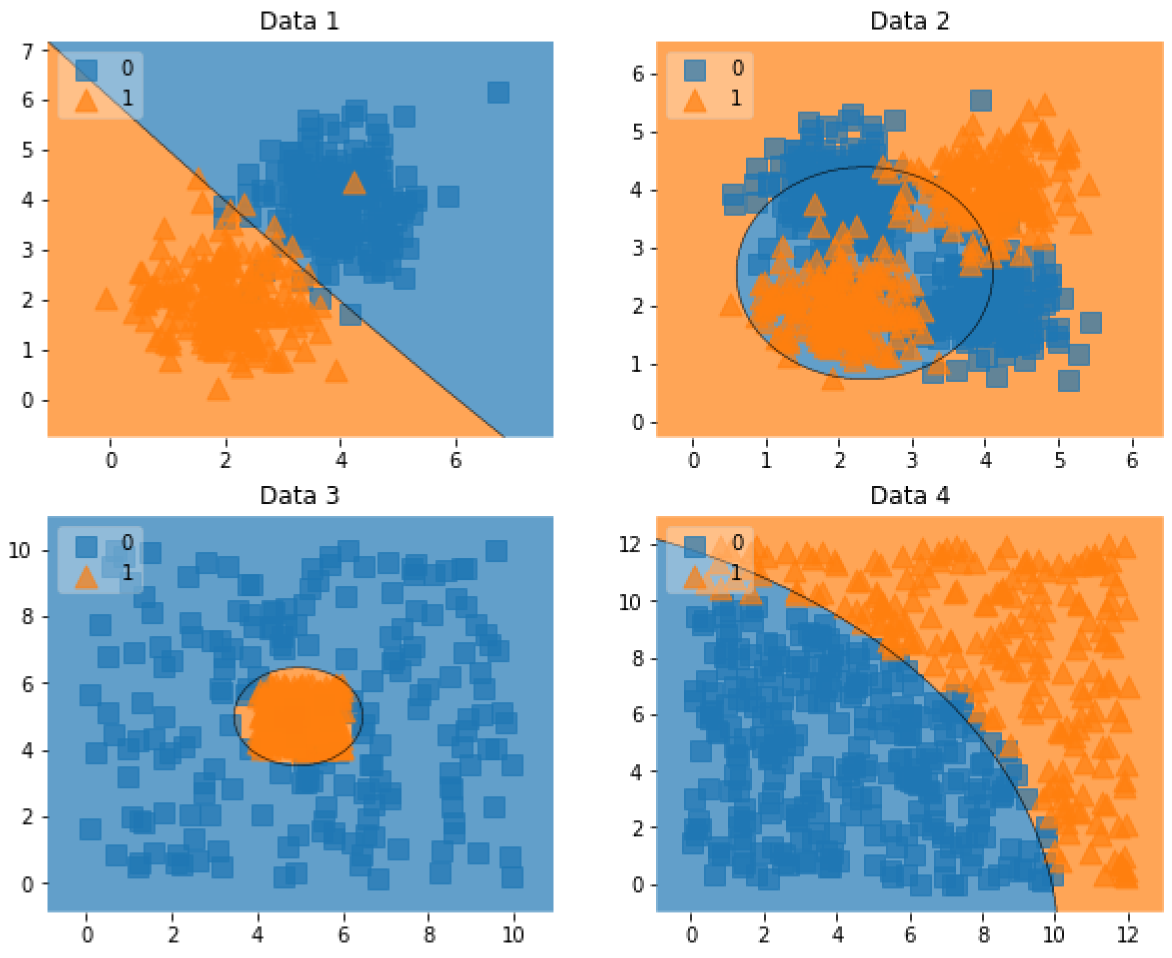


Question 3b: State your observations on the datasets the NB algorithm performed poorly.

Answer: As we know naive bayes forms a distribution for a class based on the training points, in the above case the ditributions for each class will overlap with one another, leading to low accuracy.

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

In [23]:
```
# Creating a list of classifiers and their names for plotting
data_list = [Data1_X,Data2_X,Data3_X,Data4_X]
label_list = [Data1_Y,Data2_Y,Data3_Y, Data4_Y]
labels = ['Data 1','Data 2','Data 3','Data 4']
# Plotting the decision boundaries
fig = plt.figure(figsize=(10,8))
count = 0;

for X, y, label in zip(data_list,label_list, labels):
    count = count + 1;
    nb.fit(X, y)
    ax = plt.subplot(2,2,count)
    fig = plot_decision_regions(X=X, y=y, clf=nb, legend=2,
                          scatter_kwargs=scatter_kwargs,
                          contourf_kwargs=contourf_kwargs,
                          scatter_highlight_kwargs=scatter_highlight_kwargs)

    plt.title(label)
plt.show()
```



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

**Answer:

1. Linear
2. Ellipsoidal
3. Ellipsoidal
4. curve

** Actually there're no decision regions per se, the points when classified to a class changes to the color of that class based on it's probability, which when seen cumulatively appears as a region. In general Naive Bayes only provides with the probablity of a class given a point label based on it's distribution

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

Answer: NB performs well when the data from different classes are well seperated meaning the distribution for the classes has zero to little overlap, if there is some overlap between the distribution of the classes, the performance of NB classifier suffers

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: 1. D1, 2. D4, 3. D3, 4.D2

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 [24]:
```
svm_linear = SVC(C=0.5, kernel='linear')
```

In [25]:
```
svm_scores_1 = cross_val_score(svm_linear, Data1_X, Data1_Y, cv=10, scoring='a
ccuracy')
svm_scores_2 = cross_val_score(svm_linear, Data2_X, Data2_Y, cv=10, scoring='a
ccuracy')
svm_scores_3 = cross_val_score(svm_linear, Data3_X, Data3_Y, cv=10, scoring='a
ccuracy')
svm_scores_4 = cross_val_score(svm_linear, Data4_X, Data4_Y, cv=10, scoring='a
ccuracy')
```

```
In [26]: svm_accuracy_df = pd.DataFrame.from_dict({ 'Dataset': [1,2,3,4],
             'Score': [svm_scores_1.mean(),svm_scores_2.mean(),svm_scores_3.mean(),svm_
         scores_4.mean()]})
         print(svm_accuracy_df)
```

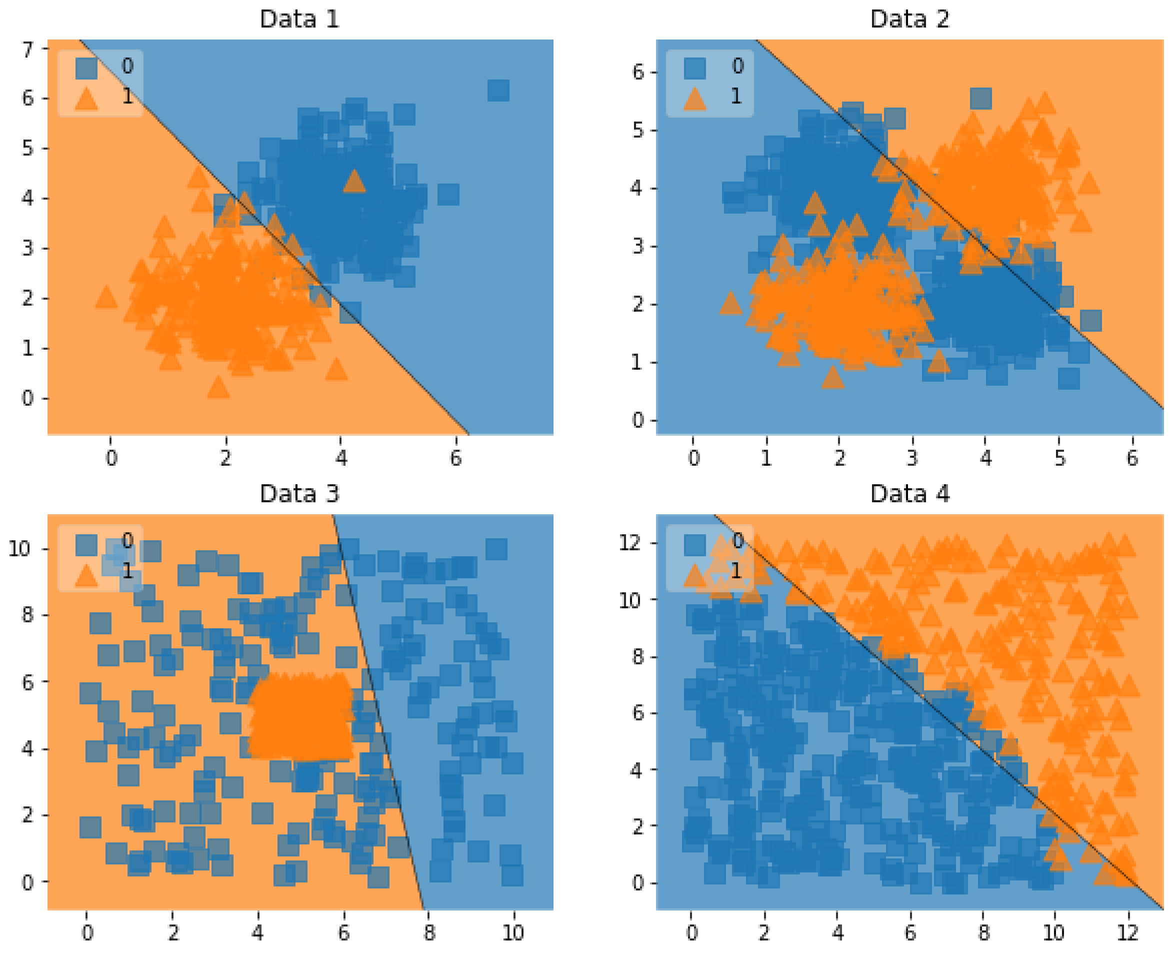
```
   Dataset     Score
0        1  0.967500
1        2  0.141250
2        3  0.642500
3        4  0.925946
```

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

Answer: 1. D1, 2. D4, 3. D3, 4.D2

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

In [27]:
```
# Creating a list of classifiers and their names for plotting
data_list = [Data1_X,Data2_X,Data3_X,Data4_X]
label_list = [Data1_Y,Data2_Y,Data3_Y, Data4_Y]
labels = ['Data 1','Data 2','Data 3','Data 4']
# Plotting the decision boundaries
fig = plt.figure(figsize=(10,8))
count = 0;

for X, y, label in zip(data_list,label_list, labels):
    count = count + 1;
    svm_linear.fit(X, y)
    ax = plt.subplot(2,2,count)
    fig = plot_decision_regions(X=X, y=y, clf=svm_linear, legend=2,
                        scatter_kwargs=scatter_kwargs,
                        contourf_kwargs=contourf_kwargs,
                        scatter_highlight_kwargs=scatter_highlight_kwargs)

    plt.title(label)
plt.show()
```



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

Answer: As we can see from the above plots, svm is able to do a clearer seperation on D1 compared to D2, where SVM fairs better than D4, for which svm although looks to have failed miserbaly is still better than SVM Linear classification on D3

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 [28]:
```
svm_poly = SVC(C=0.5, kernel='poly',degree=1, gamma = 'auto')
svm_poly_2 = SVC(C=0.5, kernel='poly',degree=2, gamma = 'auto')
svm_poly_3 = SVC(C=0.5, kernel='poly',degree=3, gamma = 'auto')
```

In [29]:
```
svm_poly_scores = cross_val_score(svm_poly, Data2_X, Data2_Y, cv=10, scoring=
'accuracy')
svm_poly_scores_2 = cross_val_score(svm_poly_2, Data2_X, Data2_Y, cv=10, scori
ng='accuracy')
svm_poly_scores_3 = cross_val_score(svm_poly_3, Data2_X, Data2_Y, cv=10, scori
ng='accuracy')
```

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

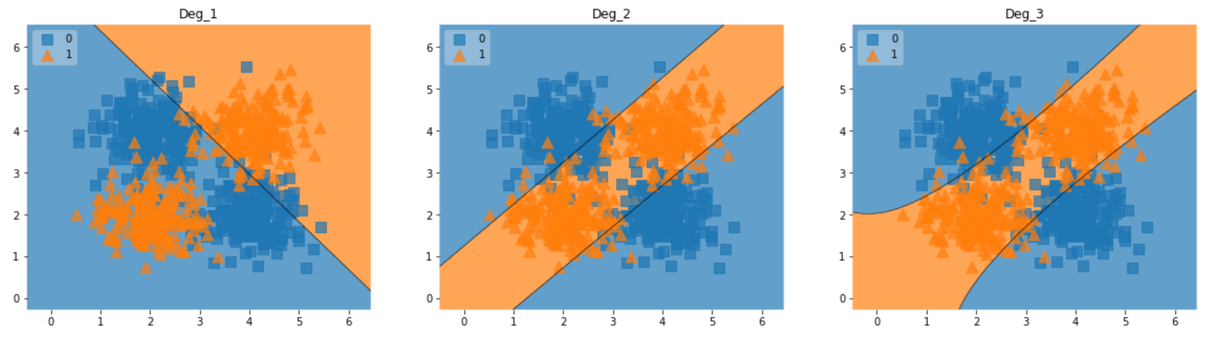
In [30]:
```
svm_poly_accuracy_df = pd.DataFrame.from_dict({ 'Order': [1,2,3],
    'Score': [svm_poly_scores.mean(),svm_poly_scores_2.mean(),svm_poly_scores_
3.mean()]})
print(svm_poly_accuracy_df)
```

```
   Order    Score
0      1  0.13375
1      2  0.86500
2      3  0.87625
```

Answer: Degree 3, Degree 2 and Degree 1

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

In [31]:
```
# Creating a list of classifiers and their names for plotting
clf_list = [svm_poly,svm_poly_2,svm_poly_3]
labels = ['Deg_1','Deg_2','Deg_3']

# Plotting the decision boundaries
fig = plt.figure(figsize=(20,5))
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.title(label)
plt.show()
```



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

Answer: SVM with degree 1 is a linear separator trying to separate linearly unseparable data hence it fails miserably. SVM with degree 2 perfoms better as it uses 2 linear separators to chart out a region for class 1. SVM with degree 3 performs even better because it uses curves to further refine the decision region

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 [32]:
```
svm_rbf_001 = SVC(C = 0.5, kernel='rbf', gamma=0.01)
svm_rbf_01 = SVC(C = 0.5, kernel='rbf', gamma=0.1)
svm_rbf_1 = SVC(C = 0.5, kernel='rbf', gamma=1)
```

In [33]:
```
svm_rbf_scores_001 = cross_val_score(svm_rbf_001, Data2_X, Data2_Y, cv=10, sco
ring='accuracy')
svm_rbf_scores_01 = cross_val_score(svm_rbf_01, Data2_X, Data2_Y, cv=10, scori
ng='accuracy')
svm_rbf_scores_1 = cross_val_score(svm_rbf_1, Data2_X, Data2_Y, cv=10, scoring
='accuracy')
```

In [34]:
```
svm_rbf_accuracy_df = pd.DataFrame.from_dict({ 'Order': ['001','01','1'],
    'Score': [svm_rbf_scores_001.mean(),svm_rbf_scores_01.mean(),svm_rbf_score
s_1.mean()]})
print(svm_rbf_accuracy_df)
```

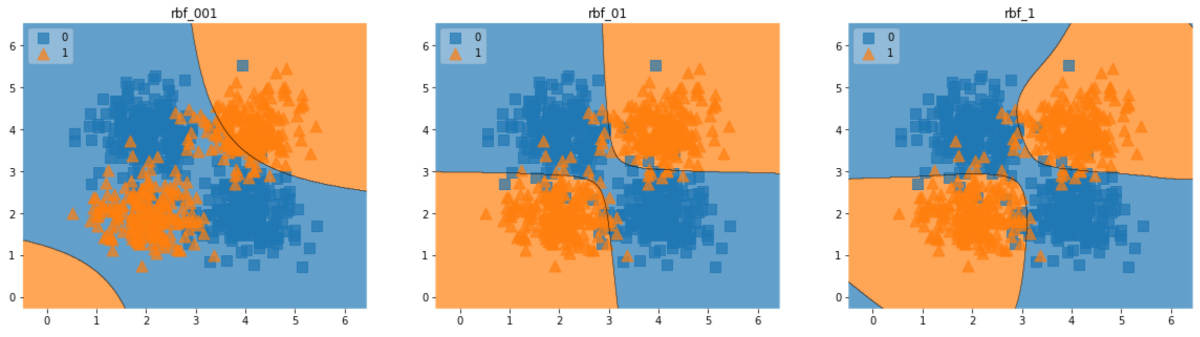
```
  Order    Score
0   001  0.30125
1    01  0.93625
2     1  0.94000
```

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

Answer: 1. svm_rbf_scores_1, 2. svm_rbf_scores_01, 3.svm_rbf_scores_001

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

In [35]:
```
# Creating a list of classifiers and their names for plotting
clf_list = [svm_rbf_001,svm_rbf_01,svm_rbf_1]
labels = ['rbf_001','rbf_01','rbf_1']

# Plotting the decision boundaries
fig = plt.figure(figsize=(20,5))
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.title(label)
plt.show()
```



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

Answer: 1. svm_rbf_.001 performs very badly because we tell the kernel that the variance between the points is very high and hence the rbf tries to account for it. 2. svm_rbf_.01 performs very well better because the variance set for the kernel appropriates the variance in the data 3. svm_rbf_1 also performs very well, infact marginally better than svm_rbf_1 because the decrease in variance fits the data better

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: svm_rbf_1 (i.e. gamma = 1) is better suited because it covers the pattern of points very well

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 [36]:
```
svm_rbf_scores_01_cv_3 = cross_val_score(svm_rbf_01, Data2_X, Data2_Y, cv=3, s
coring='accuracy')
svm_rbf_scores_01_cv_4 = cross_val_score(svm_rbf_01, Data2_X, Data2_Y, cv=4, s
coring='accuracy')
svm_rbf_scores_01_cv_5 = cross_val_score(svm_rbf_01, Data2_X, Data2_Y, cv=5, s
coring='accuracy')
svm_rbf_scores_01_cv_6 = cross_val_score(svm_rbf_01, Data2_X, Data2_Y, cv=6, s
coring='accuracy')
```

In [37]:
```
svm_rbf_accuracy_cv_df = pd.DataFrame.from_dict({ 'Order': ['cv_3','cv_4','cv_
5', 'cv_6'],
    'Score': [svm_rbf_scores_01_cv_3.mean(),svm_rbf_scores_01_cv_4.mean(),svm_
rbf_scores_01_cv_5.mean()
              ,svm_rbf_scores_01_cv_6.mean()]})
print(svm_rbf_accuracy_cv_df)
```

```
  Order     Score
0  cv_3  0.903827
1  cv_4  0.916250
2  cv_5  0.927500
3  cv_6  0.932553
```

Answer: This may be due to the increase in training data points the more the data points used to trained the model, the better the model 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.

In [38]:
```
dt4_accuracy_score = cross_val_score(dt4, Data3_X, Data3_Y, cv=30, scoring='ac
curacy')
nb_accuracy_score = cross_val_score(nb, Data3_X, Data3_Y, cv=30, scoring='accu
racy')
knn5_accuracy_score = cross_val_score(knn5, Data3_X, Data3_Y, cv=30, scoring=
'accuracy')
svm_linear_accuracy_score = cross_val_score(svm_linear, Data3_X, Data3_Y, cv=3
0, scoring='accuracy')
svm_poly_accuracy_score = cross_val_score(svm_poly_3, Data3_X, Data3_Y, cv=30,
scoring='accuracy')
svm_rbf_accuracy_score = cross_val_score(svm_rbf_1, Data3_X, Data3_Y, cv=30, s
coring='accuracy')
```

In [39]:
```
dt4_precision_score = cross_val_score(dt4, Data3_X, Data3_Y, cv=30, scoring='p
recision')
nb_precision_score = cross_val_score(nb, Data3_X, Data3_Y, cv=30, scoring='pre
cision')
knn5_precision_score = cross_val_score(knn5, Data3_X, Data3_Y, cv=30, scoring=
'precision')
svm_linear_precision_score = cross_val_score(svm_linear, Data3_X, Data3_Y, cv=
30, scoring='precision')
svm_poly_precision_score = cross_val_score(svm_poly_3, Data3_X, Data3_Y, cv=30
, scoring='precision')
svm_rbf_precision_score = cross_val_score(svm_rbf_1, Data3_X, Data3_Y, cv=30,
scoring='precision')
```

In [40]:
```
classifier_rank_accuracy_df = pd.DataFrame.from_dict({ 'Classifier': ['dt','n
b','knn', 'svm_linear', 'svm_poly', 'svm_rbf'],
    'Score': [dt4_accuracy_score.mean(),nb_accuracy_score.mean(),knn5_accuracy
_score.mean()
              ,svm_linear_accuracy_score.mean(),svm_poly_accuracy_score.mean
(),svm_rbf_accuracy_score.mean()]})
print(classifier_rank_accuracy_df)
```

```
   Classifier     Score
0          dt  0.971825
1          nb  0.959127
2         knn  0.949206
3  svm_linear  0.642857
4    svm_poly  0.855556
5     svm_rbf  0.954365
```

In [41]:
```
classifier_rank_precision_df = pd.DataFrame.from_dict({ 'Classifier': ['dt','n
b','knn', 'svm_linear', 'svm_poly', 'svm_rbf'],
    'Score': [knn5_precision_score.mean(),nb_precision_score.mean(),knn5_preci
sion_score.mean()
              ,svm_linear_precision_score.mean(),svm_poly_precision_score.mean
(),svm_rbf_precision_score.mean()]})
print(classifier_rank_accuracy_df)
```

```
   Classifier     Score
0          dt  0.971825
1          nb  0.959127
2         knn  0.949206
3  svm_linear  0.642857
4    svm_poly  0.855556
5     svm_rbf  0.954365
```

Answer: Yes in both cases decision tree with max depth = 4 is the best classifier

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 [42]:
```
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("Bagging Accuracy: %.2f (+/- %.2f) #estimators: %d" % (scores.mean
(), scores.std(), n_est))
```

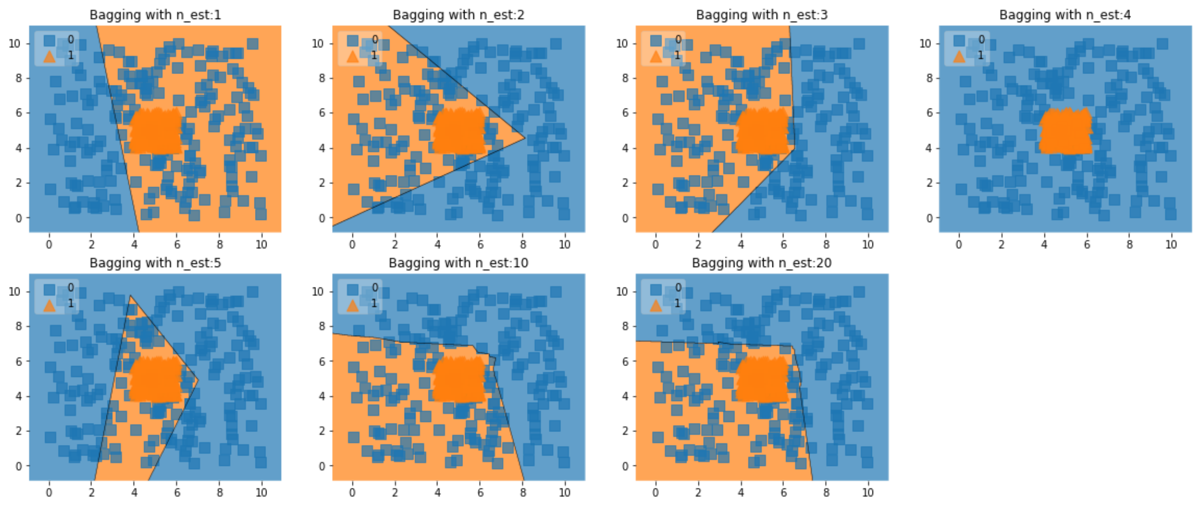
```
Bagging Accuracy: 0.58 (+/- 0.07) #estimators: 1
Bagging Accuracy: 0.57 (+/- 0.08) #estimators: 2
Bagging Accuracy: 0.64 (+/- 0.04) #estimators: 3
Bagging Accuracy: 0.59 (+/- 0.08) #estimators: 4
Bagging Accuracy: 0.62 (+/- 0.04) #estimators: 5
Bagging Accuracy: 0.75 (+/- 0.11) #estimators: 10
Bagging Accuracy: 0.68 (+/- 0.07) #estimators: 20
```

Answer: Accuracy Mean and SD is almost same for estimators 1,2 and 4. Esimators 3 and 4 are also interms of SD and Mean. Bagging with 10 estimators show the highest accuracy and higher standar deviation suggesting that although the mean accuracy is high the spread of the accuracy is large compared to other estimators. Bagging with 20 estimators have mean accuracy and sd little less compared to estimator 20, suggesting the effect of some kind of overfitting of training data on the model

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

In [43]:
```
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()
```

```
/users/PES0801/nifaullah/.local/lib/python3.6/site-packages/mlxtend/plotting/
decision_regions.py:247: 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 region with many estimators (say 10 or 20) is definitely better than the qualtiy of decision region with 1 estimator, implying that model is more refined as the number of sample increases.

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.

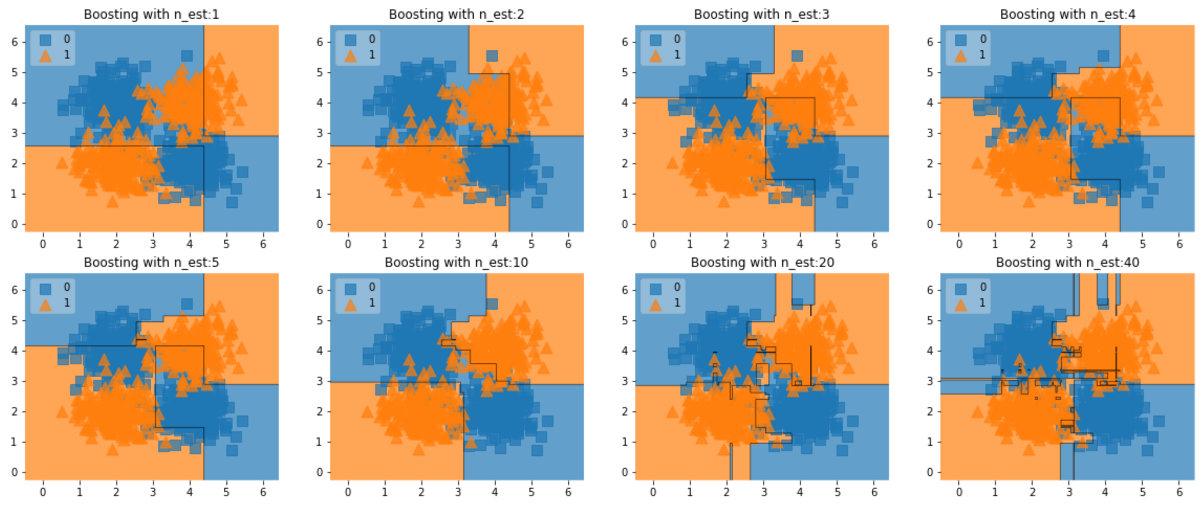
In [44]:
```
dt = DecisionTreeClassifier(max_depth=2)
n_est_list = [1,2,3,4,5,10,20,40]
for n_est in n_est_list:
    # create an instance of a boosting classifier with 'n_est' estimators
    boosting = AdaBoostClassifier(base_estimator=dt, n_estimators=n_est)
    # compute cross-validation accuracy for each bagging classifier
    scores = cross_val_score(boosting, Data2_X, Data2_Y, cv=10, scoring='accur
acy')
    print("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 (+/- 0.04) #estimators: 20
Boosting Accuracy: 0.91 (+/- 0.02) #estimators: 40
```

Answer: Boosting accuracy is little low for estimators 1,2,3,4 because it likely underfits the model whereas boosting accuracy for estimators 20 and 40 is little lower because of likely overfitting. Estimators 5 and 10 rightly generalize the decision region to fit the model approximate enough.

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

In [45]:
```
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: From the above plot we can boosting accuracy is little low for estimators 1,2,3,4 because it underfits the model whereas boosting accuracy for estimators 20 and 40 is little lower because of overfitting. Estimators 5 and 10 rightly generalize the decision region.

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 [46]:
```
from sklearn import datasets
cancer = datasets.load_breast_cancer()
```

The features are:

```
In [47]: cancer.feature_names
```

```
Out[47]: 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 [48]: cancer.target_names
```

```
Out[48]: array(['malignant', 'benign'], dtype='<U9')
```

Create dataset for classification

```
In [49]: X = cancer.data
         Y = cancer.target
```

Number of samples are:

```
In [50]: X.shape
```

```
Out[50]: (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 [51]: #Linear SVM
         svm_linear_c_0001 = SVC(C=0.001, kernel='linear')
         svm_linear_c_001 = SVC(C=0.01, kernel='linear')
         svm_linear_c_01 = SVC(C=0.1, kernel='linear')
         svm_linear_c_1 = SVC(C=1, kernel='linear')
```

```
In [52]: svm_linear_scores_c_0001 = cross_val_score(svm_linear_c_0001, X, Y, cv=10, sco
         ring='accuracy')
```

```
In [53]: svm_linear_scores_c_001 = cross_val_score(svm_linear_c_001, X, Y, cv=10, scori
         ng='accuracy')
```

In [54]:
```
svm_linear_scores_c_01 = cross_val_score(svm_linear_c_01, X, Y, cv=10, scoring
='accuracy')
```

In [55]:
```
svm_linear_scores_c_1 = cross_val_score(svm_linear_c_1, X, Y, cv=10, scoring=
'accuracy')
```

In [56]:
```
svm_linear_accuracy_cancer = pd.DataFrame.from_dict({ 'C': ['0.001','0.01','0.
1','1'],
                                                  'Score': [svm_linear_scores_
c_0001.mean(),
                                                            svm_linear_scores_
c_001.mean(),
                                                            svm_linear_scores_
c_01.mean(),
                                                            svm_linear_scores_
c_1.mean()]})
print(svm_linear_accuracy_cancer)
```

```
       C     Score
0  0.001  0.940246
1   0.01  0.947236
2    0.1  0.947237
3      1  0.954318
```

In [57]:
```
#Polynomial varying C and degree
#RBF varying C and gamma
svm_poly_1_c_0001 = SVC(C = 0.001, kernel='poly', degree=1, gamma = 'auto')
svm_poly_1_c_001 = SVC(C = 0.01, kernel='poly', degree=1, gamma = 'auto')
svm_poly_1_c_01 = SVC(C = 0.1, kernel='poly', degree=1, gamma = 'auto')
svm_poly_1_c_1 = SVC(C = 1, kernel='poly', degree=1, gamma = 'auto')
svm_poly_2_c_0001 = SVC(C = 0.001, kernel='poly', degree=2, gamma = 'auto')
svm_poly_2_c_001 = SVC(C = 0.01, kernel='poly', degree=2, gamma = 'auto')
svm_poly_2_c_01 = SVC(C = 0.1, kernel='poly', degree=2, gamma = 'auto')
svm_poly_2_c_1 = SVC(C = 1, kernel='poly', degree=2, gamma = 'auto')
```

In [58]:
```
svm_poly_scores_1_c_0001 = cross_val_score(svm_poly_1_c_0001, X, Y, cv=10, sco
ring='accuracy')
```

In [59]:
```
svm_poly_scores_1_c_001 = cross_val_score(svm_poly_1_c_001, X, Y, cv=10, scori
ng='accuracy')
```

In [60]:
```
svm_poly_scores_1_c_01 = cross_val_score(svm_poly_1_c_01, X, Y, cv=10, scoring
='accuracy')
```

In [61]:
```
svm_poly_scores_1_c_1 = cross_val_score(svm_poly_1_c_1, X, Y, cv=10, scoring=
'accuracy')
```

In [62]:
```
svm_poly_scores_2_c_0001 = cross_val_score(svm_poly_2_c_0001, X, Y, cv=10, sco
ring='accuracy')
```

In [63]:
```
svm_poly_scores_2_c_001 = cross_val_score(svm_poly_2_c_001, X, Y, cv=10, scori
ng='accuracy')
```

In [64]:
```
svm_poly_scores_2_c_01 = cross_val_score(svm_poly_2_c_01, X, Y, cv=10, scoring
='accuracy')
```

In [65]:
```
svm_poly_scores_2_c_1 = cross_val_score(svm_poly_2_c_1, X, Y, cv=10, scoring=
'accuracy')
```

In [66]:
```
svm_poly_accuracy_cancer = pd.DataFrame.from_dict({ 'Degree': ['1','1','1','1'
,'2','2','2','2'],
                                                  'C': ['.001','.01','.1','1',
'.001','.01','.1','1'],
                                                  'Score': [svm_poly_scores_1_
c_0001.mean(),
                                                            svm_poly_scores_1_
c_001.mean(),
                                                            svm_poly_scores_1_
c_01.mean(),
                                                            svm_poly_scores_1_
c_1.mean(),
                                                            svm_poly_scores_2_
c_0001.mean(),
                                                            svm_poly_scores_2_
c_001.mean(),
                                                            svm_poly_scores_2_
c_01.mean(),
                                                            svm_poly_scores_2_
c_1.mean()]})
print(svm_poly_accuracy_cancer)
```

```
  Degree     C     Score
0      1  .001  0.928027
1      1   .01  0.940278
2      1    .1  0.947267
3      1     1  0.947267
4      2  .001  0.949085
5      2   .01  0.957920
6      2    .1  0.957920
7      2     1  0.957951
```

In [67]:
```
#RBF varying C and gamma
svm_rbf_0001_c_0001 = SVC(C = 0.001, kernel='rbf', gamma=0.001)
svm_rbf_0001_c_001 = SVC(C = 0.01, kernel='rbf', gamma=0.001)
svm_rbf_0001_c_01 = SVC(C = 0.1, kernel='rbf', gamma=0.001)
svm_rbf_0001_c_1 = SVC(C = 1, kernel='rbf', gamma=0.001)

svm_rbf_001_c_0001 = SVC(C = 0.001, kernel='rbf', gamma=0.01)
svm_rbf_001_c_001 = SVC(C = 0.01, kernel='rbf', gamma=0.01)
svm_rbf_001_c_01 = SVC(C = 0.1, kernel='rbf', gamma=0.01)
svm_rbf_001_c_1 = SVC(C = 1, kernel='rbf', gamma=0.01)

svm_rbf_01_c_0001 = SVC(C = 0.001, kernel='rbf', gamma=0.1)
svm_rbf_01_c_001 = SVC(C = 0.01, kernel='rbf', gamma=0.1)
svm_rbf_01_c_01 = SVC(C = 0.1, kernel='rbf', gamma=0.1)
svm_rbf_01_c_1 = SVC(C = 1, kernel='rbf', gamma=0.1)

svm_rbf_1_c_0001 = SVC(C = 0.001, kernel='rbf', gamma=1)
svm_rbf_1_c_001 = SVC(C = 0.01, kernel='rbf', gamma=1)
svm_rbf_1_c_01 = SVC(C = 0.1, kernel='rbf', gamma=1)
svm_rbf_1_c_1 = SVC(C = 1, kernel='rbf', gamma=1)

svm_rbf_10_c_0001 = SVC(C = 0.001, kernel='rbf', gamma=10)
svm_rbf_10_c_001 = SVC(C = 0.01, kernel='rbf', gamma=10)
svm_rbf_10_c_01 = SVC(C = 0.1, kernel='rbf', gamma=10)
svm_rbf_10_c_1 = SVC(C = 1, kernel='rbf', gamma=10)

svm_rbf_100_c_0001 = SVC(C = 0.001, kernel='rbf', gamma=100)
svm_rbf_100_c_001 = SVC(C = 0.01, kernel='rbf', gamma=100)
svm_rbf_100_c_01 = SVC(C = 0.1, kernel='rbf', gamma=100)
svm_rbf_100_c_1 = SVC(C = 1, kernel='rbf', gamma=100)
```

In [68]:
```
svm_rbf_scores_0001_c_0001 = cross_val_score(svm_rbf_0001_c_0001, X, Y, cv=10,
scoring='accuracy')
```

In [69]:
```
svm_rbf_scores_0001_c_001 = cross_val_score(svm_rbf_0001_c_001, X, Y, cv=10, s
coring='accuracy')
```

In [70]:
```
svm_rbf_scores_0001_c_01 = cross_val_score(svm_rbf_0001_c_01, X, Y, cv=10, sco
ring='accuracy')
```

In [71]:
```
svm_rbf_scores_0001_c_1 = cross_val_score(svm_rbf_0001_c_1, X, Y, cv=10, scori
ng='accuracy')
```

In [72]:
```
svm_rbf_scores_001_c_0001 = cross_val_score(svm_rbf_001_c_0001, X, Y, cv=10, s
coring='accuracy')
```

In [73]:
```
svm_rbf_scores_001_c_001 = cross_val_score(svm_rbf_001_c_001, X, Y, cv=10, sco
ring='accuracy')
```

In [74]:
```
svm_rbf_scores_001_c_01 = cross_val_score(svm_rbf_001_c_01, X, Y, cv=10, scori
ng='accuracy')
```

In [75]:
```
svm_rbf_scores_001_c_1 = cross_val_score(svm_rbf_001_c_1, X, Y, cv=10, scoring
='accuracy')
```

In [76]:
```
svm_rbf_scores_01_c_0001 = cross_val_score(svm_rbf_01_c_0001, X, Y, cv=10, sco
ring='accuracy')
```

In [77]:
```
svm_rbf_scores_01_c_001 = cross_val_score(svm_rbf_01_c_001, X, Y, cv=10, scori
ng='accuracy')
```

In [78]:
```
svm_rbf_scores_01_c_01 = cross_val_score(svm_rbf_01_c_01, X, Y, cv=10, scoring
='accuracy')
```

In [79]:
```
svm_rbf_scores_01_c_1 = cross_val_score(svm_rbf_01_c_1, X, Y, cv=10, scoring=
'accuracy')
```

In [80]:
```
svm_rbf_scores_1_c_0001 = cross_val_score(svm_rbf_1_c_0001, X, Y, cv=10, scori
ng='accuracy')
```

In [81]:
```
svm_rbf_scores_1_c_001 = cross_val_score(svm_rbf_1_c_001, X, Y, cv=10, scoring
='accuracy')
```

In [82]:
```
svm_rbf_scores_1_c_01 = cross_val_score(svm_rbf_1_c_01, X, Y, cv=10, scoring=
'accuracy')
```

In [83]:
```
svm_rbf_scores_1_c_1 = cross_val_score(svm_rbf_1_c_1, X, Y, cv=10, scoring='ac
curacy')
```

In [84]:
```
svm_rbf_scores_10_c_0001 = cross_val_score(svm_rbf_10_c_0001, X, Y, cv=10, sco
ring='accuracy')
```

In [85]:
```
svm_rbf_scores_10_c_001 = cross_val_score(svm_rbf_10_c_001, X, Y, cv=10, scori
ng='accuracy')
```

In [86]:
```
svm_rbf_scores_10_c_01 = cross_val_score(svm_rbf_10_c_01, X, Y, cv=10, scoring
='accuracy')
```

In [87]:
```
svm_rbf_scores_10_c_1 = cross_val_score(svm_rbf_10_c_1, X, Y, cv=10, scoring=
'accuracy')
```

In [88]:
```
svm_rbf_scores_100_c_0001 = cross_val_score(svm_rbf_100_c_0001, X, Y, cv=10, s
coring='accuracy')
```

In [89]:
```
svm_rbf_scores_100_c_001 = cross_val_score(svm_rbf_100_c_001, X, Y, cv=10, sco
ring='accuracy')
```

In [90]:
```
svm_rbf_scores_100_c_01 = cross_val_score(svm_rbf_100_c_01, X, Y, cv=10, scori
ng='accuracy')
```

In [91]:
```
svm_rbf_scores_100_c_1 = cross_val_score(svm_rbf_100_c_1, X, Y, cv=10, scoring
='accuracy')
```

In [92]:
```
svm_rbf_accuracy_cancer = pd.DataFrame.from_dict({ 'Gamma': ['.001','.001','.0
01','.001',
                                                              '.01','.01','.01'
,'.01',
                                                              '.1','.1','.1','.
1',
                                                              '1','1','1','1',
                                                              '10','10','10','1
0',
                                                              '100','100','100'
,'100'],
                                                   'C': ['.001','.01','.1','1',
                                                         '.001','.01','.1','1',
                                                         '.001','.01','.1','1',
                                                         '.001','.01','.1','1',
                                                         '.001','.01','.1','1',
                                                         '.001','.01','.1','1'
],
                                                   'Score': [
                                                             svm_rbf_scores_000
1_c_0001.mean(),
                                                             svm_rbf_scores_000
1_c_001.mean(),
                                                             svm_rbf_scores_000
1_c_01.mean(),
                                                             svm_rbf_scores_000
1_c_1.mean(),
                                                             svm_rbf_scores_000
1_c_0001.mean(),
                                                             svm_rbf_scores_001
_c_001.mean(),
                                                             svm_rbf_scores_001
_c_01.mean(),
                                                             svm_rbf_scores_001
_c_1.mean(),
                                                             svm_rbf_scores_01_
c_0001.mean(),
                                                             svm_rbf_scores_01_
c_001.mean(),
                                                             svm_rbf_scores_01_
c_01.mean(),
                                                             svm_rbf_scores_01_
c_1.mean(),
                                                             svm_rbf_scores_1_c
_0001.mean(),
                                                             svm_rbf_scores_1_c
_001.mean(),
                                                             svm_rbf_scores_1_c
_01.mean(),
                                                             svm_rbf_scores_1_c
_1.mean(),
                                                             svm_rbf_scores_10_
c_0001.mean(),
                                                             svm_rbf_scores_10_
c_001.mean(),
                                                             svm_rbf_scores_10_
c_01.mean(),
                                                            svm_rbf_scores_10_
c_1.mean(),
                                                            svm_rbf_scores_100
_c_0001.mean(),
                                                            svm_rbf_scores_100
_c_001.mean(),
                                                            svm_rbf_scores_100
_c_01.mean(),
                                                            svm_rbf_scores_100
_c_1.mean()]})
print(svm_rbf_accuracy_cancer)
```

```
   Gamma     C     Score
0   .001  .001  0.627427
1   .001   .01  0.627427
2   .001    .1  0.627427
3   .001     1  0.924242
4    .01  .001  0.627427
5    .01   .01  0.627427
6    .01    .1  0.627427
7    .01     1  0.632660
8     .1  .001  0.627427
9     .1   .01  0.627427
10    .1    .1  0.627427
11    .1     1  0.627427
12     1  .001  0.627427
13     1   .01  0.627427
14     1    .1  0.627427
15     1     1  0.627427
16    10  .001  0.627427
17    10   .01  0.627427
18    10    .1  0.627427
19    10     1  0.627427
20   100  .001  0.627427
21   100   .01  0.627427
22   100    .1  0.627427
23   100     1  0.627427
```

Answer: SVM Polynomial with degree 2 has the best accuracy with .977951. Generally SVM Polynomial classifier seems to be performing better on this data with SVM Linear a close second. SVM RBF performance appears to be very substandard

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

```
In [93]: dt2_score_cancer = cross_val_score(dt2, X, Y, cv=10, scoring='accuracy')
```

```
In [94]: dt4_score_cancer = cross_val_score(dt4, X, Y, cv=10, scoring='accuracy')
```

```
In [95]: dt6_score_cancer = cross_val_score(dt6, X, Y, cv=10, scoring='accuracy')
In [96]: dt8_score_cancer = cross_val_score(dt8, X, Y, cv=10, scoring='accuracy')
```

```
In [97]: dt10_score_cancer = cross_val_score(dt10, X, Y, cv=10, scoring='accuracy')
```

```
In [98]: dt50_score_cancer = cross_val_score(dt50, X, Y, cv=10, scoring='accuracy')
```

```
In [100]: dt_accuracy_cancer = { 'Depth': [2,4,6,8,10,50],
              'Score': [dt2_score_cancer.mean(),dt4_score_cancer.mean(),dt6_score_cancer
          .mean(),dt8_score_cancer.mean(),
                        dt10_score_cancer.mean(), dt50_score.mean()]
                             }
          dt_accuracy_df_cancer = pd.DataFrame.from_dict(dt_accuracy_cancer)
          print(dt_accuracy_df_cancer)

             Depth     Score
          0      2  0.919378
          1      4  0.922858
          2      6  0.919442
          3      8  0.912330
          4     10  0.917656
          5     50  0.945000
```

Answer: DT with MaxDepth = 50 is the best performer on the data

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?

```
In [104]: cancer.data

Out[104]: array([[1.799e+01, 1.038e+01, 1.228e+02, ..., 2.654e-01, 4.601e-01,
                  1.189e-01],
                 [2.057e+01, 1.777e+01, 1.329e+02, ..., 1.860e-01, 2.750e-01,
                  8.902e-02],
                 [1.969e+01, 2.125e+01, 1.300e+02, ..., 2.430e-01, 3.613e-01,
                  8.758e-02],
                 ...,
                 [1.660e+01, 2.808e+01, 1.083e+02, ..., 1.418e-01, 2.218e-01,
                  7.820e-02],
                 [2.060e+01, 2.933e+01, 1.401e+02, ..., 2.650e-01, 4.087e-01,
                  1.240e-01],
                 [7.760e+00, 2.454e+01, 4.792e+01, ..., 0.000e+00, 2.871e-01,
                  7.039e-02]])
```

```
In [106]: cancer_df = pd.DataFrame(data=cancer.data,  columns=cancer.feature_names)
```

In [111]:
```
cancer_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 30 columns):
mean radius                569 non-null float64
mean texture               569 non-null float64
mean perimeter             569 non-null float64
mean area                  569 non-null float64
mean smoothness            569 non-null float64
mean compactness           569 non-null float64
mean concavity             569 non-null float64
mean concave points        569 non-null float64
mean symmetry              569 non-null float64
mean fractal dimension     569 non-null float64
radius error               569 non-null float64
texture error              569 non-null float64
perimeter error            569 non-null float64
area error                 569 non-null float64
smoothness error           569 non-null float64
compactness error          569 non-null float64
concavity error            569 non-null float64
concave points error       569 non-null float64
symmetry error             569 non-null float64
fractal dimension error    569 non-null float64
worst radius               569 non-null float64
worst texture              569 non-null float64
worst perimeter            569 non-null float64
worst area                 569 non-null float64
worst smoothness           569 non-null float64
worst compactness          569 non-null float64
worst concavity            569 non-null float64
worst concave points       569 non-null float64
worst symmetry             569 non-null float64
worst fractal dimension    569 non-null float64
dtypes: float64(30)
memory usage: 133.5 KB
```

```
In [108]: cancer_df.corr()
```

Out[108]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	co
mean radius	1.000000	0.323782	0.997855	0.987357	0.170581	0.506124	0.6
mean texture	0.323782	1.000000	0.329533	0.321086	-0.023389	0.236702	0.3
mean perimeter	0.997855	0.329533	1.000000	0.986507	0.207278	0.556936	0.7
mean area	0.987357	0.321086	0.986507	1.000000	0.177028	0.498502	0.6
mean smoothness	0.170581	-0.023389	0.207278	0.177028	1.000000	0.659123	0.5
mean compactness	0.506124	0.236702	0.556936	0.498502	0.659123	1.000000	0.8
mean concavity	0.676764	0.302418	0.716136	0.685983	0.521984	0.883121	1.0
mean concave points	0.822529	0.293464	0.850977	0.823269	0.553695	0.831135	0.9
mean symmetry	0.147741	0.071401	0.183027	0.151293	0.557775	0.602641	0.5
mean fractal dimension	-0.311631	-0.076437	-0.261477	-0.283110	0.584792	0.565369	0.3
radius error	0.679090	0.275869	0.691765	0.732562	0.301467	0.497473	0.6
texture error	-0.097317	0.386358	-0.086761	-0.066280	0.068406	0.046205	0.0
perimeter error	0.674172	0.281673	0.693135	0.726628	0.296092	0.548905	0.6
area error	0.735864	0.259845	0.744983	0.800086	0.246552	0.455653	0.6
smoothness error	-0.222600	0.006614	-0.202694	-0.166777	0.332375	0.135299	0.0
compactness error	0.206000	0.191975	0.250744	0.212583	0.318943	0.738722	0.6
concavity error	0.194204	0.143293	0.228082	0.207660	0.248396	0.570517	0.6
concave points error	0.376169	0.163851	0.407217	0.372320	0.380676	0.642262	0.6
symmetry error	-0.104321	0.009127	-0.081629	-0.072497	0.200774	0.229977	0.1

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	co
fractal dimension error	-0.042641	0.054458	-0.005523	-0.019887	0.283607	0.507318	0.4
worst radius	0.969539	0.352573	0.969476	0.962746	0.213120	0.535315	0.6
worst texture	0.297008	0.912045	0.303038	0.287489	0.036072	0.248133	0.2
worst perimeter	0.965137	0.358040	0.970387	0.959120	0.238853	0.590210	0.7
worst area	0.941082	0.343546	0.941550	0.959213	0.206718	0.509604	0.6
worst smoothness	0.119616	0.077503	0.150549	0.123523	0.805324	0.565541	0.4
worst compactness	0.413463	0.277830	0.455774	0.390410	0.472468	0.865809	0.7
worst concavity	0.526911	0.301025	0.563879	0.512606	0.434926	0.816275	0.8
worst concave points	0.744214	0.295316	0.771241	0.722017	0.503053	0.815573	0.8
worst symmetry	0.163953	0.105008	0.189115	0.143570	0.394309	0.510223	0.4
worst fractal dimension	0.007066	0.119205	0.051019	0.003738	0.499316	0.687382	0.5

30 rows × 30 columns

In [112]:
```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
```

In [113]:
```
#apply SelectKBest class to extract top 10 best features
bestfeatures = SelectKBest(score_func=chi2, k=10)
fit = bestfeatures.fit(X,Y)
dfscores = pd.DataFrame(fit.scores_)
dfcolumns = pd.DataFrame(cancer.feature_names)
#concat two dataframes for better visualization
featureScores = pd.concat([dfcolumns,dfscores],axis=1)
featureScores.columns = ['Specs','Score']  #naming the dataframe columns
print(featureScores.nlargest(10,'Score'))  #print 10 best features
```

```
              Specs          Score
23       worst area  112598.431564
3         mean area   53991.655924
13       area error    8758.504705
22  worst perimeter    3665.035416
2    mean perimeter    2011.102864
20     worst radius     491.689157
0       mean radius     266.104917
12  perimeter error     250.571896
21    worst texture     174.449400
1      mean texture      93.897508
```

**Answer:

I can use the above technique i.e. univariate selection find out the the top 10 features which influence the prediction. source(https://towardsdatascience.com/feature-selection-techniques-in-machine-learning-with-python-f24e7da3f36e)

For classification I would use SVM Poly of Deg 2 with C = 1 since it has the best accuracy rate although the execution is very slow, as it would reduce the attribution of Cancer, which again if wrongly attributed would cause immense trauma to the patient either ways
**