Hands-on Exercise CLASS Module

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

```
Requirement already satisfied: mlxtend in /users/PES0801/krishnateja
0612/.local/lib/python3.6/site-packages
Requirement already satisfied: joblib>=0.13.2 in /users/PES0801/kris
hnateja0612/.local/lib/python3.6/site-packages (from mlxtend)
Requirement already satisfied: scikit-learn>=0.20.3 in /users/PES080
1/krishnateja0612/.local/lib/python3.6/site-packages (from mlxtend)
Requirement already satisfied: matplotlib>=3.0.0 in /users/PES0801/k
rishnateja0612/.local/lib/python3.6/site-packages (from mlxtend)
Requirement already satisfied: setuptools in /usr/local/anaconda5/li
b/python3.6/site-packages (from mlxtend)
Requirement already satisfied: scipy>=1.2.1 in /users/PES0801/krishn
ateja0612/.local/lib/python3.6/site-packages (from mlxtend)
Requirement already satisfied: numpy>=1.16.2 in /users/PES0801/krish
nateja0612/.local/lib/python3.6/site-packages (from mlxtend)
Requirement already satisfied: pandas>=0.24.2 in /users/PES0801/kris
hnateja0612/.local/lib/python3.6/site-packages (from mlxtend)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.
0.1 in /usr/local/anaconda5/lib/python3.6/site-packages (from matplo
tlib>=3.0.0->mlxtend)
Requirement already satisfied: kiwisolver>=1.0.1 in /users/PES0801/k
rishnateja0612/.local/lib/python3.6/site-packages (from matplotlib>=
3.0.0->mlxtend)
Requirement already satisfied: cycler>=0.10 in /usr/local/anaconda5/
lib/python3.6/site-packages (from matplotlib>=3.0.0->mlxtend)
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/an
aconda5/lib/python3.6/site-packages (from matplotlib>=3.0.0->mlxtend
)
Requirement already satisfied: pytz>=2017.2 in /usr/local/anaconda5/
lib/python3.6/site-packages (from pandas>=0.24.2->mlxtend)
Requirement already satisfied: six in /usr/local/anaconda5/lib/pytho
n3.6/site-packages (from cycler>=0.10->matplotlib>=3.0.0->mlxtend)
You are using pip version 9.0.1, however version 19.3.1 is available
.
You should consider upgrading via the 'pip install --upgrade pip' co
mmand.
```

In [74]:
```
# Generating Data1
import numpy as np
np.random.seed(100)

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

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


# Generating Data2

np.random.seed(100)

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

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


# Generating Data3

np.random.seed(100)

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

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


# Generating Data4

np.random.seed(100)

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

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 [51]:
```
from sklearn.tree import DecisionTreeClassifier
dt = DecisionTreeClassifier(max_depth=2)
dt_scores = cross_val_score(dt, Data3_X, Data3_Y, cv=10, scoring='accu
racy')
[dt_scores.mean(), dt_scores.std()]
```

Out[51]:
```
[0.875, 0.03354101966249685]
```

In [116]:
```
from sklearn.tree import DecisionTreeClassifier
dt = DecisionTreeClassifier(max_depth=4)
dt_scores = cross_val_score(dt, Data3_X, Data3_Y, cv=10, scoring='accu
racy')
[dt_scores.mean(), dt_scores.std()]
```

Out[116]:
```
[0.9724999999999999, 0.02076655965729518]
```

In [19]:
```
from sklearn.tree import DecisionTreeClassifier
dt = DecisionTreeClassifier(max_depth=6)
dt_scores = cross_val_score(dt, Data3_X, Data3_Y, cv=10, scoring='accu
racy')
[dt_scores.mean(), dt_scores.std()]
```

Out[19]:
```
[0.95250000000000001, 0.03249999999999998]
```

In [20]:
```
from sklearn.tree import DecisionTreeClassifier
dt = DecisionTreeClassifier(max_depth=8)
dt_scores = cross_val_score(dt, Data3_X, Data3_Y, cv=10, scoring='accu
racy')
[dt_scores.mean(), dt_scores.std()]
```

Out[20]:
```
[0.95749999999999991, 0.031721443851123785]
```

In [21]:
```
from sklearn.tree import DecisionTreeClassifier
dt = DecisionTreeClassifier(max_depth=10)
dt_scores = cross_val_score(dt, Data3_X, Data3_Y, cv=10, scoring='accu
racy')
[dt_scores.mean(), dt_scores.std()]
```

Out[21]:
```
[0.95749999999999991, 0.031721443851123785]
```

In [107]:
```
from sklearn.tree import DecisionTreeClassifier
dt = DecisionTreeClassifier(max_depth=50)
dt_scores = cross_val_score(dt, Data3_X, Data3_Y, cv=10, scoring='accu
racy')
[dt_scores.mean(), dt_scores.std()]
```

Out[107]:
```
[0.9399999999999998, 0.04062019202317979]
```

In [38]:
```
dt_scores
```

Out[38]:
```
array([ 0.975,  0.95 ,  1.   ,  0.975,  0.925,  0.975,  0.925,  0.9
,
        0.95 ,  0.975])
```

In [117]:
```
0.9724999999999999-0.9399999999999998
```

Out[117]:
```
0.032500000000000084
```

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

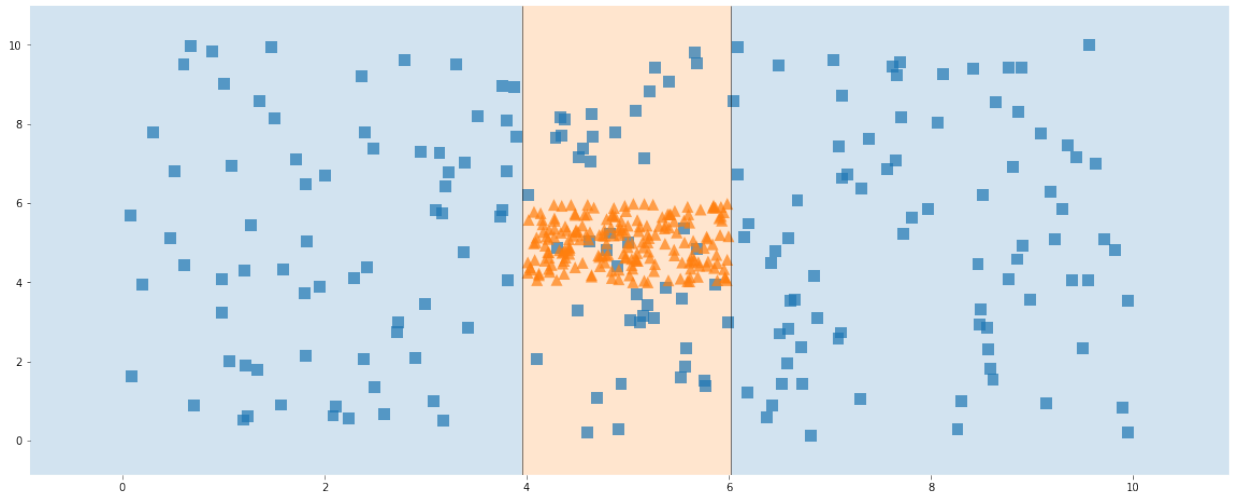
max_depth=2 has lowest accuracy when compared to all other given depths. Since very few axis parallel line is drawn within the depth of two,there will be high entropy still which is the reason for low accuracy rate. This phenomenon of not covering enough attributes on training data itself is called as 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?

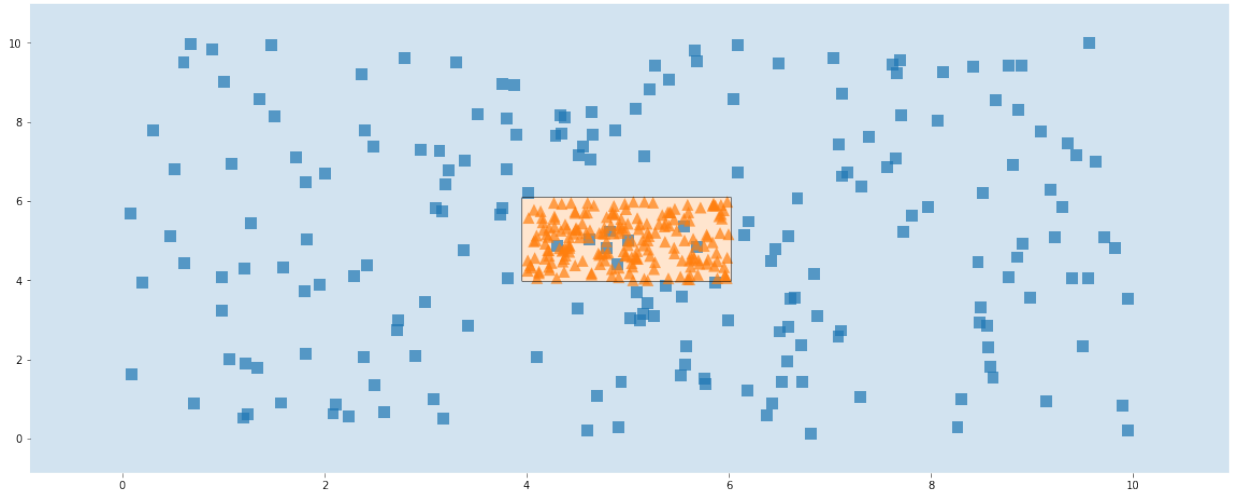
The accuracy for max_depth= 50 is 0.9399999999999998 while the highest accuracy is 0.032500000000000084(for depth of 4). Theur difference is 0.032500000000000084.This phenomenon is called underfitting of the data.

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

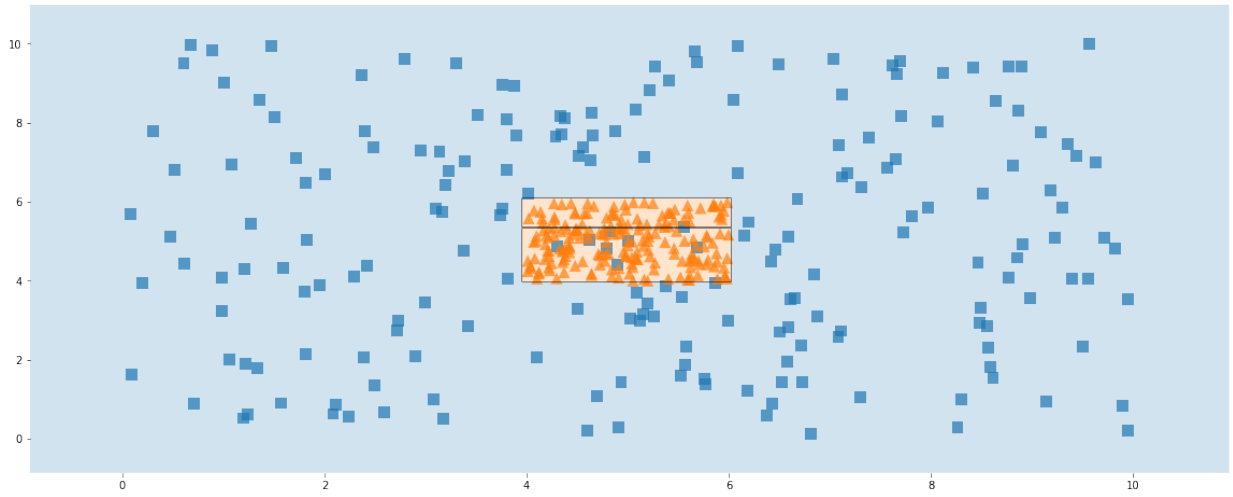
In [109]:
```
import matplotlib.pyplot as plt
from mlxtend.plotting import plot_decision_regions
dt = DecisionTreeClassifier(max_depth=2)
dt.fit(Data3_X, Data3_Y)
scatter_kwargs = {'s': 120, 'edgecolor':'none', 'alpha': 0.7}
contourf_kwargs = {'alpha': 0.2}
scatter_highlight_kwargs = {'s': 120, 'label': 'Test data', 'alpha': 0
.7}
fig = plt.figure(figsize=(20, 8))
fig = plot_decision_regions(X=Data3_X, y=Data3_Y, clf=dt, legend=0,sca
tter_kwargs=scatter_kwargs,
                      contourf_kwargs=contourf_kwargs,
                      scatter_highlight_kwargs=scatter_highlight_kwarg
s)
plt.show()
```



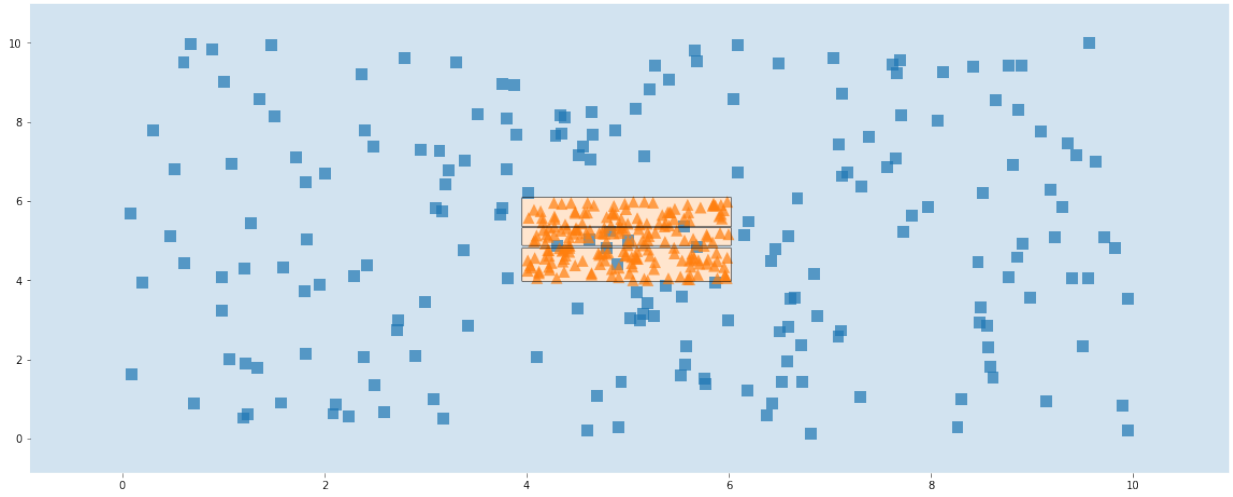
In [110]:
```
import matplotlib.pyplot as plt
from mlxtend.plotting import plot_decision_regions
dt = DecisionTreeClassifier(max_depth=4)
dt.fit(Data3_X, Data3_Y)
scatter_kwargs = {'s': 120, 'edgecolor':'none', 'alpha': 0.7}
contourf_kwargs = {'alpha': 0.2}
scatter_highlight_kwargs = {'s': 120, 'label': 'Test data', 'alpha': 0
.7}
fig = plt.figure(figsize=(20, 8))
fig = plot_decision_regions(X=Data3_X, y=Data3_Y, clf=dt, legend=0,sca
tter_kwargs=scatter_kwargs,
                      contourf_kwargs=contourf_kwargs,
                      scatter_highlight_kwargs=scatter_highlight_kwarg
s)
plt.show()
```



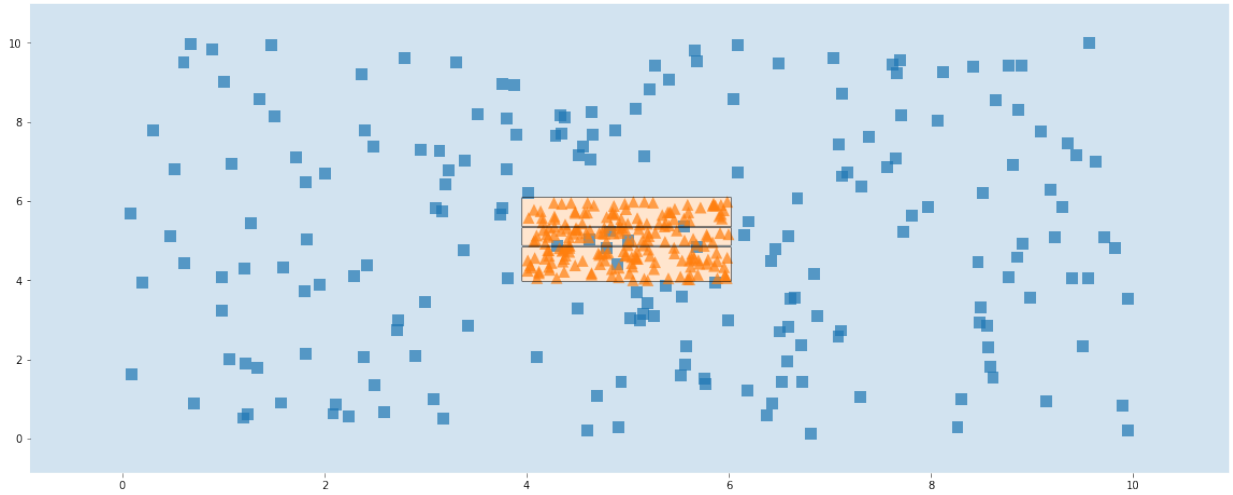
In [111]:
```
import matplotlib.pyplot as plt
from mlxtend.plotting import plot_decision_regions
dt = DecisionTreeClassifier(max_depth=6)
#svm_poly_5.fit(Data1_X, Data1_Y)
dt.fit(X=Data3_X, y=Data3_Y)
scatter_kwargs = {'s': 120, 'edgecolor':'none', 'alpha': 0.7}
contourf_kwargs = {'alpha': 0.2}
scatter_highlight_kwargs = {'s': 120, 'label': 'Test data', 'alpha': 0
.7}
fig = plt.figure(figsize=(20, 8))
fig = plot_decision_regions(X=Data3_X, y=Data3_Y, clf=dt, legend=0,sca
tter_kwargs=scatter_kwargs,
                      contourf_kwargs=contourf_kwargs,
                      scatter_highlight_kwargs=scatter_highlight_kwarg
s)
plt.show()
```



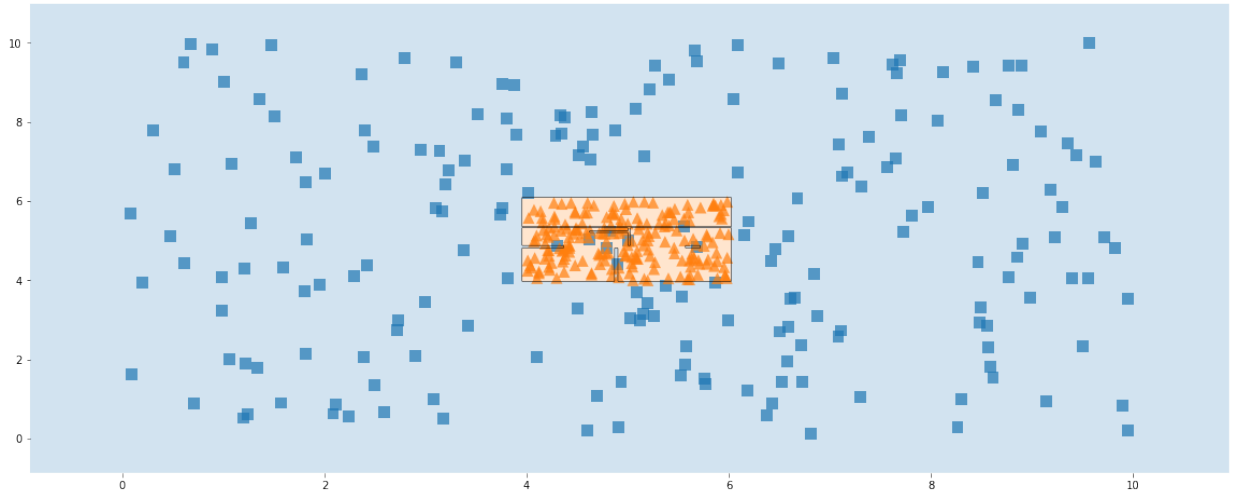
In [112]:
```
import matplotlib.pyplot as plt
from mlxtend.plotting import plot_decision_regions
dt = DecisionTreeClassifier(max_depth=8)
dt.fit(X=Data3_X, y=Data3_Y)
scatter_kwargs = {'s': 120, 'edgecolor':'none', 'alpha': 0.7}
contourf_kwargs = {'alpha': 0.2}
scatter_highlight_kwargs = {'s': 120, 'label': 'Test data', 'alpha': 0
.7}
fig = plt.figure(figsize=(20, 8))
fig = plot_decision_regions(X=Data3_X, y=Data3_Y, clf=dt, legend=0,sca
tter_kwargs=scatter_kwargs,
                      contourf_kwargs=contourf_kwargs,
                      scatter_highlight_kwargs=scatter_highlight_kwarg
s)
plt.show()
```



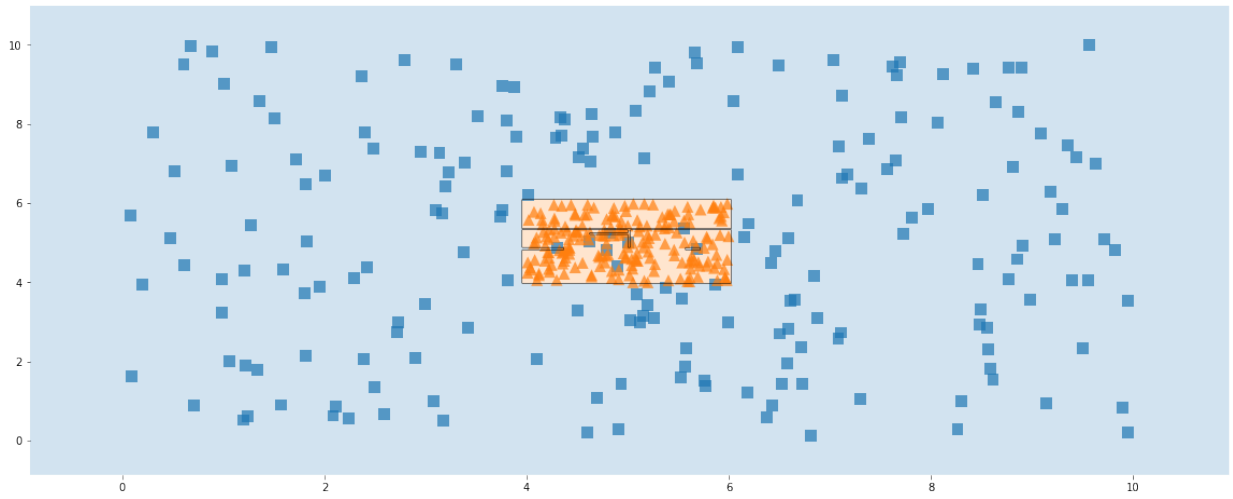
In [113]:
```
import matplotlib.pyplot as plt
from mlxtend.plotting import plot_decision_regions
dt = DecisionTreeClassifier(max_depth=10)
dt.fit(X=Data3_X, y=Data3_Y)
scatter_kwargs = {'s': 120, 'edgecolor':'none', 'alpha': 0.7}
contourf_kwargs = {'alpha': 0.2}
scatter_highlight_kwargs = {'s': 120, 'label': 'Test data', 'alpha': 0
.7}
fig = plt.figure(figsize=(20, 8))
fig = plot_decision_regions(X=Data3_X, y=Data3_Y, clf=dt, legend=0,sca
tter_kwargs=scatter_kwargs,
                      contourf_kwargs=contourf_kwargs,
                      scatter_highlight_kwargs=scatter_highlight_kwarg
s)
plt.show()
```



In [114]:
```
import matplotlib.pyplot as plt
from mlxtend.plotting import plot_decision_regions
dt = DecisionTreeClassifier(max_depth=50)
dt.fit(X=Data3_X, y=Data3_Y)
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}
fig = plt.figure(figsize=(20, 8))
fig = plot_decision_regions(X=Data3_X, y=Data3_Y, clf=dt, legend=0,sca
tter_kwargs=scatter_kwargs,
                      contourf_kwargs=contourf_kwargs,
                      scatter_highlight_kwargs=scatter_highlight_kwarg
s)
plt.show()
```



In [115]:
```
import matplotlib.pyplot as plt
from mlxtend.plotting import plot_decision_regions
dt = DecisionTreeClassifier(max_depth=100)
dt.fit(X=Data3_X, y=Data3_Y)
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}
fig = plt.figure(figsize=(20, 8))
fig = plot_decision_regions(X=Data3_X, y=Data3_Y, clf=dt, legend=0,sca
tter_kwargs=scatter_kwargs,
                      contourf_kwargs=contourf_kwargs,
                      scatter_highlight_kwargs=scatter_highlight_kwarg
s)
plt.show()
```



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

All the depths from 4 classify the data but there's chance of overfitting(as we see in diagrams) when we take depth higher than 4. Therefore, max_depth=4 is safe classifier with good accuracy. The plots ascertain the accuracy values we got above.

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 [10]:
```
knn = KNeighborsClassifier(n_neighbors=1)
knn_scores = cross_val_score(knn, Data2_X, Data2_Y, cv=10, scoring='ac
curacy')
[knn_scores.mean(), knn_scores.std()]
```

Out[10]: [0.9125, 0.02091650066335192]

In [42]:
```
knn = KNeighborsClassifier(n_neighbors=5)
knn_scores = cross_val_score(knn, Data2_X, Data2_Y, cv=10, scoring='ac
curacy')
[knn_scores.mean(), knn_scores.std()]
```

Out[42]: [0.93499999999999994, 0.032500000000000001]

In [43]:
```
knn = KNeighborsClassifier(n_neighbors=10)
knn_scores = cross_val_score(knn, Data2_X, Data2_Y, cv=10, scoring='ac
curacy')
[knn_scores.mean(), knn_scores.std()]
```

Out[43]: [0.94000000000000006, 0.034369317712168793]

In [44]:
```
knn = KNeighborsClassifier(n_neighbors=50)
knn_scores = cross_val_score(knn, Data2_X, Data2_Y, cv=10, scoring='ac
curacy')
[knn_scores.mean(), knn_scores.std()]
```

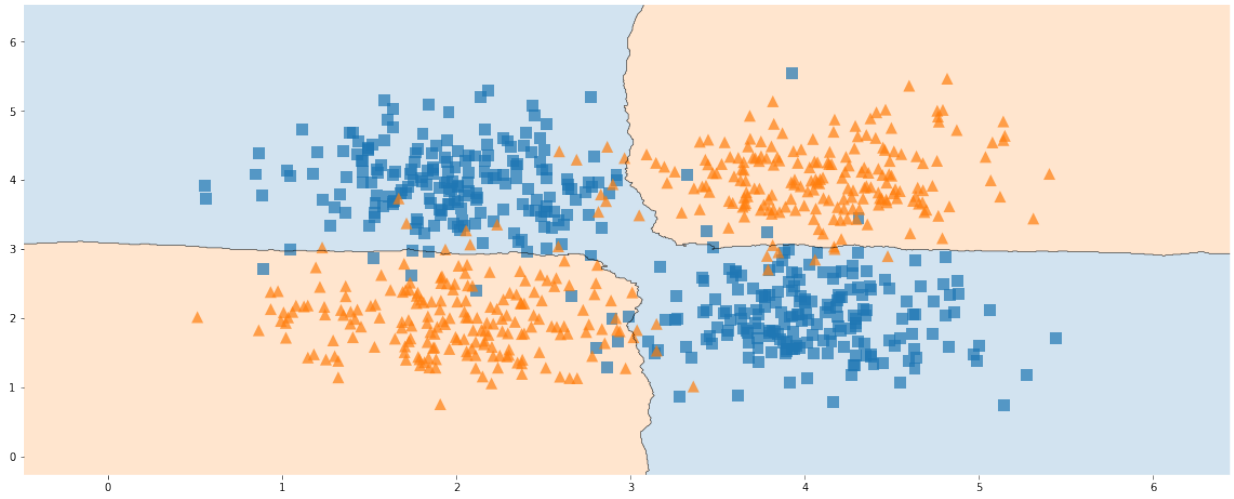
Out[44]: [0.94124999999999992, 0.030644126680328173]

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

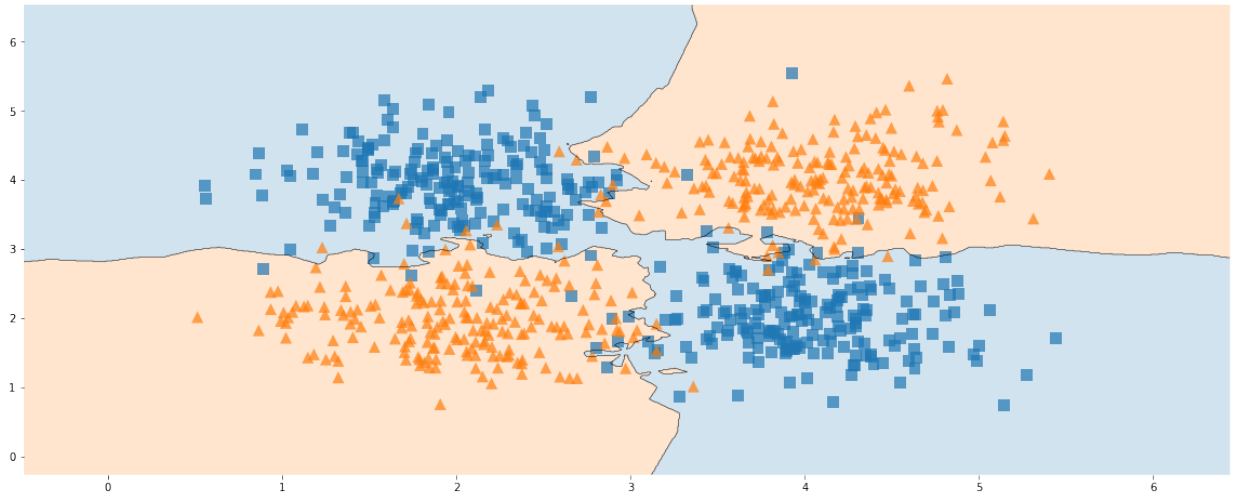
Lowest accuracy is observed when k=1(0.9125). This phenomenon is called overfitting of the data(ie: our classifier becomes sensitive to noise and misclassifies the test data which is in the case of high variance)

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

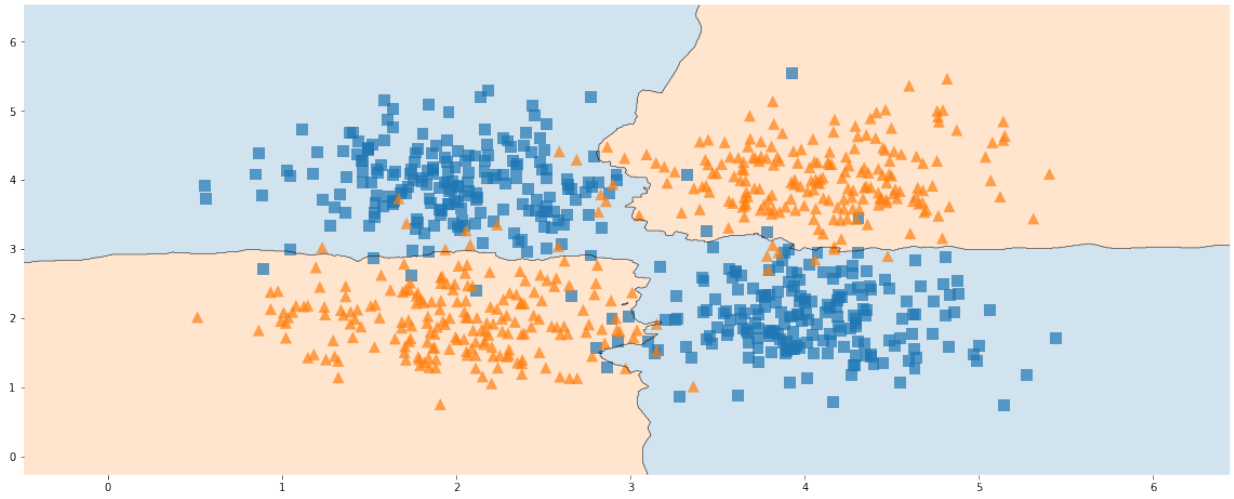
In [28]:
```
import matplotlib.pyplot as plt
from mlxtend.plotting import plot_decision_regions
knn = KNeighborsClassifier(n_neighbors=1)
#svm_poly_5.fit(Data1_X, Data1_Y)
knn.fit(Data2_X, Data2_Y)
scatter_kwargs = {'s': 120, 'edgecolor':'none', 'alpha': 0.7}
contourf_kwargs = {'alpha': 0.2}
scatter_highlight_kwargs = {'s': 120, 'label': 'Test data', 'alpha': 0
.7}
fig = plt.figure(figsize=(20, 8))
fig = plot_decision_regions(X=Data2_X, y=Data2_Y, clf=knn, legend=0,sc
atter_kwargs=scatter_kwargs,
                      contourf_kwargs=contourf_kwargs,
                      scatter_highlight_kwargs=scatter_highlight_kwarg
s)
plt.show()
```



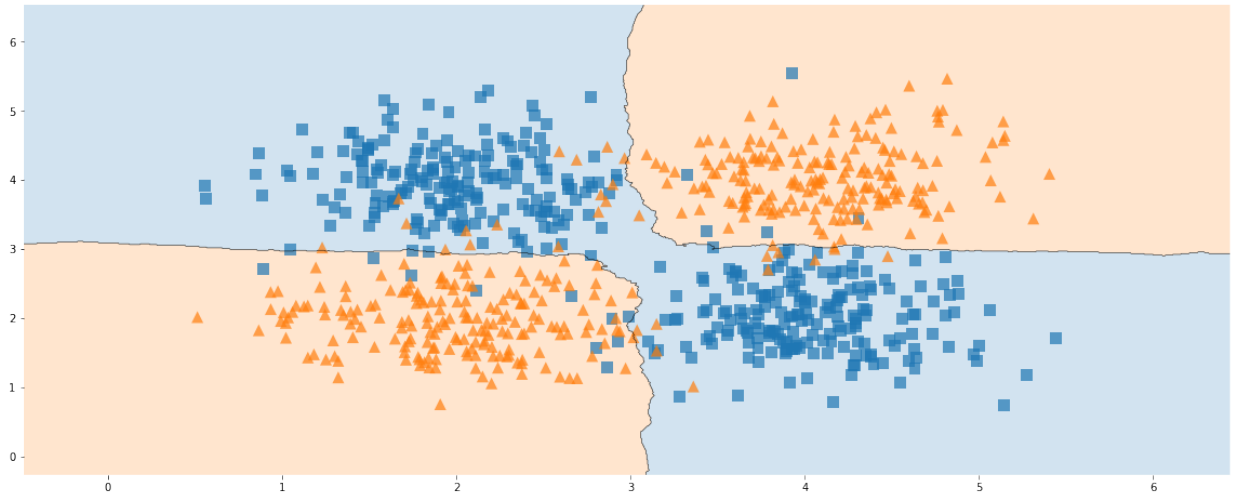
In [29]:
```
import matplotlib.pyplot as plt
from mlxtend.plotting import plot_decision_regions
knn = KNeighborsClassifier(n_neighbors=5)
#svm_poly_5.fit(Data1_X, Data1_Y)
knn.fit(Data2_X, Data2_Y)
scatter_kwargs = {'s': 120, 'edgecolor':'none', 'alpha': 0.7}
contourf_kwargs = {'alpha': 0.2}
scatter_highlight_kwargs = {'s': 120, 'label': 'Test data', 'alpha': 0
.7}
fig = plt.figure(figsize=(20, 8))
fig = plot_decision_regions(X=Data2_X, y=Data2_Y, clf=knn, legend=0,sc
atter_kwargs=scatter_kwargs,
                      contourf_kwargs=contourf_kwargs,
                      scatter_highlight_kwargs=scatter_highlight_kwarg
s)
plt.show()
```



In [30]:
```
import matplotlib.pyplot as plt
from mlxtend.plotting import plot_decision_regions
knn = KNeighborsClassifier(n_neighbors=10)
#svm_poly_5.fit(Data1_X, Data1_Y)
knn.fit(Data2_X, Data2_Y)
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}
fig = plt.figure(figsize=(20, 8))
fig = plot_decision_regions(X=Data2_X, y=Data2_Y, clf=knn, legend=0,sc
atter_kwargs=scatter_kwargs,
                      contourf_kwargs=contourf_kwargs,
                      scatter_highlight_kwargs=scatter_highlight_kwarg
s)
plt.show()
```



In [31]:
```
import matplotlib.pyplot as plt
from mlxtend.plotting import plot_decision_regions
knn = KNeighborsClassifier(n_neighbors=50)
#svm_poly_5.fit(Data1_X, Data1_Y)
knn.fit(Data2_X, Data2_Y)
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}
fig = plt.figure(figsize=(20, 8))
fig = plot_decision_regions(X=Data2_X, y=Data2_Y, clf=knn, legend=0,sc
atter_kwargs=scatter_kwargs,
                            contourf_kwargs=contourf_kwargs,
                            scatter_highlight_kwargs=scatter_highlight_kwarg
s)
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.

As K value increases, we can see that out decision boundary refrain from noise and makes the curve smoother. At k=infinity, the model becomes simplest model(ie all points belong to single class whose training data dominates(majority class)). This leads to underfitting of the data(high bias and low variance).

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 [46]:
```
nb = GaussianNB()
nb_scores = cross_val_score(nb, Data2_X, Data2_Y, cv=10, scoring='accu
racy')
[nb_scores.mean(), nb_scores.std()]
```

Out[46]: [0.049999999999999996, 0.026809513236909017]

In [47]:
```
nb = GaussianNB()
nb_scores = cross_val_score(nb, Data1_X, Data1_Y, cv=10, scoring='accu
racy')
[nb_scores.mean(), nb_scores.std()]
```

Out[47]: [0.96750000000000003, 0.031721443851123791]

In [48]:
```
nb = GaussianNB()
nb_scores = cross_val_score(nb, Data3_X, Data3_Y, cv=10, scoring='accu
racy')
[nb_scores.mean(), nb_scores.std()]
```

Out[48]: [0.95999999999999996, 0.027838821814150098]

In [49]:
```
nb = GaussianNB()
nb_scores = cross_val_score(nb, Data4_X, Data4_Y, cv=10, scoring='accu
racy')
[nb_scores.mean(), nb_scores.std()]
```

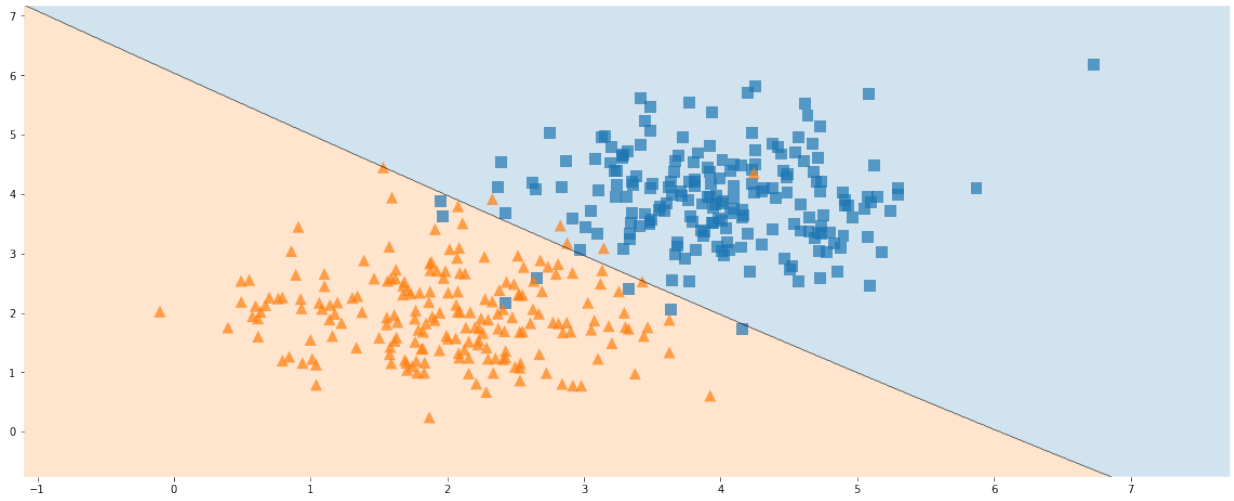
Out[49]: [0.96407362945178066, 0.026052087140989725]

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

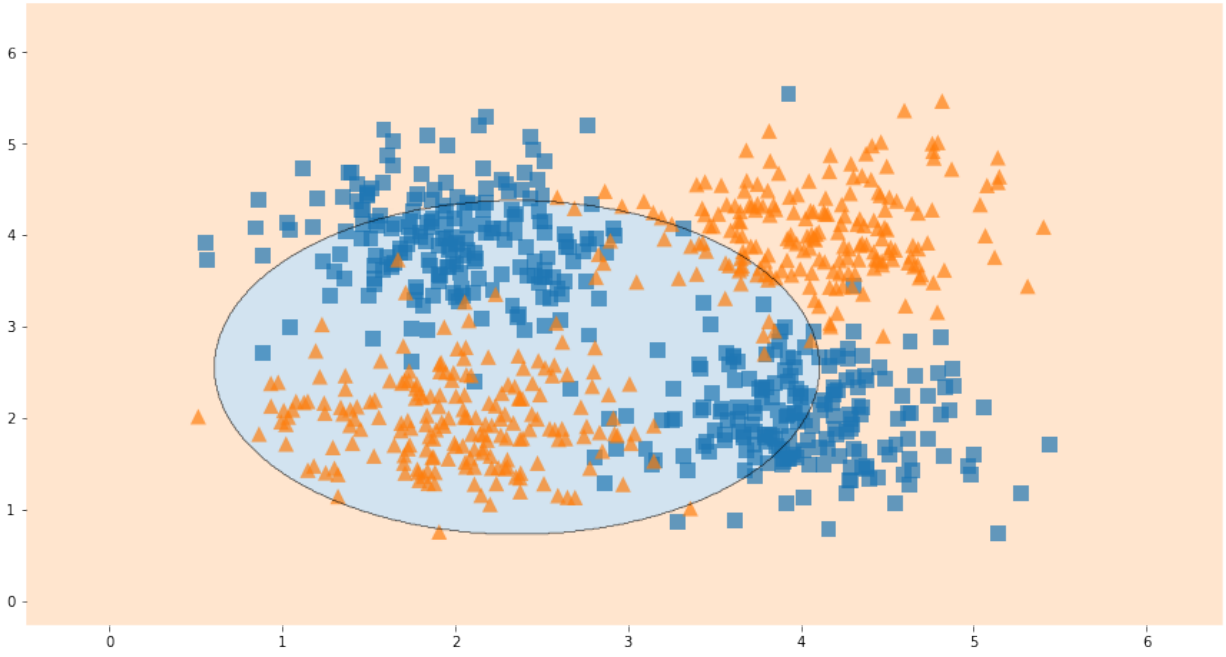
NB algorithm assumes independence of attributes. This makes the algorithm vulnerable to data which has covariance/correlation. Therefore it performs poorly on dataset 2.

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

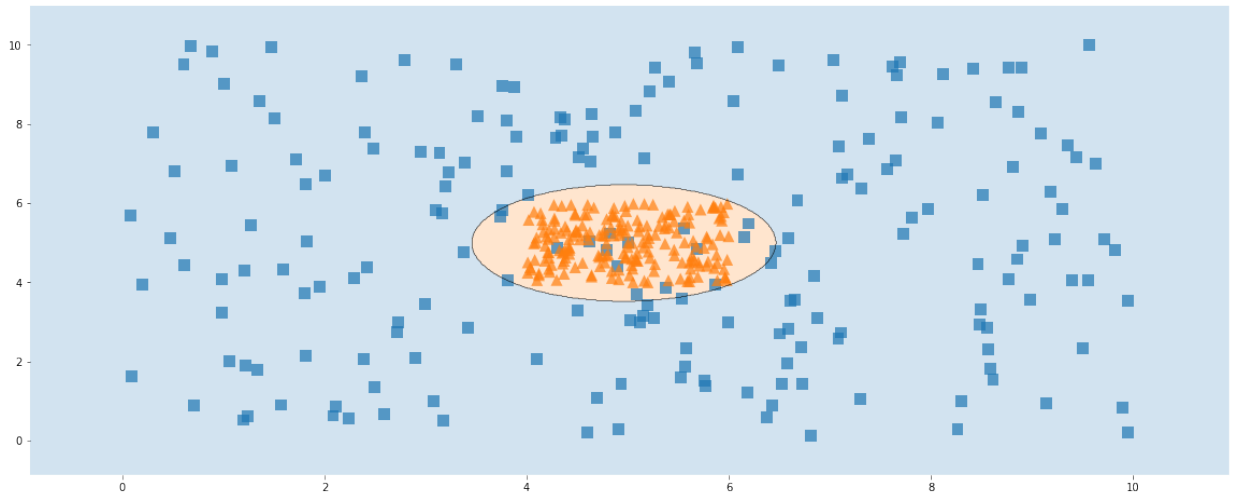
In [50]:
```
import matplotlib.pyplot as plt
from mlxtend.plotting import plot_decision_regions
nb = GaussianNB()
nb.fit(Data1_X, Data1_Y)
scatter_kwargs = {'s': 120, 'edgecolor':'none', 'alpha': 0.7}
contourf_kwargs = {'alpha': 0.2}
scatter_highlight_kwargs = {'s': 120, 'label': 'Test data', 'alpha': 0
.7}
fig = plt.figure(figsize=(20, 8))
fig = plot_decision_regions(X=Data1_X, y=Data1_Y, clf=nb, legend=0,sca
tter_kwargs=scatter_kwargs,
                      contourf_kwargs=contourf_kwargs,
                      scatter_highlight_kwargs=scatter_highlight_kwarg
s)
plt.show()
```



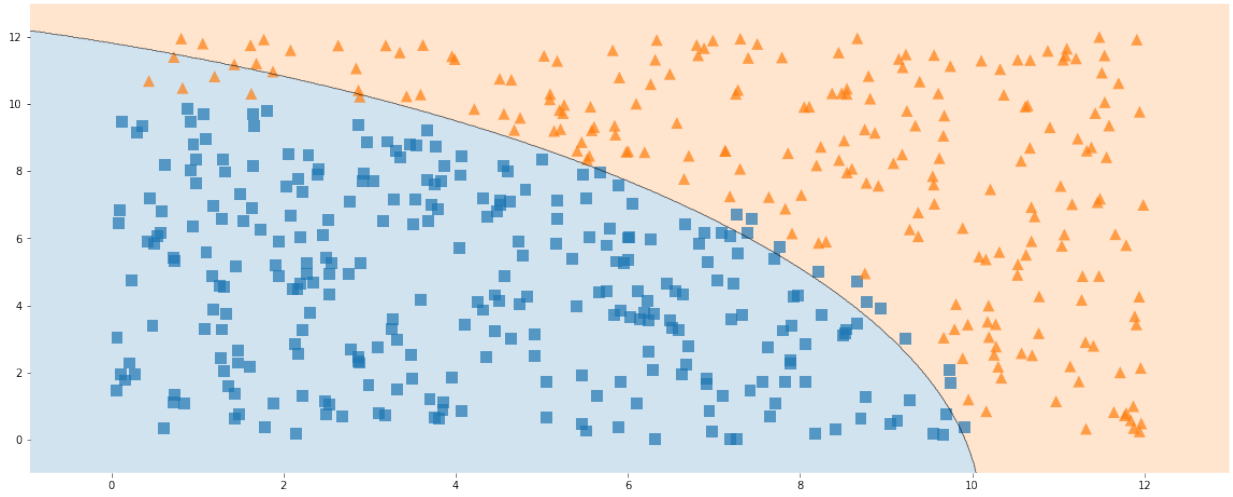
In [52]:
```
import matplotlib.pyplot as plt
from mlxtend.plotting import plot_decision_regions
nb = GaussianNB()
nb.fit(Data2_X, Data2_Y)
scatter_kwargs = {'s': 120, 'edgecolor':'none', 'alpha': 0.7}
contourf_kwargs = {'alpha': 0.2}
scatter_highlight_kwargs = {'s': 120, 'label': 'Test data', 'alpha': 0
.7}
fig = plt.figure(figsize=(15, 8))
fig = plot_decision_regions(X=Data2_X, y=Data2_Y, clf=nb, legend=0,sca
tter_kwargs=scatter_kwargs,
                      contourf_kwargs=contourf_kwargs,
                      scatter_highlight_kwargs=scatter_highlight_kwarg
s)
plt.show()
```



In [48]:
```
import matplotlib.pyplot as plt
from mlxtend.plotting import plot_decision_regions
nb = GaussianNB()
nb.fit(Data3_X, Data3_Y)
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}
fig = plt.figure(figsize=(20, 8))
fig = plot_decision_regions(X=Data3_X, y=Data3_Y, clf=nb, legend=0,sca
tter_kwargs=scatter_kwargs,
                      contourf_kwargs=contourf_kwargs,
                      scatter_highlight_kwargs=scatter_highlight_kwarg
s)
plt.show()
```



In [47]:
```
import matplotlib.pyplot as plt
from mlxtend.plotting import plot_decision_regions
nb = GaussianNB()
nb.fit(Data4_X, Data4_Y)
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}
fig = plt.figure(figsize=(20, 8))
fig = plot_decision_regions(X=Data4_X, y=Data4_Y, clf=nb, legend=0,sca
tter_kwargs=scatter_kwargs,
                      contourf_kwargs=contourf_kwargs,
                      scatter_highlight_kwargs=scatter_highlight_kwarg
s)
plt.show()
```



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

Ingeneral, decision surface of NB classifier is piecewise quadratic decision boundary for the Gaussian model.But when conditional covaiance matrices for both classes are same or almost similar(as in the case of Dataset 1) we can get a linear decision boundary.All boundaries except for the dataset 1 are quadratic in nature.

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

NB performs poorly on Dataset 2 since NB assumes independence of all the attributes where in dataset 2 , corre;ation/covariance exists between the data. Therefore we can say that NB poorly performed on that dataset.

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.

Linear SVM works best when the data is linearly seperable. On that account, we can rank the data as follows: 1>4>3>2. Since 1 is most linearly seperable it's ranked 1st. Accuracy also follows the same trend.

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 [50]:
```
svm_linear = SVC(C=0.5, kernel='linear')
svm_linear_scores = cross_val_score(svm_linear, Data2_X, Data2_Y, cv=1
0, scoring='accuracy')
[svm_linear_scores.mean(), svm_linear_scores.std()]
```

Out[50]:
```
[0.14125000000000001, 0.061249999999999992]
```

In [51]:
```
svm_linear = SVC(C=0.5, kernel='linear')
svm_linear_scores = cross_val_score(svm_linear, Data1_X, Data1_Y, cv=1
0, scoring='accuracy')
[svm_linear_scores.mean(), svm_linear_scores.std()]
```

Out[51]:
```
[0.96749999999999992, 0.02968585521759479]
```

In [52]:
```
svm_linear = SVC(C=0.5, kernel='linear')
svm_linear_scores = cross_val_score(svm_linear, Data3_X, Data3_Y, cv=1
0, scoring='accuracy')
[svm_linear_scores.mean(), svm_linear_scores.std()]
```

Out[52]:
```
[0.64250000000000007, 0.0275]
```

In [53]:
```
svm_linear = SVC(C=0.5, kernel='linear')
svm_linear_scores = cross_val_score(svm_linear, Data4_X, Data4_Y, cv=1
0, scoring='accuracy')
[svm_linear_scores.mean(), svm_linear_scores.std()]
```

Out[53]:
```
[0.9259463785514207, 0.031474819626331849]
```

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

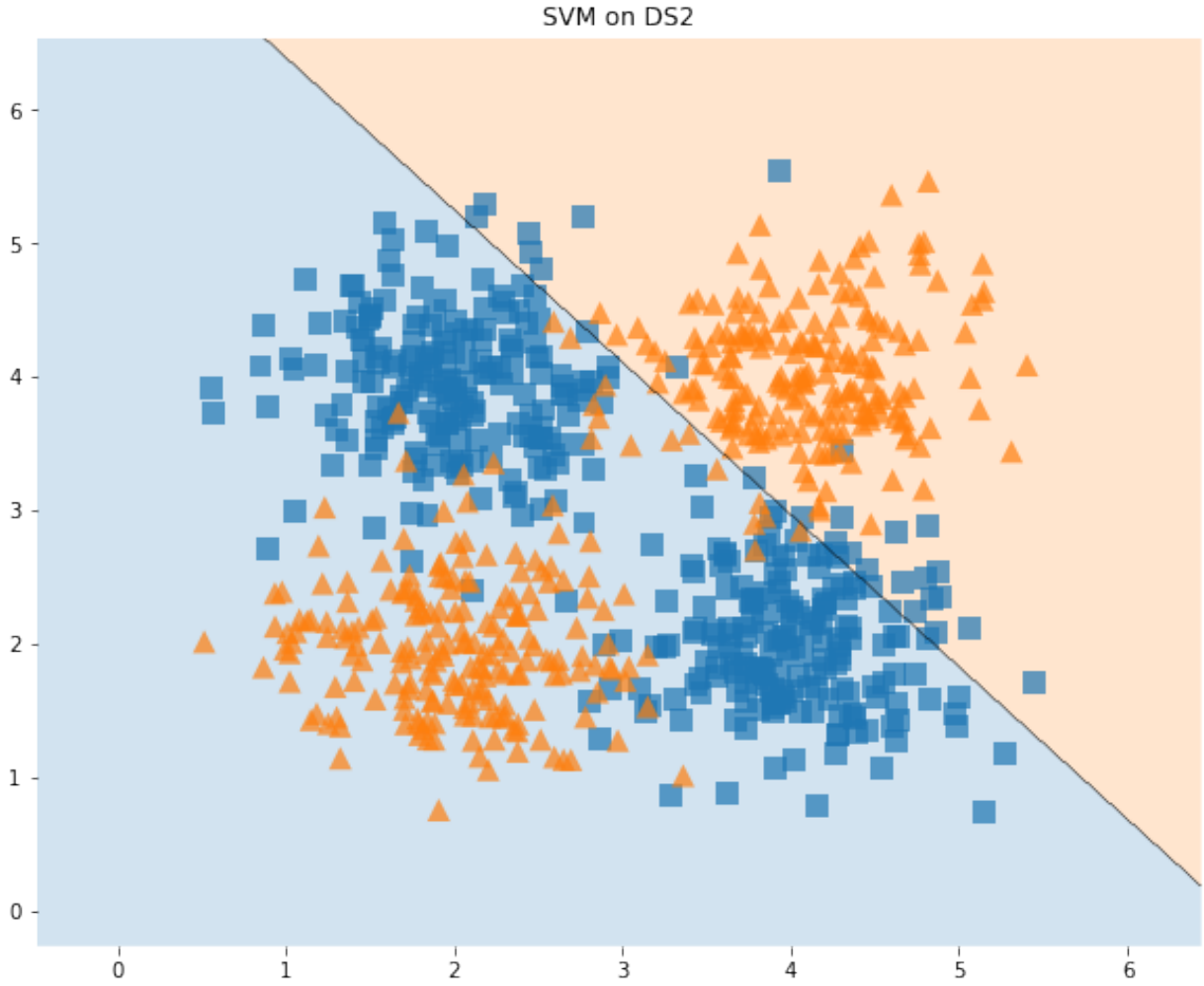
As stated above, 1 has highest accuracy followed by 4,3,2. 1>4>3>2.

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

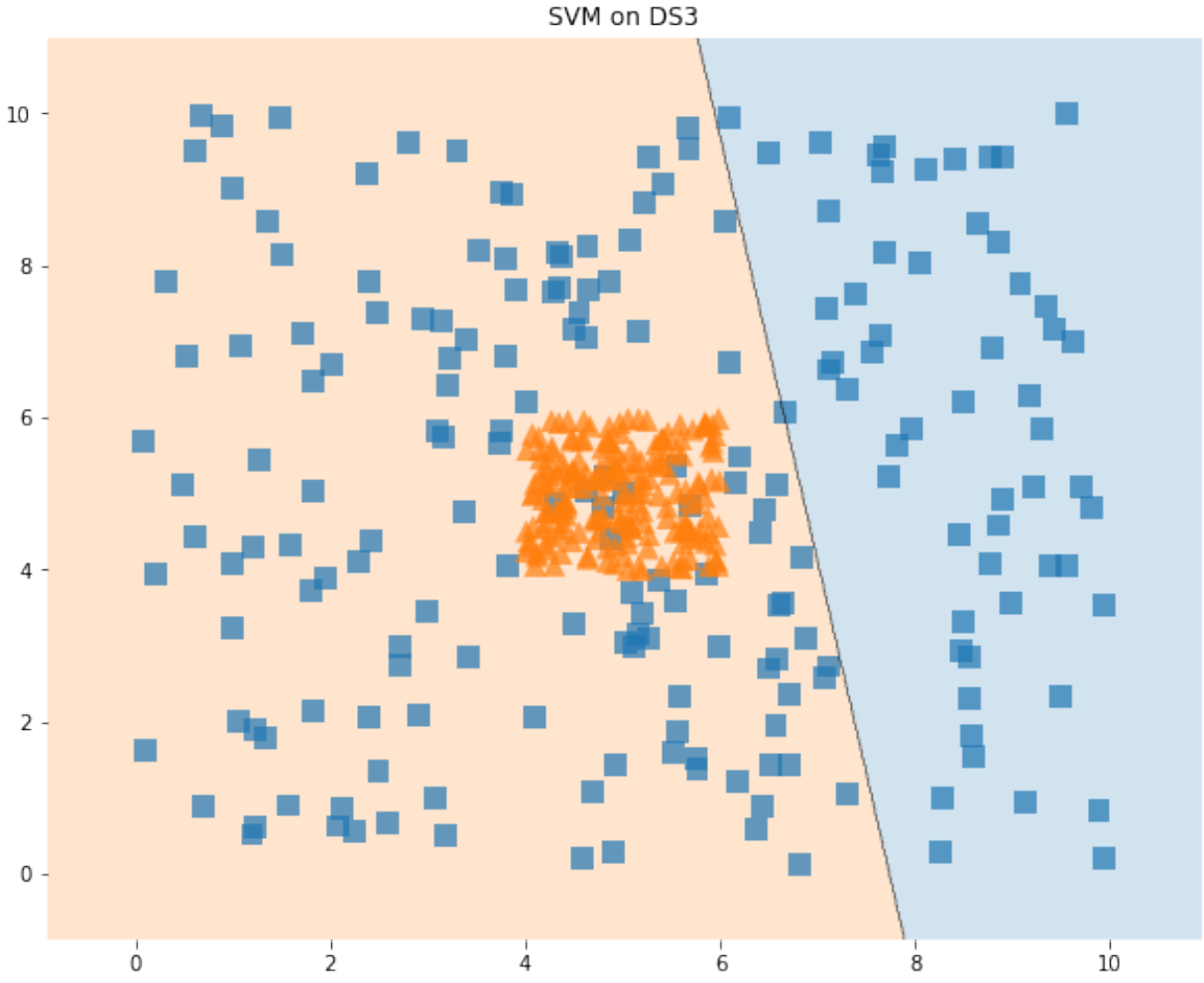
In [56]:
```
import matplotlib.pyplot as plt
from mlxtend.plotting import plot_decision_regions
svm_linear = SVC(C=0.5, kernel='linear')
svm_linear.fit(Data1_X, Data1_Y)
scatter_kwargs = {'s': 120, 'edgecolor':'none', 'alpha': 0.7}
contourf_kwargs = {'alpha': 0.2}
scatter_highlight_kwargs = {'s': 120, 'label': 'Test data', 'alpha': 0
.7}
fig = plt.figure(figsize=(10, 8))
fig = plot_decision_regions(X=Data1_X, y=Data1_Y, clf=svm_linear, lege
nd=0,scatter_kwargs=scatter_kwargs,
                      contourf_kwargs=contourf_kwargs,
                      scatter_highlight_kwargs=scatter_highlight_kwarg
s)
plt.title("SVM on DS1")
plt.show()
```


SVM on DS1
7
6
5
4
3
2
1
0
−1
0
1
2
3
4
5
6
7

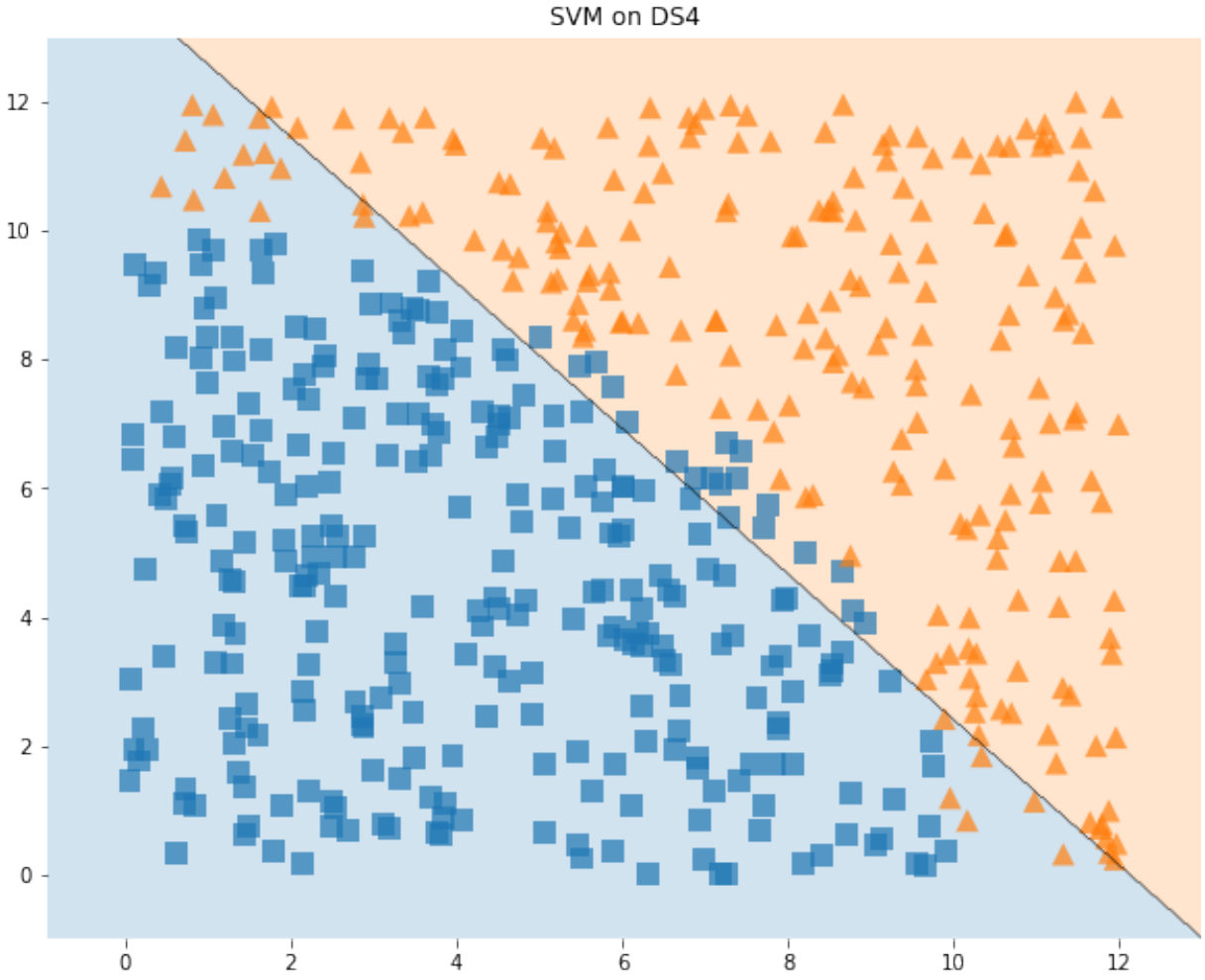
In [58]:
```
import matplotlib.pyplot as plt
from mlxtend.plotting import plot_decision_regions
svm_linear = SVC(C=0.5, kernel='linear')
svm_linear.fit(Data2_X, Data2_Y)
scatter_kwargs = {'s': 120, 'edgecolor':'none', 'alpha': 0.7}
contourf_kwargs = {'alpha': 0.2}
scatter_highlight_kwargs = {'s': 120, 'label': 'Test data', 'alpha': 0
.7}
fig = plt.figure(figsize=(10, 8))
fig = plot_decision_regions(X=Data2_X, y=Data2_Y, clf=svm_linear, lege
nd=0,scatter_kwargs=scatter_kwargs,
                      contourf_kwargs=contourf_kwargs,
                      scatter_highlight_kwargs=scatter_highlight_kwarg
s)
plt.title("SVM on DS2")
plt.show()
```



In [59]:
```
import matplotlib.pyplot as plt
from mlxtend.plotting import plot_decision_regions
svm_linear = SVC(C=0.5, kernel='linear')
svm_linear.fit(Data3_X, Data3_Y)
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}
fig = plt.figure(figsize=(10, 8))
fig = plot_decision_regions(X=Data3_X, y=Data3_Y, clf=svm_linear, lege
nd=0,scatter_kwargs=scatter_kwargs,
                      contourf_kwargs=contourf_kwargs,
                      scatter_highlight_kwargs=scatter_highlight_kwarg
s)
plt.title("SVM on DS3")
plt.show()
```



In [60]:
```
import matplotlib.pyplot as plt
from mlxtend.plotting import plot_decision_regions
svm_linear = SVC(C=0.5, kernel='linear')
svm_linear.fit(Data4_X, Data4_Y)
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}
fig = plt.figure(figsize=(10, 8))
fig = plot_decision_regions(X=Data4_X, y=Data4_Y, clf=svm_linear, lege
nd=0,scatter_kwargs=scatter_kwargs,
                      contourf_kwargs=contourf_kwargs,
                      scatter_highlight_kwargs=scatter_highlight_kwarg
s)
plt.title("SVM on DS4")
plt.show()
```



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

As stated earlier, when data is seperable, drawing hyperplane with good margin using support vectors with good accuracy is possible. Dataset 1 has linearly seperable data.This gives the provision to build support vectors and hyperplane with good margin.Data2 and Data3 are not linearly seperable. Therefor accuracy of linear svm drops(as seen in plots)

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 = SVC(C=0.5, kernel='poly',degree=1, gamma = 'auto')
svm_poly_scores = cross_val_score(svm_poly, Data2_X, Data2_Y, cv=10, s
coring='accuracy')
[svm_poly_scores.mean(), svm_poly_scores.std()]
```

Out[63]:
```
[0.13375, 0.05215661511256266]
```

In [62]:
```
svm_poly = SVC(C=0.5, kernel='poly',degree=2, gamma = 'auto')
svm_poly_scores = cross_val_score(svm_poly, Data2_X, Data2_Y, cv=10, s
coring='accuracy')
[svm_poly_scores.mean(), svm_poly_scores.std()]
```

Out[62]:
```
[0.865, 0.030516389039334235]
```

In [61]:
```
svm_poly = SVC(C=0.5, kernel='poly',degree=3, gamma = 'auto')
svm_poly_scores = cross_val_score(svm_poly, Data2_X, Data2_Y, cv=10, s
coring='accuracy')
[svm_poly_scores.mean(), svm_poly_scores.std()]
```

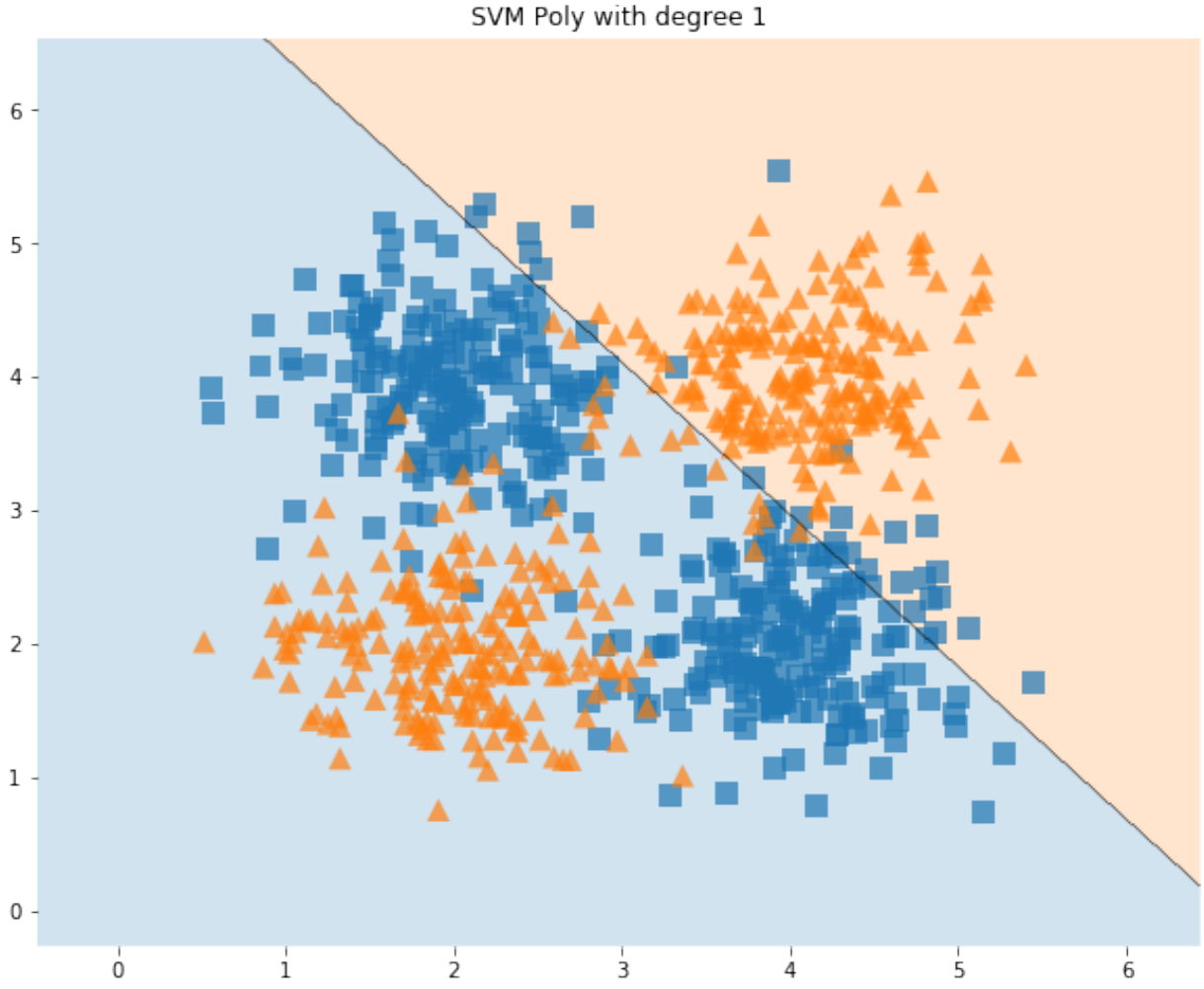
Out[61]:
```
[0.8762500000000001, 0.026487025125521375]
```

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

degree 3> degree 2 > degree 1.

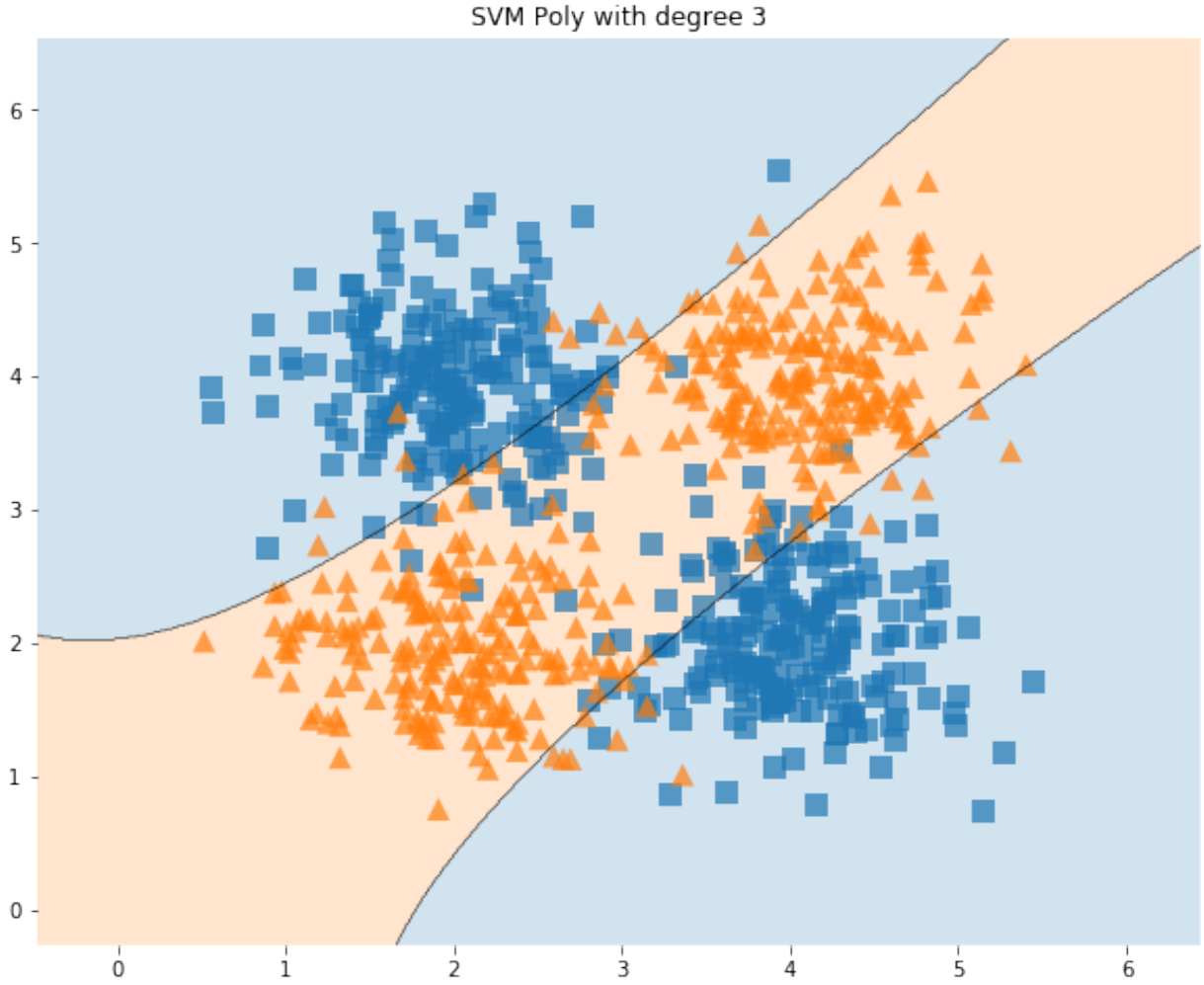
Question 5c: Plot decision regions for a polynomial kernel SVM with degree values 1, 2, and 3.

In [69]:
```
import matplotlib.pyplot as plt
from mlxtend.plotting import plot_decision_regions
svm_poly = SVC(C=0.5, kernel='poly',degree=1, gamma = 'auto')
svm_poly.fit(Data2_X, Data2_Y)
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}
fig = plt.figure(figsize=(10, 8))
fig = plot_decision_regions(X=Data2_X, y=Data2_Y, clf=svm_poly, legend
=0,scatter_kwargs=scatter_kwargs,
                      contourf_kwargs=contourf_kwargs,
                      scatter_highlight_kwargs=scatter_highlight_kwarg
s)
plt.title("SVM Poly with degree 1")
plt.show()
```

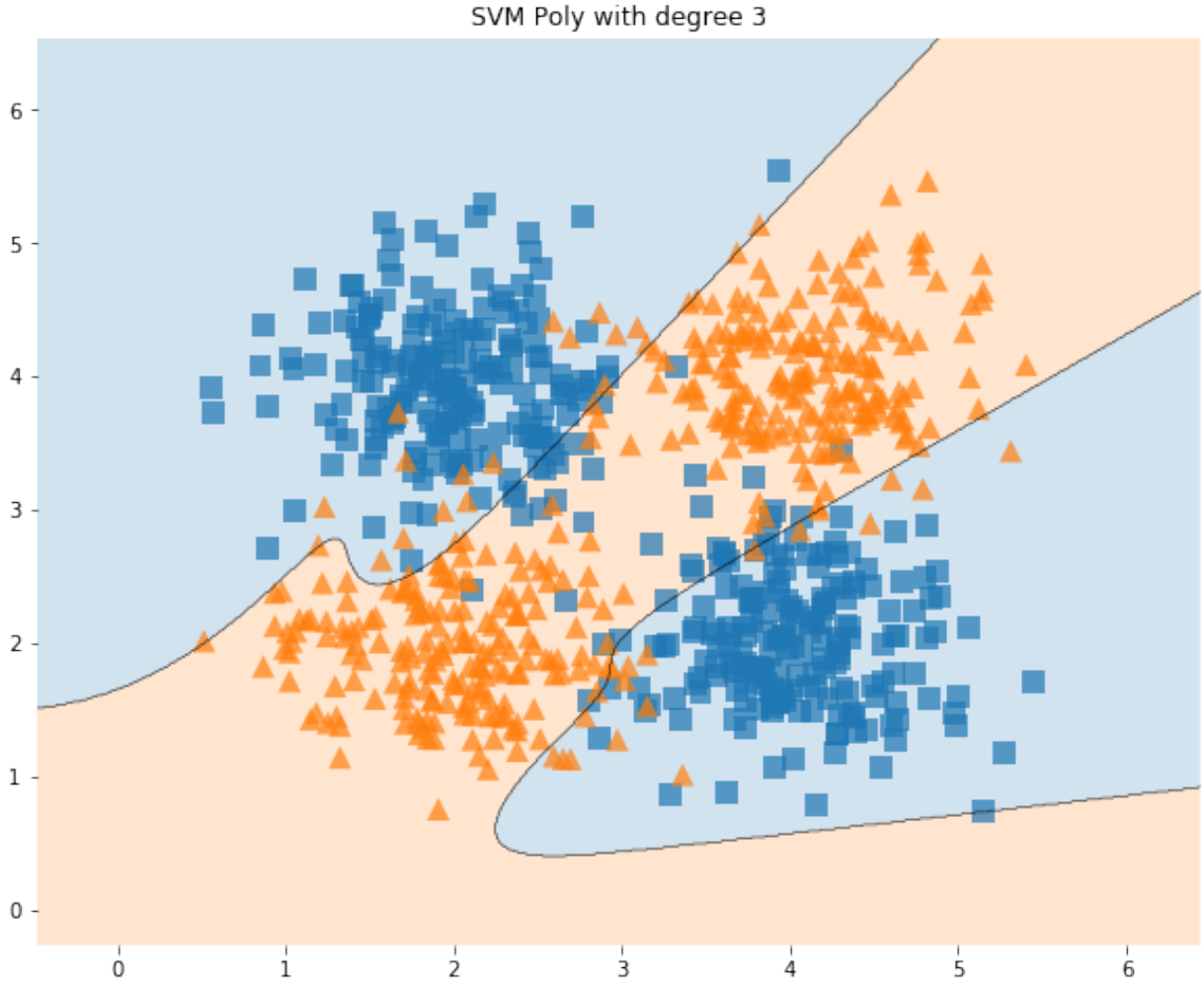

SVM Poly with degree 1
0
1
2
3
4
5
6

In [68]:
```
import matplotlib.pyplot as plt
from mlxtend.plotting import plot_decision_regions
svm_poly = SVC(C=0.5, kernel='poly',degree=2, gamma = 'auto')
svm_poly.fit(Data2_X, Data2_Y)
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}
fig = plt.figure(figsize=(10, 8))
fig = plot_decision_regions(X=Data2_X, y=Data2_Y, clf=svm_poly, legend
=0,scatter_kwargs=scatter_kwargs,
                      contourf_kwargs=contourf_kwargs,
                      scatter_highlight_kwargs=scatter_highlight_kwarg
s)
plt.title("SVM Poly with degree 2")
plt.show()
```

In [67]:
```
import matplotlib.pyplot as plt
from mlxtend.plotting import plot_decision_regions
svm_poly = SVC(C=0.5, kernel='poly',degree=3, gamma = 'auto')
svm_poly.fit(Data2_X, Data2_Y)
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}
fig = plt.figure(figsize=(10, 8))
fig = plot_decision_regions(X=Data2_X, y=Data2_Y, clf=svm_poly, legend
=0,scatter_kwargs=scatter_kwargs,
                      contourf_kwargs=contourf_kwargs,
                      scatter_highlight_kwargs=scatter_highlight_kwarg
s)
plt.title("SVM Poly with degree 3")
plt.show()
```



In [119]:
```
import matplotlib.pyplot as plt
from mlxtend.plotting import plot_decision_regions
svm_poly = SVC(C=0.5, kernel='poly',degree=10, gamma = 'auto')
svm_poly.fit(Data2_X, Data2_Y)
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}
fig = plt.figure(figsize=(10, 8))
fig = plot_decision_regions(X=Data2_X, y=Data2_Y, clf=svm_poly, legend
=0,scatter_kwargs=scatter_kwargs,
                      contourf_kwargs=contourf_kwargs,
                      scatter_highlight_kwargs=scatter_highlight_kwarg
s)
plt.title("SVM Poly with degree 10)
plt.show()
```



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

As per the plots, as degree increases, our decision surface tend to classify the data accurately.As we increase the degree, problem of overfitting can arise.(as we see with degree=10)

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 [60]:
```
svm_rbf1 = SVC(C = 0.5, kernel='rbf', gamma=0.1)
svm_rbf_scores = cross_val_score(svm_rbf, Data2_X, Data2_Y, cv=10, sco
ring='accuracy')
[svm_rbf_scores.mean(), svm_rbf_scores.std()]
```

Out[60]:
```
[0.93625000000000003, 0.029288436284649961]
```

In [61]:
```
svm_rbf2 = SVC(C = 0.5, kernel='rbf', gamma=0.01)
svm_rbf_scores = cross_val_score(svm_rbf, Data2_X, Data2_Y, cv=10, sco
ring='accuracy')
[svm_rbf_scores.mean(), svm_rbf_scores.std()]
```

Out[61]:
```
[0.30124999999999996, 0.087043451792768414]
```

In [64]:
```
svm_rbf3 = SVC(C = 0.5, kernel='rbf', gamma=1)
svm_rbf_scores = cross_val_score(svm_rbf, Data2_X, Data2_Y, cv=10, sco
ring='accuracy')
[svm_rbf_scores.mean(), svm_rbf_scores.std()]
```

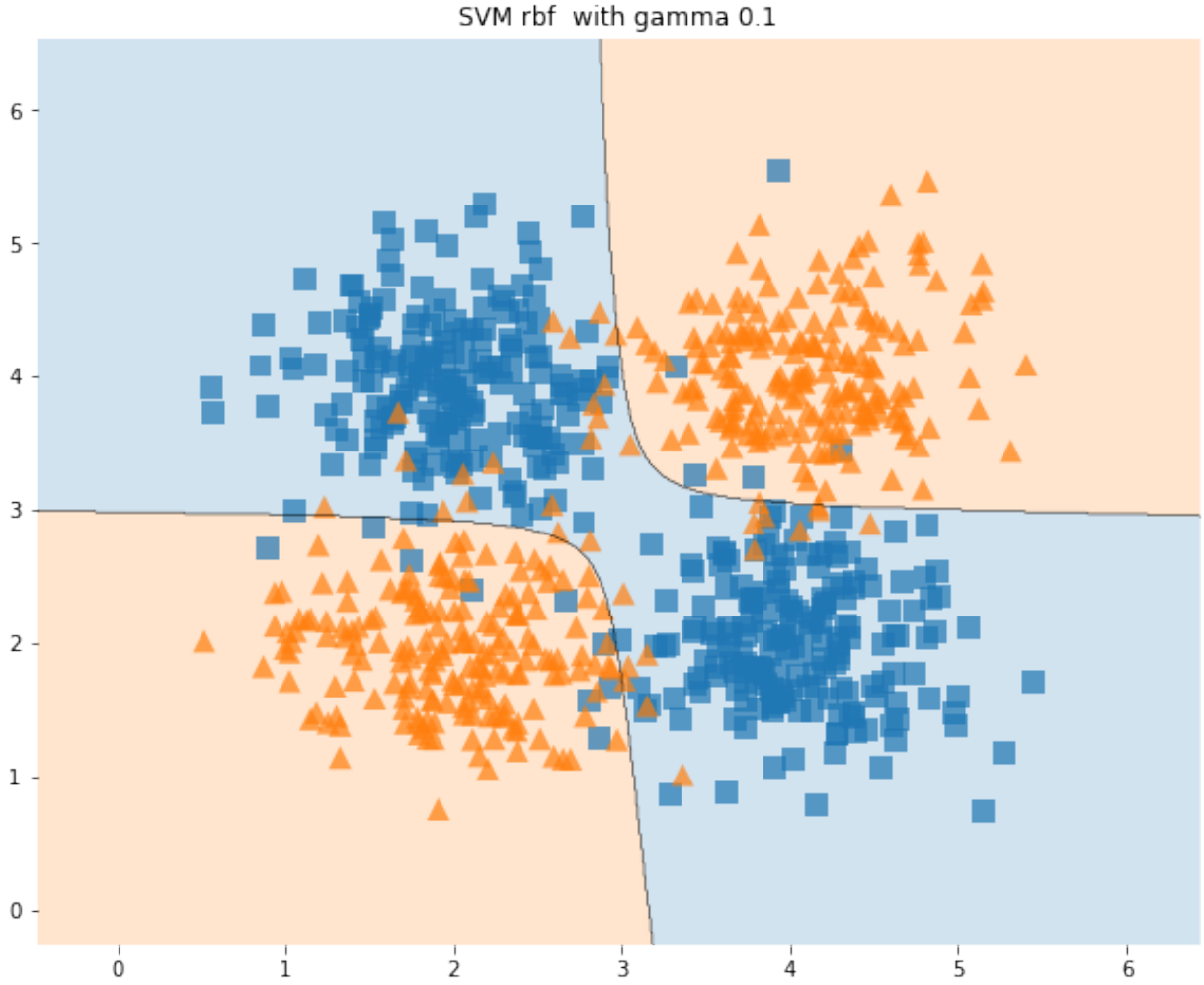
Out[64]:
```
[0.93999999999999984, 0.029474565306379]
```

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

rbf3>rbf1>rbf2

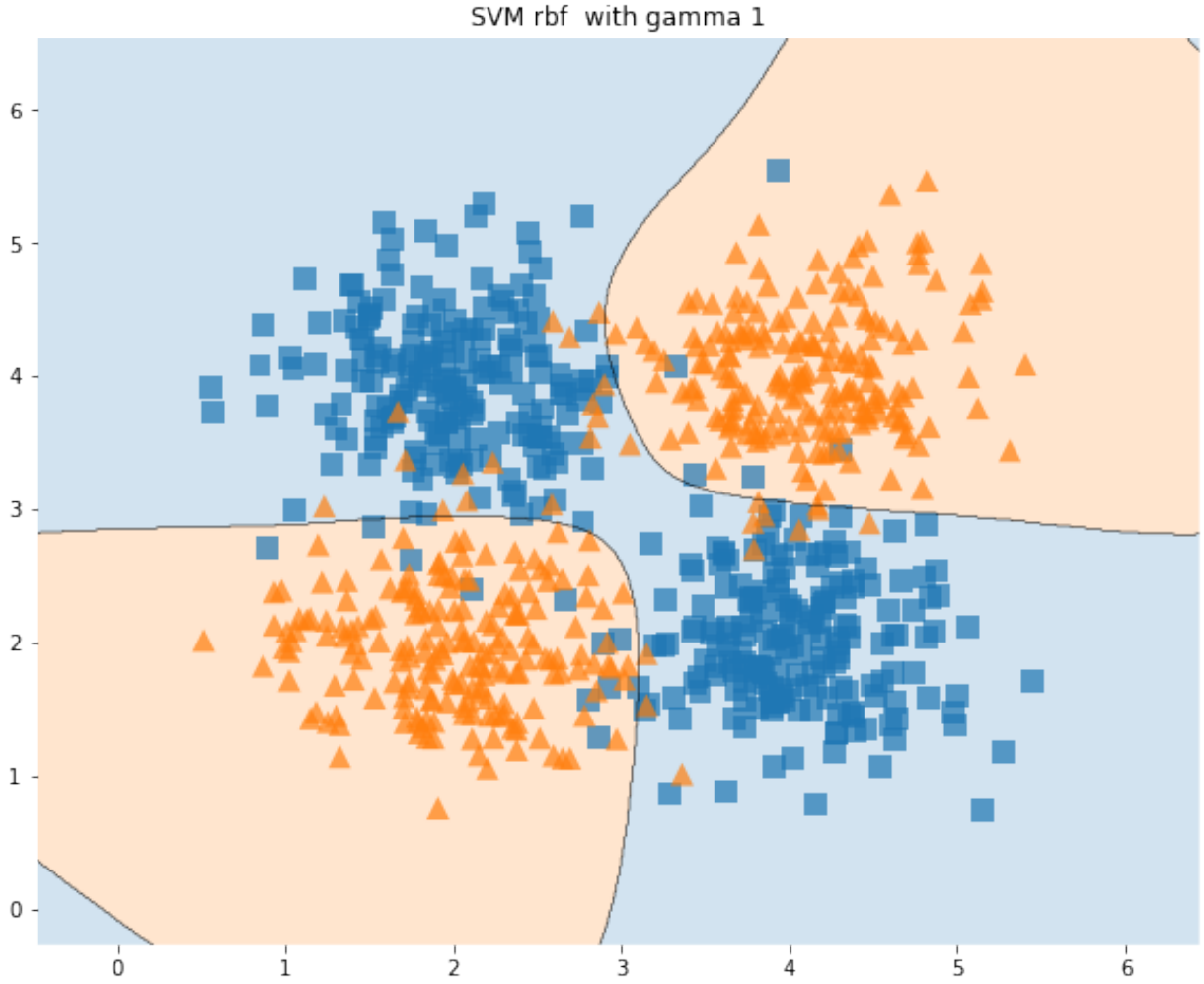
Question 5g: Plot decision regions for the above RBF Kernels

In [70]:
```
import matplotlib.pyplot as plt
from mlxtend.plotting import plot_decision_regions
svm_rbf = SVC(C = 0.5, kernel='rbf', gamma=0.1)
svm_rbf.fit(Data2_X, Data2_Y)
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}
fig = plt.figure(figsize=(10, 8))
fig = plot_decision_regions(X=Data2_X, y=Data2_Y, clf=svm_rbf, legend=
0,scatter_kwargs=scatter_kwargs,
                      contourf_kwargs=contourf_kwargs,
                      scatter_highlight_kwargs=scatter_highlight_kwarg
s)
plt.title("SVM rbf  with gamma 0.1")
plt.show()
```



In [71]:
```
import matplotlib.pyplot as plt
from mlxtend.plotting import plot_decision_regions
svm_rbf = SVC(C = 0.5, kernel='rbf', gamma=0.01)
svm_rbf.fit(Data2_X, Data2_Y)
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}
fig = plt.figure(figsize=(10, 8))
fig = plot_decision_regions(X=Data2_X, y=Data2_Y, clf=svm_rbf, legend=
0,scatter_kwargs=scatter_kwargs,
                      contourf_kwargs=contourf_kwargs,
                      scatter_highlight_kwargs=scatter_highlight_kwarg
s)
plt.title("SVM rbf  with gamma 0.01")
plt.show()
```

In [72]:
```
import matplotlib.pyplot as plt
from mlxtend.plotting import plot_decision_regions
svm_rbf = SVC(C = 0.5, kernel='rbf', gamma=1)
svm_rbf.fit(Data2_X, Data2_Y)
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}
fig = plt.figure(figsize=(10, 8))
fig = plot_decision_regions(X=Data2_X, y=Data2_Y, clf=svm_rbf, legend=
0,scatter_kwargs=scatter_kwargs,
                      contourf_kwargs=contourf_kwargs,
                      scatter_highlight_kwargs=scatter_highlight_kwarg
s)
plt.title("SVM rbf  with gamma 1")
plt.show()
```



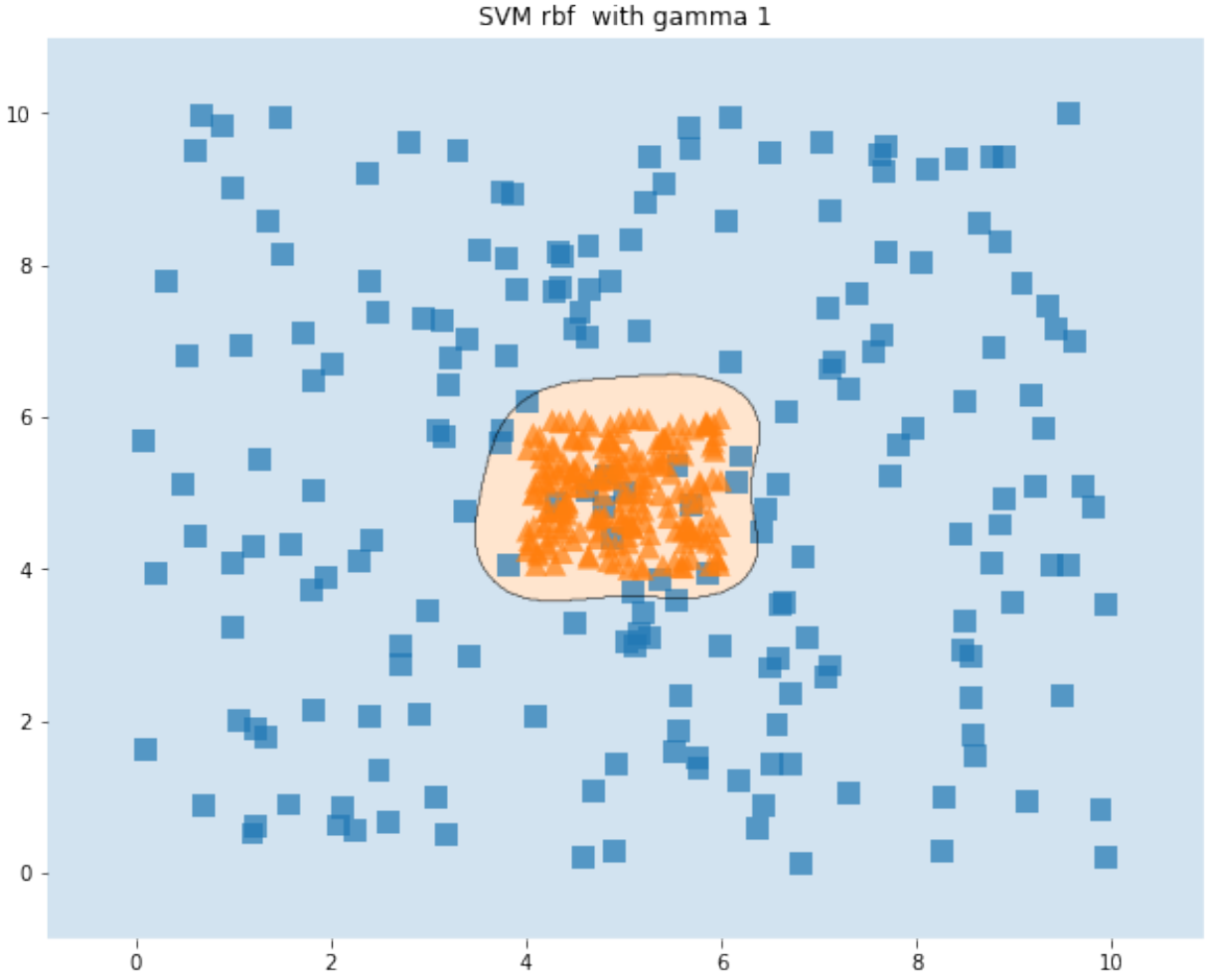
Question 5h: Explain the reason for your observations in **Question 5f** from the above decision regions.

As gamma parameter increases, our model tries to fit the training data. As gamma value increases, overfitting might occur. Gamma is inverse of the radius of influence of samples selected by the model as support vectors.

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

RBF kernel is suited for data3 as we see better decision boundaries made by rbf kernel.when we see the accuracy, RBF has good accuracy over polynomial kernel. Therefore rbf kernel is preferrable.

In [122]:
```
import matplotlib.pyplot as plt
from mlxtend.plotting import plot_decision_regions
svm_rbf = SVC(C = 0.5, kernel='rbf', gamma=1)
svm_rbf.fit(Data3_X, Data3_Y)
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}
fig = plt.figure(figsize=(10, 8))
fig = plot_decision_regions(X=Data3_X, y=Data3_Y, clf=svm_rbf, legend=
0,scatter_kwargs=scatter_kwargs,
                      contourf_kwargs=contourf_kwargs,
                      scatter_highlight_kwargs=scatter_highlight_kwarg
s)
plt.title("SVM rbf  with gamma 1")
plt.show()
```



In [ ]:
```
import matplotlib.pyplot as plt
from mlxtend.plotting import plot_decision_regions
svm_poly = SVC(C=0.5, kernel='poly',degree=10, gamma = 'auto')
svm_poly.fit(Data3_X, Data3_Y)
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}
fig = plt.figure(figsize=(10, 8))
fig = plot_decision_regions(X=Data3_X, y=Data3_Y, clf=svm_poly, legend
=0,scatter_kwargs=scatter_kwargs,
                            contourf_kwargs=contourf_kwargs,scatter_highligh
t_kwargs=scatter_highlight_kwargs)
plt.title("SVM Poly with degree 10")
plt.show()
```

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 [65]:
```
svm_rbf = SVC(C = 0.5, kernel='rbf', gamma=0.1)
svm_rbf_scores = cross_val_score(svm_rbf, Data2_X, Data2_Y, cv=3, scor
ing='accuracy')
[svm_rbf_scores.mean(), svm_rbf_scores.std()]
```

Out[65]: [0.90382673100662103, 0.025725635665399284]

In [66]:
```
svm_rbf = SVC(C = 0.5, kernel='rbf', gamma=0.1)
svm_rbf_scores = cross_val_score(svm_rbf, Data2_X, Data2_Y, cv=4, scor
ing='accuracy')
[svm_rbf_scores.mean(), svm_rbf_scores.std()]
```

Out[66]: [0.91625000000000001, 0.016345871038277501]

In [67]:
```
svm_rbf = SVC(C = 0.5, kernel='rbf', gamma=0.1)
svm_rbf_scores = cross_val_score(svm_rbf, Data2_X, Data2_Y, cv=5, scor
ing='accuracy')
[svm_rbf_scores.mean(), svm_rbf_scores.std()]
```

Out[67]: [0.92749999999999999, 0.021866069605669881]

In [68]:
```
svm_rbf = SVC(C = 0.5, kernel='rbf', gamma=0.1)
svm_rbf_scores = cross_val_score(svm_rbf, Data2_X, Data2_Y, cv=6, scor
ing='accuracy')
[svm_rbf_scores.mean(), svm_rbf_scores.std()]
```

Out[68]: [0.93255314337403894, 0.021831954406404414]

In [72]:
```
svm_rbf = SVC(C = 0.5, kernel='rbf', gamma=0.1)
svm_rbf_scores = cross_val_score(svm_rbf, Data2_X, Data2_Y, cv=40, sco
ring='accuracy')
[svm_rbf_scores.mean(), svm_rbf_scores.std()]
```

Out[72]: [0.94000000000000006, 0.047696960070847283]

As seen above, as k increases, number of folods for training data increase which exposes our model to diverse training data. Hence our accuracy increases.Our data here is divided into k folds in which every time one fold is preserved for testing the model.Hence as k increases, training increases and accuracy 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 [83]:
```
#DT
dt = DecisionTreeClassifier(max_depth=50)
dt_scores = cross_val_score(dt, Data3_X, Data3_Y, cv=30, scoring='accu
racy')
dt_scores_p = cross_val_score(dt, Data3_X, Data3_Y, cv=30, scoring='pr
ecision' )
print ("DT - Accuracy : " + str(dt_scores.mean()) + " Precision : " +
str(dt_scores_p.mean()))
#knn
knn = KNeighborsClassifier(n_neighbors=5)
knn_scores = cross_val_score(knn, Data3_X, Data3_Y, cv=30, scoring='ac
curacy')
knn_scores_p = cross_val_score(knn, Data3_X, Data3_Y, cv=30, scoring='
precision')
print ("KNN - Accuracy : " + str(knn_scores.mean()) + " Precision : "
+ str(knn_scores_p.mean()))
#nb
nb = GaussianNB()
nb_scores = cross_val_score(nb, Data3_X, Data3_Y, cv=30, scoring='accu
racy')
nb_scores_p = cross_val_score(nb, Data3_X, Data3_Y, cv=30, scoring='pr
ecision' )
print ("NB - Accuracy : " + str(nb_scores.mean()) + " Precision : " +
str(nb_scores_p.mean()))
#SVM_Linear
svm_linear = SVC(C=0.5, kernel='linear')
svm_linear_scores = cross_val_score(svm_linear, Data3_X, Data3_Y, cv=3
0, scoring='accuracy')
svm_linear_scores_p = cross_val_score(svm_linear, Data3_X, Data3_Y, cv
=30, scoring='precision')
print("Linear SVM - Accuracy : " + str(svm_linear_scores.mean()) + " P
recision : " + str(svm_linear_scores_p.mean()))
#SVM_Polynomial
svm_poly = SVC(C=0.5, kernel='poly',degree=2, gamma = 'auto')
svm_poly_scores = cross_val_score(svm_poly, Data3_X, Data3_Y, cv=30, s
coring= 'accuracy')
svm_poly_scores_p = cross_val_score(svm_poly, Data3_X, Data3_Y, cv=30,
scoring ='precision')
print("Polynomial SVM - Accuracy : " + str(svm_poly_scores.mean()) + "
Precisi on : " + str(svm_poly_scores_p.mean()))
#SVM_Rbf
svm_rbf = SVC(C = 0.5, kernel='rbf', gamma=3)
svm_rbf_scores = cross_val_score(svm_rbf, Data3_X, Data3_Y, cv=30, sco
ring='accuracy')
svm_rbf_scores_p = cross_val_score(svm_rbf, Data3_X, Data3_Y, cv=30, s
coring='precision')
print("SVM_RBF - Accuracy : " + str(svm_rbf_scores.mean()) + " Precisi
on : " + str(svm_rbf_scores_p.mean()))
```

```
DT - Accuracy : 0.9468253968253969 Precision : 0.9567460317460319
KNN - Accuracy : 0.9492063492063493 Precision : 0.9188311688311689
NB - Accuracy : 0.9591269841269843 Precision : 0.9328571428571429
Linear SVM - Accuracy : 0.6428571428571429 Precision : 0.58817016317
01632
Polynomial SVM - Accuracy : 0.8484126984126984 Precisi on : 0.792371
3323713325
SVM_RBF - Accuracy : 0.9662698412698414 Precision : 0.94535714285714
29
```

When accuracy is considered: DT>NB>KNN>RBF>SVM(polynomial)>SVM when precision is considered: DT>NB>KNN>RBF>SVM(polynomial)>SVM Linear Accuracy and precison are different measures. Precision(positive class classification) is class based where as accuracy is not. Accuracy is based on the whole of training data. For NB above, precision and accuracy are not same. But high accuracy reflects high precision as well. But not vice versa.

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 [88]:
```
svm_linear = SVC(C=0.5, kernel='linear')
svm_linear_scores = cross_val_score(svm_linear, Data3_X, Data3_Y, cv=5
, scoring='accuracy')
print(svm_linear_scores.mean(), svm_linear_scores.std())
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_estimator
s=n_est)
    # compute cross-validation accuracy for each bagging classifier
    scores = cross_val_score(bagging, Data3_X, Data3_Y, cv=5, scoring=
'accuracy')
    print("Bagging Accuracy: %.2f (+/- %.2f) #estimators: %d" % (score
s.mean(), scores.std(), n_est))
```

```
0.63 0.02573907535246751
Bagging Accuracy: 0.59 (+/- 0.05) #estimators: 1
Bagging Accuracy: 0.58 (+/- 0.11) #estimators: 2
Bagging Accuracy: 0.62 (+/- 0.10) #estimators: 3
Bagging Accuracy: 0.57 (+/- 0.09) #estimators: 4
Bagging Accuracy: 0.66 (+/- 0.10) #estimators: 5
Bagging Accuracy: 0.69 (+/- 0.05) #estimators: 10
Bagging Accuracy: 0.68 (+/- 0.07) #estimators: 20
```

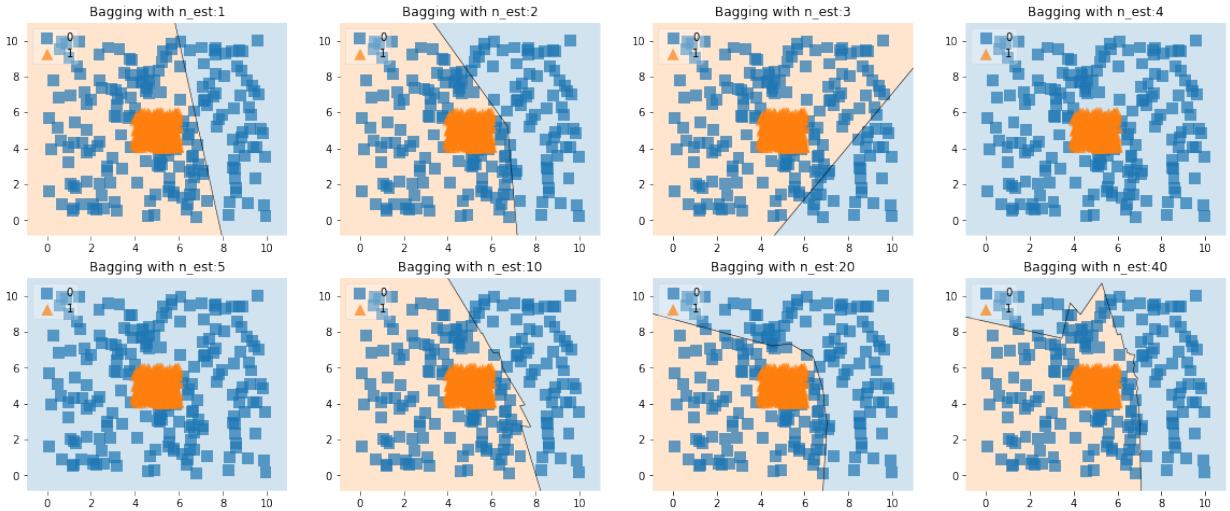
As we change the estimators, we see change(probably reduction) in variance and also different mean for different estimators.

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

In [103]:
```
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_estimator
s=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, leg
end=2,
                            scatter_kwargs=scatter_kwargs,
                            contourf_kwargs=contourf_kwargs,
                            scatter_highlight_kwargs=scatter_highlight_kwa
rgs)
    plt.title('Bagging with n_est:'+str(n_est))

plt.show()
```

```
/users/PES0801/krishnateja0612/.local/lib/python3.6/site-packages/ml
xtend/plotting/decision_regions.py:247: UserWarning: No contour leve
ls were found within the data range.
  antialiased=True)
/users/PES0801/krishnateja0612/.local/lib/python3.6/site-packages/ml
xtend/plotting/decision_regions.py:247: UserWarning: No contour leve
ls 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.

As we increase estimators, performance of the classifier increased. We had more misclassified data with one estimator.

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.

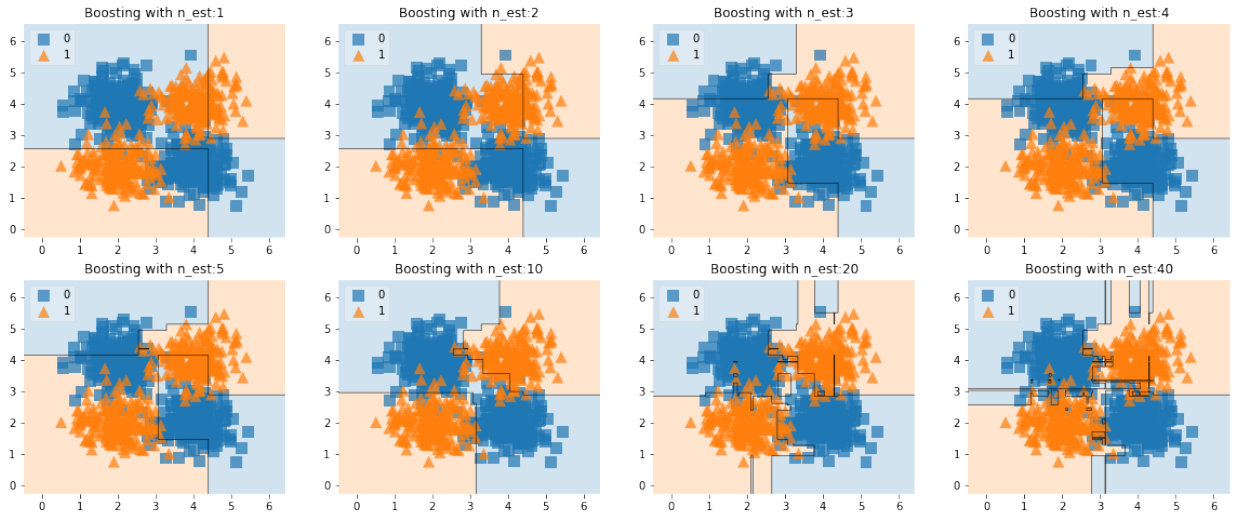
In [101]:
```
dt = DecisionTreeClassifier(max_depth=2)
dt_scores = cross_val_score(dt, Data3_X, Data3_Y, cv=10, scoring='accu
racy')
print(dt_scores.mean(), dt_scores.std())
n_est_list = [1,2,3,4,5,10,20,40]
for n_est in n_est_list:
    # create an instance of a boosting classifier with 'n_est' estimat
ors
    boosting = AdaBoostClassifier(base_estimator=dt, n_estimators=n_es
t)
    # compute cross-validation accuracy for each bagging classifier
    scores = cross_val_score(boosting, Data2_X, Data2_Y, cv=10, scorin
g='accuracy')
    print("Boosting Accuracy: %.2f (+/- %.4f) #estimators: %d" % (scor
es.mean (), scores.std(), n_est))
```

```
0.875 0.03354101966249685
Boosting Accuracy: 0.88 (+/- 0.0281) #estimators: 1
Boosting Accuracy: 0.88 (+/- 0.0292) #estimators: 2
Boosting Accuracy: 0.90 (+/- 0.0377) #estimators: 3
Boosting Accuracy: 0.90 (+/- 0.0354) #estimators: 4
Boosting Accuracy: 0.92 (+/- 0.0275) #estimators: 5
Boosting Accuracy: 0.92 (+/- 0.0354) #estimators: 10
Boosting Accuracy: 0.91 (+/- 0.0353) #estimators: 20
Boosting Accuracy: 0.91 (+/- 0.0225) #estimators: 40
```

As we used boosting,accuracy has increased from 0.88 to 0.91 and deviation reduced from 0.0281 to 0.0225.

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

In [105]:
```
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_es
t)
    boosting.fit(Data2_X, Data2_Y)
    ax = plt.subplot(2,4,count)
    fig = plot_decision_regions(X=Data2_X, y=Data2_Y, clf=boosting, le
gend=2,
                        scatter_kwargs=scatter_kwargs,
                        contourf_kwargs=contourf_kwargs,
                        scatter_highlight_kwargs=scatter_highlight_kwa
rgs)
    plt.title('Boosting with n_est:'+str(n_est))
plt.show()
```



Plots here ascertain the answer in 7d. As we increase the estimators, classification becomes accurate.We also see better boundaries.

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 [2]:
```
from sklearn import datasets
cancer = datasets.load_breast_cancer()
```

The features are:

In [3]:
```
cancer.feature_names
```

Out[3]:
```
array(['mean radius', 'mean texture', 'mean perimeter', 'mean area',
       'mean smoothness', 'mean compactness', 'mean concavity',
       'mean concave points', 'mean symmetry', 'mean fractal dimensi
on',
       'radius error', 'texture error', 'perimeter error', 'area err
or',
       '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 [4]:
```
cancer.target_names
```

Out[4]:
```
array(['malignant', 'benign'], dtype='<U9')
```

Create dataset for classification

In [5]:
```
X = cancer.data
Y = cancer.target
```

Number of samples are:

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

Out[7]:
```
(569, 30)
```

In [8]:
```
Y.shape
```

Out[8]:
```
(569,)
```

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 [6]:
```
#Linear SVM
for C in [0.001, 0.01, 0.1, 1] :
    svm_linear = SVC(C=C, kernel='linear')
    svm_linear_scores = cross_val_score(svm_linear, X, Y, cv=10, scori
ng='accuracy')
    print("C = " + str(C) + " Accuracy : " + str(svm_linear_scores.mea
n()))
```

```
C = 0.001 Accuracy : 0.9402460893613342
C = 0.01 Accuracy : 0.947235545760954
C = 0.1 Accuracy : 0.9472366260478783
C = 1 Accuracy : 0.9543179068360554
```

In [7]:
```
#Polynomial varying C and degree
for C in [0.001,0.01,0.1,1]:
    for d in [1,2] :
        svm_poly = SVC(C=C, kernel='poly',degree=d, gamma = 'auto')
        svm_poly_scores = cross_val_score(svm_poly, X, Y, cv=10, scori
ng='accuracy')
        print("C = " + str(C) + " Degree : " + str(d) + " Accuracy : "
+ str(svm_linear_scores.mean()))
```

```
C = 0.001 Degree : 1 Accuracy : 0.9543179068360554
C = 0.001 Degree : 2 Accuracy : 0.9543179068360554
C = 0.01 Degree : 1 Accuracy : 0.9543179068360554
C = 0.01 Degree : 2 Accuracy : 0.9543179068360554
C = 0.1 Degree : 1 Accuracy : 0.9543179068360554
C = 0.1 Degree : 2 Accuracy : 0.9543179068360554
C = 1 Degree : 1 Accuracy : 0.9543179068360554
C = 1 Degree : 2 Accuracy : 0.9543179068360554
```

In [9]:
```
#RBF varying C and gamma
for C in [0.001,0.01,0.1,1] :
    for g in [0.001, 0.01, 0.1, 1, 10, 100] :
        svm_rbf = SVC(C = C, kernel='rbf', gamma=g)
        svm_rbf_scores = cross_val_score(svm_rbf, X, Y, cv=10, scoring
='accuracy')
        print("C = " + str(C) + " Gamma : " + str(g) + " Accuracy : "
+ str(svm_rbf_scores.mean()))
```

```
C = 0.001 Gamma : 0.001 Accuracy : 0.6274274047186933
C = 0.001 Gamma : 0.01 Accuracy : 0.6274274047186933
C = 0.001 Gamma : 0.1 Accuracy : 0.6274274047186933
C = 0.001 Gamma : 1 Accuracy : 0.6274274047186933
C = 0.001 Gamma : 10 Accuracy : 0.6274274047186933
C = 0.001 Gamma : 100 Accuracy : 0.6274274047186933
C = 0.01 Gamma : 0.001 Accuracy : 0.6274274047186933
C = 0.01 Gamma : 0.01 Accuracy : 0.6274274047186933
C = 0.01 Gamma : 0.1 Accuracy : 0.6274274047186933
C = 0.01 Gamma : 1 Accuracy : 0.6274274047186933
C = 0.01 Gamma : 10 Accuracy : 0.6274274047186933
C = 0.01 Gamma : 100 Accuracy : 0.6274274047186933
C = 0.1 Gamma : 0.001 Accuracy : 0.6274274047186933
C = 0.1 Gamma : 0.01 Accuracy : 0.6274274047186933
C = 0.1 Gamma : 0.1 Accuracy : 0.6274274047186933
C = 0.1 Gamma : 1 Accuracy : 0.6274274047186933
C = 0.1 Gamma : 10 Accuracy : 0.6274274047186933
C = 0.1 Gamma : 100 Accuracy : 0.6274274047186933
C = 1 Gamma : 0.001 Accuracy : 0.9242416385792065
C = 1 Gamma : 0.01 Accuracy : 0.6326603145795523
C = 1 Gamma : 0.1 Accuracy : 0.6274274047186933
C = 1 Gamma : 1 Accuracy : 0.6274274047186933
C = 1 Gamma : 10 Accuracy : 0.6274274047186933
C = 1 Gamma : 100 Accuracy : 0.6274274047186933
```

Accuracy of SVM(linear) with C=1 is 95.4%.Polynomial SVM is 95.4% accurate . SVM(RB kernel)is 92.4% accurate

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

In [ ]:
```
for d in [2,4,6,8,10,50] :
    dt = DecisionTreeClassifier(max_depth=d)
    dt_scores = cross_val_score(dt, X, Y, cv=10, scoring='accuracy')
    print("Max_Depth = " + str(d) + " Accuracy : " + str(dt_scores.mea
n()))
```

Tree depth with 4,6 gives out highest accuracy.

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?

If the data is linearly seperable, I will use SVM. If data is non linear which can't be seperated, I will use SVM(rbf). But in general, there is no single best classifier over the space of all possibe problems. So i would see the data,process it and then see appropriate model that can be used on that data.