Hands-on Exercise CLASS Module

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

```
Requirement already satisfied: mlxtend in ./.local/lib/python3.6/site-package
s
Requirement already satisfied: matplotlib>=3.0.0 in ./.local/lib/python3.6/si
te-packages (from mlxtend)
Requirement already satisfied: numpy>=1.16.2 in ./.local/lib/python3.6/site-p
ackages (from mlxtend)
Requirement already satisfied: pandas>=0.24.2 in ./.local/lib/python3.6/site-
packages (from mlxtend)
Requirement already satisfied: scikit-learn>=0.20.3 in ./.local/lib/python3.
6/site-packages (from mlxtend)
Requirement already satisfied: joblib>=0.13.2 in ./.local/lib/python3.6/site-
packages (from mlxtend)
Requirement already satisfied: setuptools in /usr/local/anaconda5/lib/python
3.6/site-packages (from mlxtend)
Requirement already satisfied: scipy>=1.2.1 in ./.local/lib/python3.6/site-pa
ckages (from mlxtend)
Requirement already satisfied: python-dateutil>=2.1 in ./.local/lib/python3.
6/site-packages (from matplotlib>=3.0.0->mlxtend)
Requirement already satisfied: pillow>=6.2.0 in ./.local/lib/python3.6/site-p
ackages (from matplotlib>=3.0.0->mlxtend)
Requirement already satisfied: cycler>=0.10 in ./.local/lib/python3.6/site-pa
ckages (from matplotlib>=3.0.0->mlxtend)
Requirement already satisfied: certifi>=2020.06.20 in ./.local/lib/python3.6/
site-packages (from matplotlib>=3.0.0->mlxtend)
Requirement already satisfied: kiwisolver>=1.0.1 in ./.local/lib/python3.6/si
te-packages (from matplotlib>=3.0.0->mlxtend)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.3 in /u
sr/local/anaconda5/lib/python3.6/site-packages (from matplotlib>=3.0.0->mlxte
nd)
Requirement already satisfied: pytz>=2017.2 in ./.local/lib/python3.6/site-pa
ckages (from pandas>=0.24.2->mlxtend)
Requirement already satisfied: threadpoolctl>=2.0.0 in ./.local/lib/python3.
6/site-packages (from scikit-learn>=0.20.3->mlxtend)
Requirement already satisfied: six>=1.5 in /usr/local/anaconda5/lib/python3.
6/site-packages (from python-dateutil>=2.1->matplotlib>=3.0.0->mlxtend)
You are using pip version 9.0.1, however version 20.2.4 is available.
You should consider upgrading via the 'pip install --upgrade pip' command.
```

In [9]:
```
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 [3]:
```
# Generating Data1

np.random.seed(100)

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

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


# Generating Data2

np.random.seed(100)

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

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


# Generating Data3

np.random.seed(100)

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

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


# Generating Data4

np.random.seed(100)

Data4_X = np.random.uniform(0,12,[500,2])
Data4_Y = np.ones([500]).astype(int)
Data4_Y[np.multiply(Data4_X[:,0],Data4_X[:,0]) + np.multiply(Data4_X[:,1],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 [4]:
```
Depth = [2,4,6,8,10,50]
Scores = {}
for x in Depth:
    dt = DecisionTreeClassifier(max_depth=x)
    Scores[x] = cross_val_score(dt, Data3_X, Data3_Y, cv=10, scoring='accurac
y').mean()
Scores
```

Out[4]:
```
{2: 0.875,
 4: 0.97,
 6: 0.9674999999999999,
 8: 0.9499999999999998,
 10: 0.9424999999999999,
 50: 0.9450000000000001}
```

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

Answer: The value with the lowest accuracy was where the depth is equal to 2. This 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?

Answer: The mean accuracy for depth=50 was 94.68%. The highest accuracy was at depth=4 at about 97.18%. This phenominal is called overfitting.

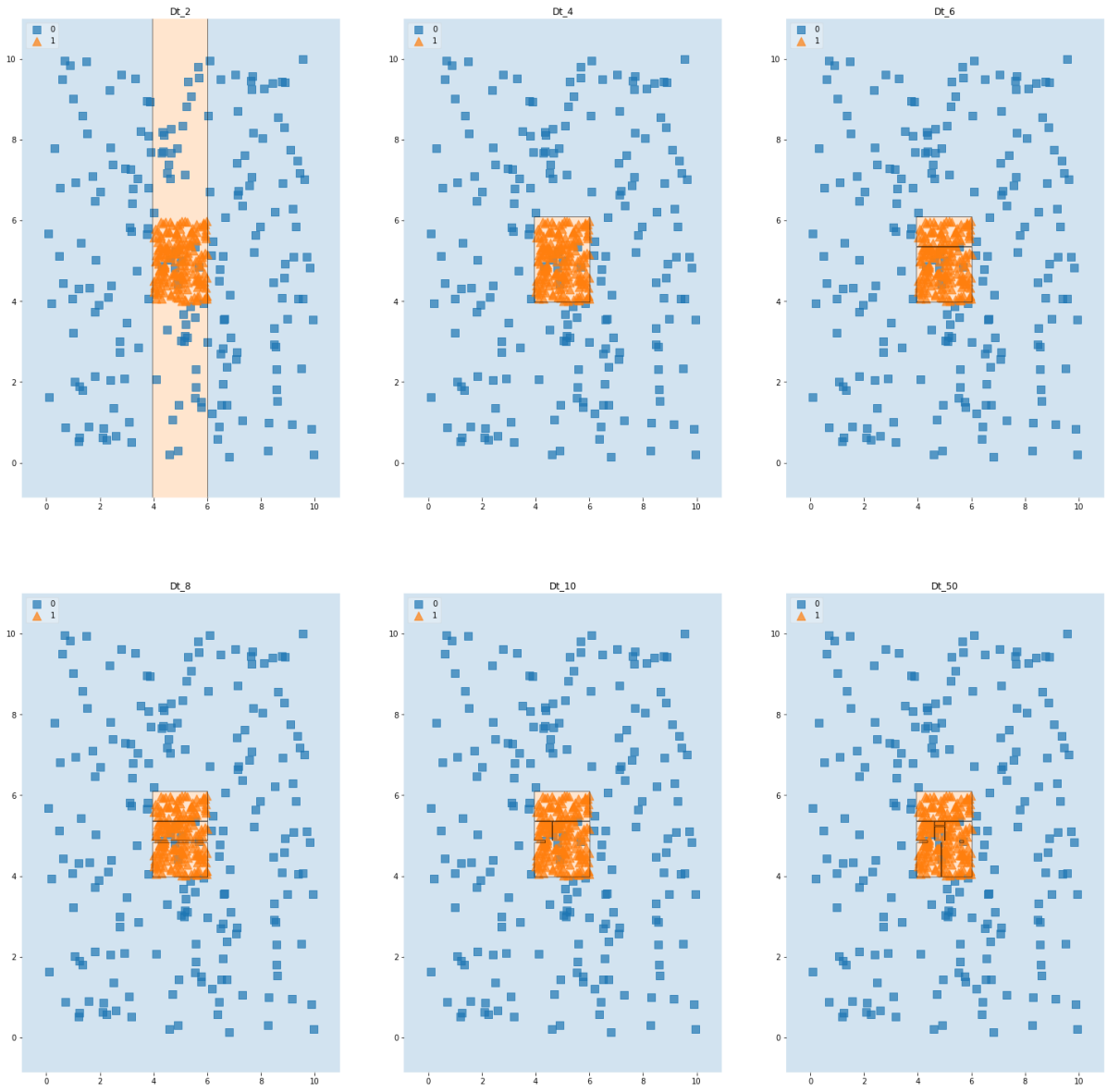
Question 1d: Plot decision regions for the above decision tree models

In [5]:
```
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}
Dt_2 = DecisionTreeClassifier(max_depth=2)
Dt_4 = DecisionTreeClassifier(max_depth=4)
Dt_6 = DecisionTreeClassifier(max_depth=6)
Dt_8 = DecisionTreeClassifier(max_depth=8)
Dt_10 = DecisionTreeClassifier(max_depth=10)
Dt_50 = DecisionTreeClassifier(max_depth=50)

clf_list = [Dt_2, Dt_4, Dt_6, Dt_8, Dt_10, Dt_50]
labels = ['Dt_2', 'Dt_4', 'Dt_6', 'Dt_8', 'Dt_10', 'Dt_50']

fig = plt.figure(figsize=(25,25))
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: The depth that is best suited for our data is 4. This is because it allows us to create a sufficient separation of our classes without creating too specific of partitions. Going much deaper sacrifices the generalizeability of our 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 [6]:
```
Depth = [1,5,10,50]
Scores = {}
for x in Depth:
    knn = KNeighborsClassifier(n_neighbors=x)
    Scores[x] = cross_val_score(knn, Data3_X, Data3_Y, cv=10, scoring='accurac
y').mean()
Scores
```

Out[6]:
```
{1: 0.9425000000000001, 5: 0.9475, 10: 0.9375, 50: 0.8775000000000001}
```

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

Answer: I observed the lowest accuracy when the neighbors was equal to 50.

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

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

clf_list = [knn1, knn5, knn10, knn50]
labels = ['knn1', 'knn5', 'knn10', 'knn50']

fig = plt.figure(figsize=(10,8))
count = 0;
for clf, label in zip(clf_list, labels):
    count = count + 1;
    clf.fit(Data3_X, Data3_Y)
    ax = plt.subplot(2,2,count)
    fig = plot_decision_regions(X=Data3_X, y=Data3_Y, clf=clf, legend=2,
                        scatter_kwargs=scatter_kwargs,
                        contourf_kwargs=contourf_kwargs,
                        scatter_highlight_kwargs=scatter_highlight_kwargs)
    plt.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 the number of neighbors is increasing, the boundary is expanding beyond the optimal classification boundary and will more commonly misclassify new data points.

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 [8]:
```
nb = GaussianNB()
DataX = [Data1_X,Data2_X,Data3_X,Data4_X]
DataY = [Data1_Y,Data2_Y,Data3_Y,Data4_Y]
for x, y in zip(DataX, DataY):
    nb_scores = cross_val_score(nb, x, y, cv=10, scoring='accuracy')
    print([nb_scores.mean(), nb_scores.std()])
```

```
[0.9675, 0.03172144385112379]
[0.049999999999999996, 0.026809513236909017]
[0.96, 0.027838821814150098]
[0.9640000000000001, 0.02154065922853801]
```

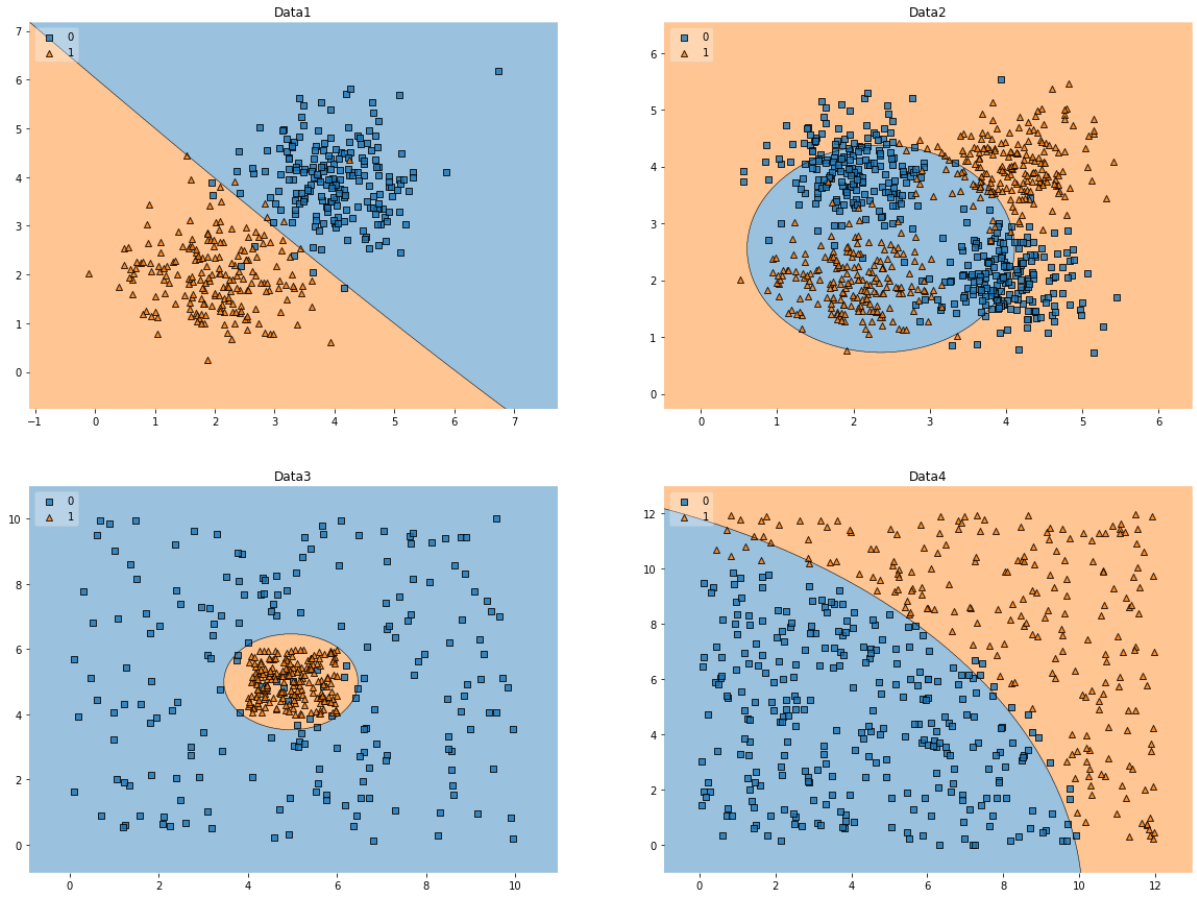
Question 3b: State your observations on the datasets the NB algorithm performed poorly.

Answer: The algorithm performed poorly only on the second data set.

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

In [9]:
```
Labels = ['Data1', 'Data2', 'Data3', 'Data4']

fig = plt.figure(figsize=(20,15))
count = 0;
for x, y , label in zip(DataX, DataY, 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()
```



In [ ]:
```

```

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

Answer: For the first data set we have a line cutting the region diagnonally. It is drawn this way to separate the two classes and it appears as a line because the probability of it being one class or the other are equal along that line. The variances are about the same for each class and the means for each class are relfected accross the line which results in a diagonal line across the plane. For the second data set we have an oval off center of the data points towards the origin. We have an oval because the variance is likely greater along the x direction which allows for a greater SD across this direction when we assume normality. This means our prediction will allow for more flexibility in the direction when computing the probability of class in this direction. The oval struggles to properly draw a diagonal oval because of the independence assumption and settles for a boundary that has about 50% accuracy. The third dataset has an oval for a decision boundary

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

Answer: The naive bayes performed poorly on dataset 2, because there is some covariance within the classes which cannot be capture by naive bayes. The best regions would be diagonal ovals, but it cannot capture this because of the independence assumption.

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: From best to worst it will be 1, 4, 3, and then 2.

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 [10]:
```
svm_linear = SVC(C=0.5, kernel='linear')
for x, y in zip(DataX, DataY):
    svm_linear_scores = cross_val_score(svm_linear, x, y, cv=10, scoring='accu
racy')
    print([svm_linear_scores.mean(), svm_linear_scores.std()])
```

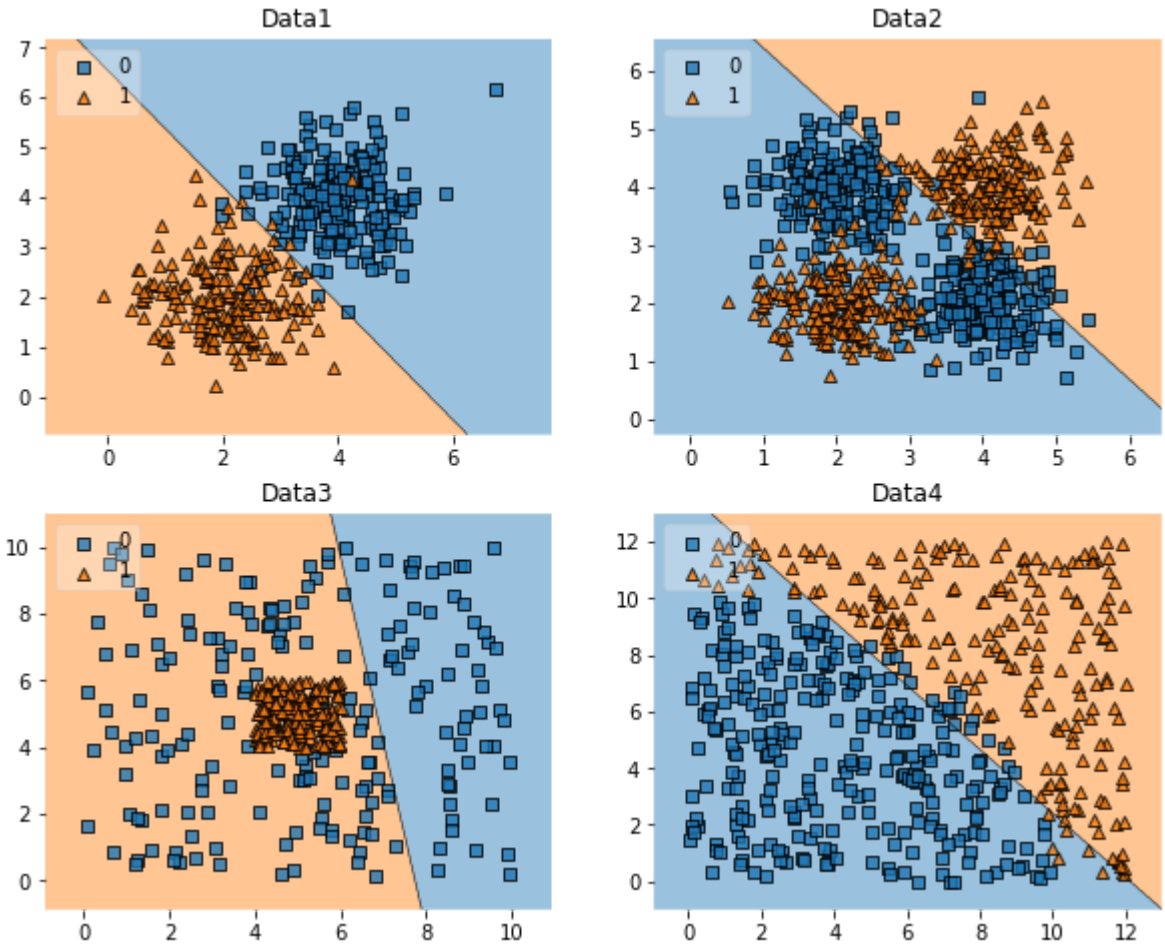
```
[0.9674999999999999, 0.02968585521759479]
[0.14125000000000001, 0.06124999999999999]
[0.6425000000000001, 0.0275]
[0.9219999999999999, 0.02749545416973502]
```

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

Answer: In decreasing order: 1, 4, 3, 2.

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

In [11]:
```
fig = plt.figure(figsize=(10,8))
count = 0
for x, y , label in zip(DataX, DataY, 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: We have high accuracy for 1 and 4 because they are nearly linearly separable. 2 and 3 cannot be separated well by just a line so you have low prediction accuracy.

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 [12]:
```
Degrees = [1,2,3]
for x in Degrees:
    svm_poly = SVC(C=0.5, kernel='poly',degree=x, gamma = 'auto')
    svm_poly_scores = cross_val_score(svm_poly, Data2_X, Data2_Y, cv=10, scori
ng='accuracy')
    print([svm_poly_scores.mean(), svm_poly_scores.std()])
```

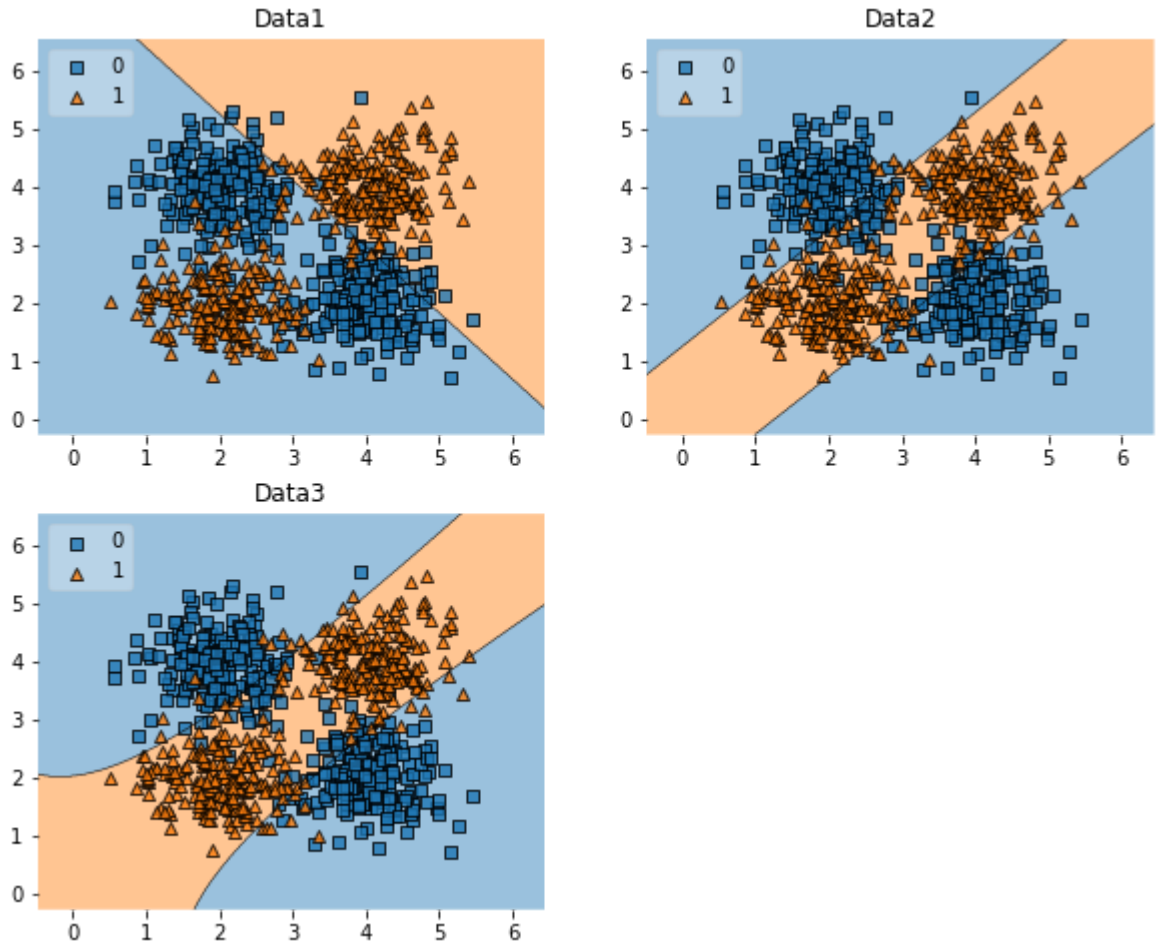
```
[0.13375, 0.05215661511256266]
[0.865, 0.030516389039334235]
[0.8762500000000001, 0.026487025125521375]
```

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

Answer: In decreasing order would be degree 3, 2, then 1.

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

In [13]:
```
fig = plt.figure(figsize=(10,8))
count = 0
for d, label in zip(Degrees, Labels):
    count = count + 1;
    svm_poly = SVC(C=0.5, kernel='poly',degree=d, gamma = 'auto')
    svm_poly.fit(Data2_X,Data2_Y)
    ax = plt.subplot(2,2,count)
    fig = plot_decision_regions(X=Data2_X, y=Data2_Y, clf=svm_poly, 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: The reason we are able to better separate our decision regions is because we are now using a polynomial classifier which allows us to separate the classes with multiple lines and curved lines.

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 [14]:
```
gamma = [.01, .1,1]
for x in gamma:
    svm_rbf = SVC(C=0.5, kernel='rbf', gamma = x)
    svm_rbf_scores = cross_val_score(svm_rbf, Data2_X, Data2_Y, cv=10, scoring
='accuracy')
    print([svm_rbf_scores.mean(), svm_rbf_scores.std()])
```

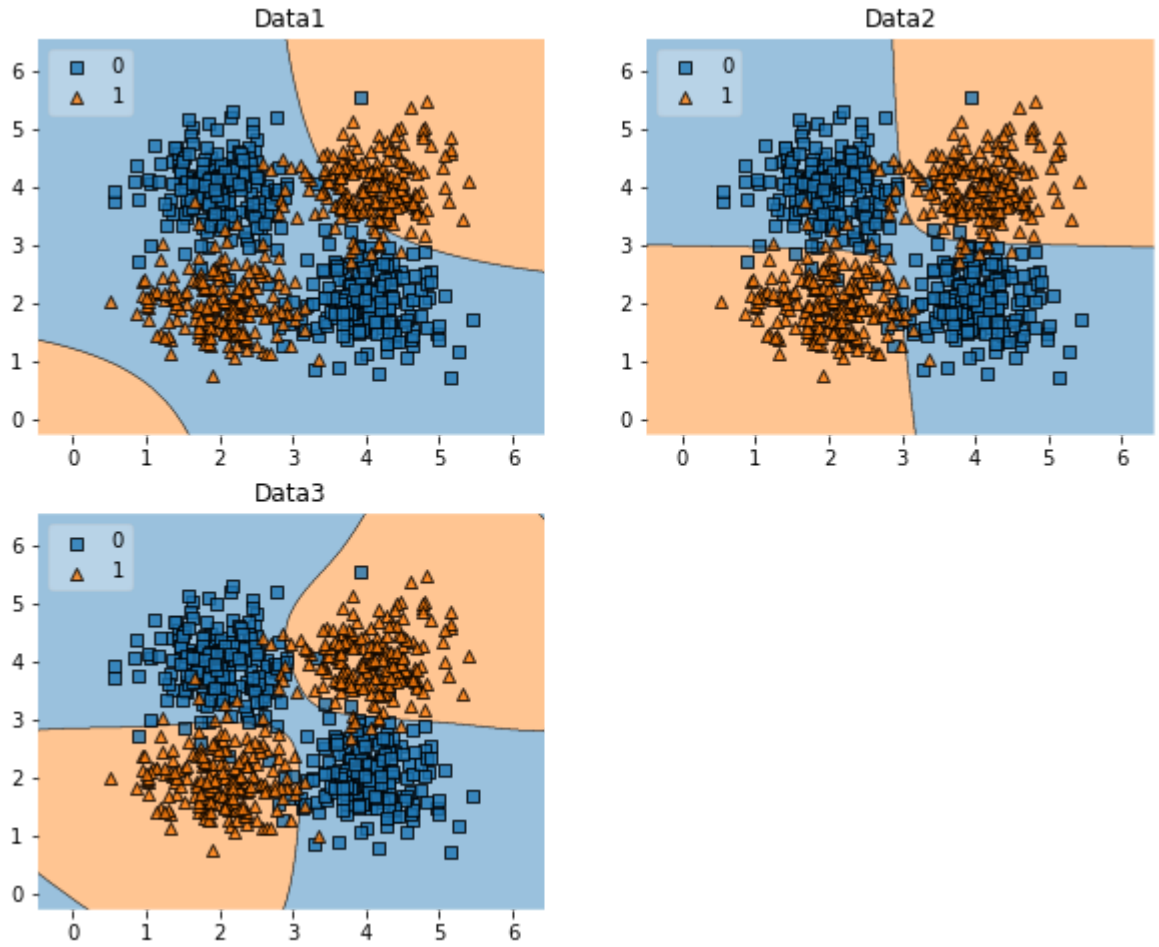
```
[0.30124999999999996, 0.08704345179276841]
[0.93625, 0.02928843628464996]
[0.9399999999999998, 0.029474565306379]
```

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

Answer: In decreasing order of accuracy, it goes degree 3, 2, then 1.

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

In [15]:
```
fig = plt.figure(figsize=(10,8))
count = 0
for d, label in zip(gamma, Labels):
    count = count + 1;
    svm_rbf = SVC(C=0.5, kernel='rbf', gamma = d)
    svm_rbf.fit(Data2_X,Data2_Y)
    ax = plt.subplot(2,2,count)
    fig = plot_decision_regions(X=Data2_X, y=Data2_Y, clf=svm_rbf, 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: Our classifier becomes more accurate as we raise our gamma value. A higher gamma value implies a lower SD in the kernel which reduces the similarity measure between points. This causes our classifier to create tighter and more precise decision boundaries.

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

Answer:

The radial based classifier is better suited, because the shape of its boundary is better suited for this kind of classification. It allows us to create these reflected hyperbolic boundaries that are not possible or very difficult with the polynomial kernel.

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 [20]:
```
cv = [3,4,5,6]
for x in cv:
    svm_rbf = SVC(C=0.5, kernel='rbf', gamma = 0.1)
    svm_rbf_scores = cross_val_score(svm_rbf, Data2_X, Data2_Y, cv=x, scoring=
'accuracy')
    print([svm_rbf_scores.mean(), svm_rbf_scores.std()])
```

```
[0.903748134380896, 0.026014752189391494]
[0.91625, 0.0163458710382775]
[0.9275, 0.02186606960566988]
[0.9325178618187259, 0.021970907802390045]
[0.94017094017094, 0.16184116711161767]
```

Answer: The mean accuracy increases with the number of folds there are in the dataset. The reason I believe this works is because there is more data in the training set when making the prediction on the test data in each iteration. Having more training data can often lead to better predictions.

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 [16]:
```
Dt = DecisionTreeClassifier(max_depth=4)
nb = GaussianNB()
knn = KNeighborsClassifier(n_neighbors=3)
svm_linear = SVC(C=0.5, kernel='linear')
svm_poly = SVC(C=0.5, kernel='poly',degree=3, gamma = 'auto')
svm_rbf = SVC(C=0.5, kernel='rbf', gamma = 1)

Classifiers = [Dt, nb, knn, svm_linear, svm_poly, svm_rbf]
Labels = ['Decision Tree', 'Naive Bayes', 'knn', 'SVM Linear', 'SVM Poly', 'SV
M RBF']
Accuracy = {}
Precision = {}

for c, label in zip(Classifiers, Labels):
    Accuracy[label] = cross_val_score(c, Data3_X, Data3_Y, cv=30, scoring='acc
uracy').mean()
    Precision[label] = cross_val_score(c, Data3_X, Data3_Y, cv=30, scoring='pr
ecision').mean()
print(Accuracy)
print(Precision)
```

```
{'Decision Tree': 0.96978021978022, 'Naive Bayes': 0.9597069597069599, 'knn':
0.9472527472527473, 'SVM Linear': 0.6406593406593408, 'SVM Poly': 0.855311355
3113554, 'SVM RBF': 0.9547619047619048}
{'Decision Tree': 0.9541666666666668, 'Naive Bayes': 0.9294973544973547, 'kn
n': 0.9138095238095238, 'SVM Linear': 0.5875420875420876, 'SVM Poly': 0.79443
96344396345, 'SVM RBF': 0.9227380952380952}
```

Answer: Accuracy Ranked best to worst: Decision Tree, Naive Bayes, SVM rbf, KNN, SVM poly, SVM Linear. Precision Ranked best to worst: Decision Tree, Naive Bayes, SVM rbf, KNN, SVM poly, SVM Linear. The best classifiers ranked according to precision and accuracy were the same.

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 [36]:
```
svm_linear = SVC(C=0.5, kernel='linear')
n_est_list = [1,2,3,4,5,10,20]
for n_est in n_est_list:
    bagging = BaggingClassifier(base_estimator=svm_linear, n_estimators=n_est)
    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))
```

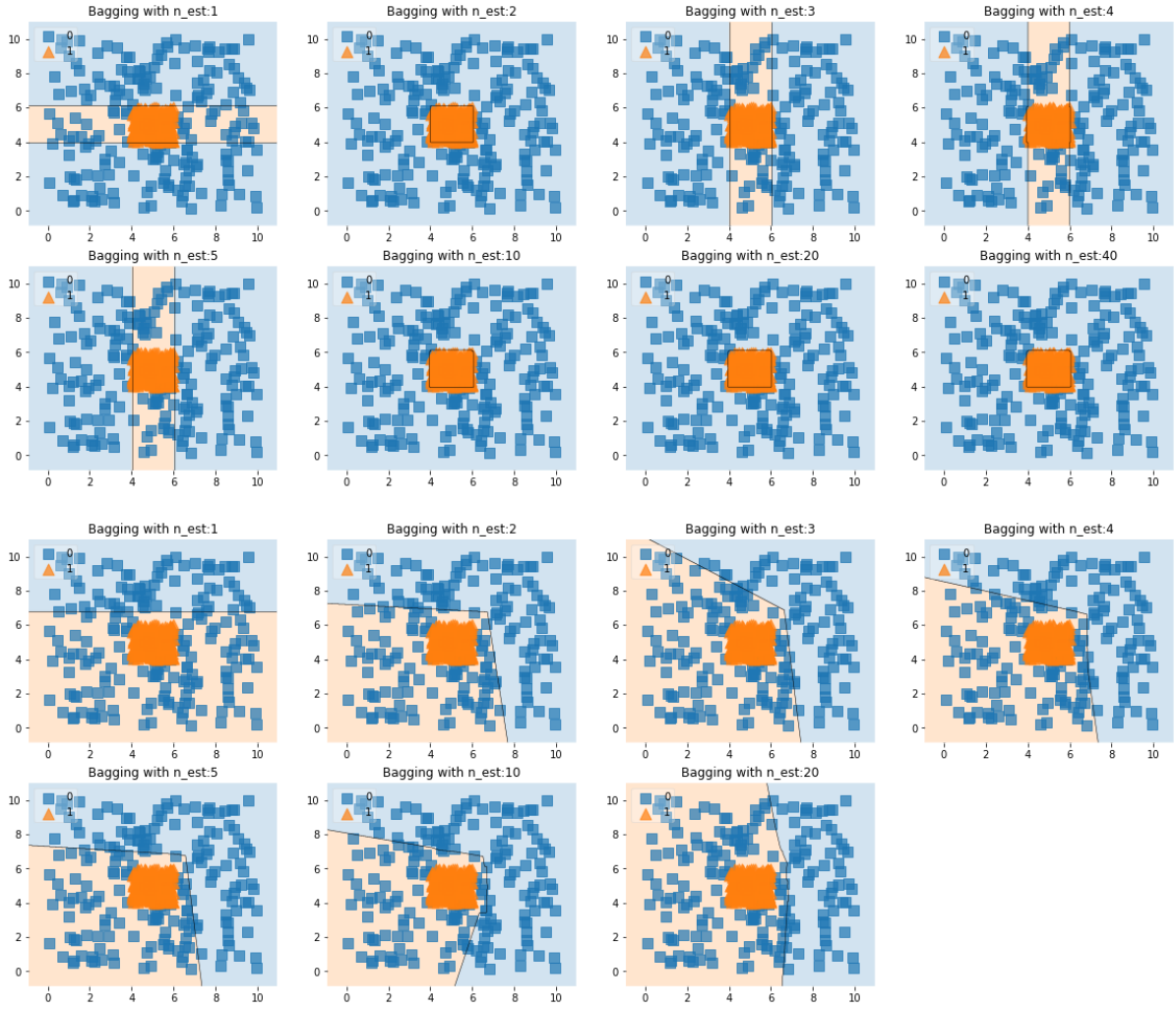
```
Bagging Accuracy: 0.88 (+/- 0.02) #estimators: 1
Bagging Accuracy: 0.93 (+/- 0.05) #estimators: 2
Bagging Accuracy: 0.87 (+/- 0.02) #estimators: 3
Bagging Accuracy: 0.91 (+/- 0.05) #estimators: 4
Bagging Accuracy: 0.93 (+/- 0.06) #estimators: 5
Bagging Accuracy: 0.96 (+/- 0.05) #estimators: 10
Bagging Accuracy: 0.89 (+/- 0.04) #estimators: 20
```

Answer: After running the experiment multiple times, it appears that the mean accuracy and standard deviation both increase slightly with the first few additional bags. After 5 - 10 bags, the accuracy seems to continue slowly increasing, but the standard deviation starts to fall below where is started. It makes clear sense to me why the accuracy will slowly increase with multiple bags. All of our data points are getting more equal representation in the bags, because each individual bag contains on average only 62.7% of the data points. So when many bags vote, all points are represented more equally which leads to better accuracy in predicting all points. As for the concave nature of the standard deviation values, the standard deviation decreasing over the long run is clear, because all points will converge toward equal relative representation amongst the many bags. As for why the standard deviation tend to increase from 1 to 5, I would explain this by saying that the convergence toward equal representation of points with more bags is outweighed by the increasing complexity and possibilities for representation of points with increasing bags when you only have a few. There more potential for the data point representation to be relatively skewed with just a few bags than many, because with many bags, the representation converges due to the law of large numbers. With one bag, either a point is represented or it is not. With 2, there will still be some points with no representaion and some that are represented twice. From 3-10, there are still likely to be some points with little representation and some with a lot of representation relative to each other. This increasingly entropic system is what causes a larger standard deviation, but as you continue adding bags the relative representation of points balances.

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

In [37]:
```
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()
```



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: With many estimators we seem to do slightly better. The decision boundary chosen by the linear SVM using bagging is not very effective because a linear classifier is not the best classifier for this dataset. One thing to note, however, is that when bagging the linear SVM multiple times you can start to get more irregular boundaries which can help to capture the non-linear boundary with more bags.

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 [38]:
```
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))
```

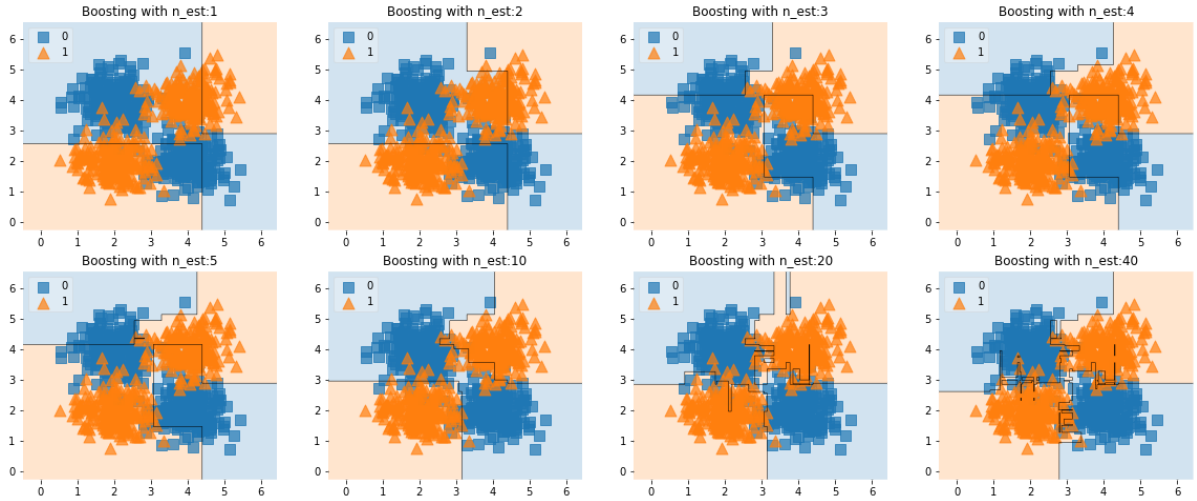
```
Boosting Accuracy: 0.88 (+/- 0.03) #estimators: 1
Boosting Accuracy: 0.88 (+/- 0.03) #estimators: 2
Boosting Accuracy: 0.90 (+/- 0.04) #estimators: 3
Boosting Accuracy: 0.90 (+/- 0.04) #estimators: 4
Boosting Accuracy: 0.92 (+/- 0.03) #estimators: 5
Boosting Accuracy: 0.92 (+/- 0.04) #estimators: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

Answer: The standard deviation remained constant with more bags while the accuracy slowly increased.

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

In [28]:
```
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: The noticeable difference from regular bagging is that the standard deviation remained constant. This is explainable due to the fact the the boosting procedure is selecting more points whose under-representation is leading to misclassification. This gives the subsequent bags a direction for choosing points which leads to a constant standard deviation because there is a correction mechanism for under-represented points. It allows the relative representation of the points to balance more quickly (or in a more effective way due to weighting) than it would by the law of large numbers for the simple bagging procedure. This correction mechanism leads to more consistent guessing as you increase the bags from 1.

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 [3]:
```
from sklearn import datasets
cancer = datasets.load_breast_cancer()
```

The features are:

In [4]:
```
cancer.feature_names
```

Out[4]:
```
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 [5]:
```
cancer.target_names
```

Out[5]:
```
array(['malignant', 'benign'], dtype='<U9')
```

Create dataset for classification

In [6]:
```
X = cancer.data
Y = cancer.target
```

Number of samples are:

In [7]:
```
X.shape
```

Out[7]:
```
(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 [13]:
```
#Linear SVM
C = [.001,.01,.1, 1]
for c in C:
    svm_linear = SVC(C=c, kernel='linear')
    scores = cross_val_score(svm_linear, X, Y, cv=10, scoring='accuracy')
    print("Linear SVM Accuracy: %.2f C: %.2f" % (scores.mean(), c))
```

```
Linear SVM Accuracy: 0.94 C: 0.00
Linear SVM Accuracy: 0.95 C: 0.01
Linear SVM Accuracy: 0.95 C: 0.10
Linear SVM Accuracy: 0.95 C: 0.50
Linear SVM Accuracy: 0.95 C: 1.00
```

In [56]:
```
#Polynomial varying C and degree
C = [0.001,0.01,0.1,1]
degree = [1,2]
gamma = [0.001, 0.01, 0.1, 1]
for c in C:
    for d in degree:
        svm_poly = SVC(C=c, kernel='poly',degree=d, gamma = 'auto')
        scores = cross_val_score(svm_poly, X, Y, cv=10, scoring='accuracy')
        print("Polynomial SVM Accuracy: %.2f C: = %.3f  Degree: %.3f" % (score
s.mean(), c, d))
```

```
Linear SVM Accuracy: 0.93 C: = 0.001  Degree: 1.000
Linear SVM Accuracy: 0.95 C: = 0.001  Degree: 2.000
Linear SVM Accuracy: 0.94 C: = 0.010  Degree: 1.000
Linear SVM Accuracy: 0.96 C: = 0.010  Degree: 2.000
Linear SVM Accuracy: 0.95 C: = 0.100  Degree: 1.000
Linear SVM Accuracy: 0.96 C: = 0.100  Degree: 2.000
Linear SVM Accuracy: 0.95 C: = 1.000  Degree: 1.000
Linear SVM Accuracy: 0.96 C: = 1.000  Degree: 2.000
```

In [12]:
```
#RBF varying C and gamma
#Polynomial varying C and degree
C = [0.001,0.01,0.1,1]
gamma = [0.001, 0.01, 0.1, 1, 10, 100]
for c in C:
    for g in gamma:
        svm_rbf = SVC(C=c, kernel='rbf', gamma = g)
        scores = cross_val_score(svm_rbf, X, Y, cv=10, scoring='accuracy')
        print("RBF SVM Accuracy: %.2f C: = %.3f  Gamma: %.5f" % (scores.mean
(), c, g))
```

```
RBF SVM Accuracy: 0.63 C: = 0.001  Gamma: 0.00100
RBF SVM Accuracy: 0.63 C: = 0.001  Gamma: 0.01000
RBF SVM Accuracy: 0.63 C: = 0.001  Gamma: 0.10000
RBF SVM Accuracy: 0.63 C: = 0.001  Gamma: 1.00000
RBF SVM Accuracy: 0.63 C: = 0.001  Gamma: 10.00000
RBF SVM Accuracy: 0.63 C: = 0.001  Gamma: 100.00000
RBF SVM Accuracy: 0.63 C: = 0.010  Gamma: 0.00100
RBF SVM Accuracy: 0.63 C: = 0.010  Gamma: 0.01000
RBF SVM Accuracy: 0.63 C: = 0.010  Gamma: 0.10000
RBF SVM Accuracy: 0.63 C: = 0.010  Gamma: 1.00000
RBF SVM Accuracy: 0.63 C: = 0.010  Gamma: 10.00000
RBF SVM Accuracy: 0.63 C: = 0.010  Gamma: 100.00000
RBF SVM Accuracy: 0.63 C: = 0.100  Gamma: 0.00100
RBF SVM Accuracy: 0.63 C: = 0.100  Gamma: 0.01000
RBF SVM Accuracy: 0.63 C: = 0.100  Gamma: 0.10000
RBF SVM Accuracy: 0.63 C: = 0.100  Gamma: 1.00000
RBF SVM Accuracy: 0.63 C: = 0.100  Gamma: 10.00000
RBF SVM Accuracy: 0.63 C: = 0.100  Gamma: 100.00000
RBF SVM Accuracy: 0.92 C: = 1.000  Gamma: 0.00100
RBF SVM Accuracy: 0.63 C: = 1.000  Gamma: 0.01000
RBF SVM Accuracy: 0.63 C: = 1.000  Gamma: 0.10000
RBF SVM Accuracy: 0.63 C: = 1.000  Gamma: 1.00000
RBF SVM Accuracy: 0.63 C: = 1.000  Gamma: 10.00000
RBF SVM Accuracy: 0.63 C: = 1.000  Gamma: 100.00000
```

Answer: The Polynomial SVM is the most accurate classifier of them. It has its highest accuracy at 96% compared to 95% for the linear classifier with the same C value. Having 2 polynomial terms tended to make the classifier more accurate. THe RBF classifier did not perform accurately for many of the values chosen except when C=1 and gamma=.001.

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

In [60]:
```
Depth = [2,3,4,5,6,7,8,9,10,20,30,40,50,60,70,80,90,100]
Scores = {}
for x in Depth:
    dt = DecisionTreeClassifier(max_depth=x)
    Scores[x] = cross_val_score(dt, X, Y, cv=10, scoring='accuracy').mean()
Scores
```

Out[60]:
```
{2: 0.9209899749373432,
 3: 0.9104636591478696,
 4: 0.9157894736842105,
 5: 0.9262531328320801,
 6: 0.9263157894736842,
 7: 0.9192669172932331,
 8: 0.9139097744360901,
 9: 0.9140037593984962,
 10: 0.9104323308270675,
 20: 0.92453007518797,
 30: 0.9157268170426065,
 40: 0.9156954887218044,
 50: 0.9174498746867167,
 60: 0.9227130325814535,
 70: 0.9174812030075186,
 80: 0.906829573934837,
 90: 0.9104010025062657,
 100: 0.9068609022556393}
```

Answer: The best classifier was at a max_depth of 6. The classifier was also close to its most accurate point at 5, 20, and 60.

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: It is not easy to know which approach to choose. No one classifier will perform best on all datasets according to the no free lunch theorem. I would try to fit the data to many different models and give preference to models which have a high sensitity. The reason I would do this is it is much more important to find all of the cases of malignant cancer than it is to misclassify.

In [ ]:
```
```