In [6]:
```
import matplotlib.pyplot as plt
from itertools import product
import numpy as np
from collections import Counter
from sklearn.base import BaseEstimator, RegressorMixin, ClassifierMixin
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor, export_gr
import graphviz

from IPython.display import Image

%matplotlib inline
```

Load Data

In [7]:
```
data_train = np.loadtxt('svm-train.txt')
data_test = np.loadtxt('svm-test.txt')
x_train, y_train = data_train[:, 0: 2], data_train[:, 2].reshape(-1, 1)
x_test, y_test = data_test[:, 0: 2], data_test[:, 2].reshape(-1, 1)
```

In [8]:
```
# Change target to 0-1 label
y_train_label = np.array(list(map(lambda x: 1 if x > 0 else 0, y_train))).reshape
```

Decision Tree Class

In [5]:
```
class Decision_Tree(BaseEstimator):

    def __init__(self, split_loss_function, leaf_value_estimator,
                 depth=0, min_sample=5, max_depth=10):
        '''
        Initialize the decision tree classifier

        :param split_loss_function: method for splitting node
        :param leaf_value_estimator: method for estimating leaf value
        :param depth: depth indicator, default value is 0, representing root node
        :param min_sample: an internal node can be splitted only if it contains p
        :param max_depth: restriction of tree depth.
        '''
        self.split_loss_function = split_loss_function
        self.leaf_value_estimator = leaf_value_estimator
        self.depth = depth
        self.min_sample = min_sample
        self.max_depth = max_depth

    def fit(self, X, y=None):
        '''
        This should fit the tree classifier by setting the values self.is_leaf,
        self.split_id (the index of the feature we want ot split on, if we're spl
        self.split_value (the corresponding value of that feature where the split
        and self.value, which is the prediction value if the tree is a leaf node.
        splitting the node, we should also init self.left and self.right to be De
        objects corresponding to the left and right subtrees. These subtrees shou
        the data that fall to the left and right,respectively, of self.split_valu
        This is a recurisive tree building procedure.

        :param X: a numpy array of training data, shape = (n, m)
        :param y: a numpy array of labels, shape = (n, 1)

        :return self
        '''
        if self.depth == self.max_depth:
            self.is_leaf = True


        # Your code goes here


        return self

    def predict_instance(self, instance):
        '''
        Predict label by decision tree

        :param instance: a numpy array with new data, shape (1, m)

        :return whatever is returned by leaf_value_estimator for leaf containing
        '''
        if self.is_leaf:
            return self.value
        if instance[self.split_id] <= self.split_value:
            return self.left.predict_instance(instance)
        else:
            return self.right.predict_instance(instance)
```

Decision Tree Classifier

In [ ]:
```
def compute_entropy(label_array):
    '''
    Calulate the entropy of given label list

    :param label_array: a numpy array of labels shape = (n, 1)
    :return entropy: entropy value
    '''
    # Your code goes here
    return entropy

def compute_gini(label_array):
    '''
    Calulate the gini index of label list

    :param label_array: a numpy array of labels shape = (n, 1)
    :return gini: gini index value
    '''
    # Your code goes here
    return gini
```

In [ ]:
```
def most_common_label(y):
    '''
    Find most common label
    '''
    label_cnt = Counter(y.reshape(len(y)))
    label = label_cnt.most_common(1)[0][0]
    return label
```

In [ ]:
```
class Classification_Tree(BaseEstimator, ClassifierMixin):

    loss_function_dict = {
        'entropy': compute_entropy,
        'gini': compute_gini
    }

    def __init__(self, loss_function='entropy', min_sample=5, max_depth=10):
        '''
        :param loss_function(str): loss function for splitting internal node
        '''

        self.tree = Decision_Tree(self.loss_function_dict[loss_function],
                                  most_common_label,
                                  0, min_sample, max_depth)

    def fit(self, X, y=None):
        self.tree.fit(X,y)
        return self

    def predict_instance(self, instance):
        value = self.tree.predict_instance(instance)
        return value
```

Decision Tree Boundary

In [ ]:
```
# Training classifiers with different depth
clf1 = Classification_Tree(max_depth=1)
clf1.fit(x_train, y_train_label)

clf2 = Classification_Tree(max_depth=2)
clf2.fit(x_train, y_train_label)

clf3 = Classification_Tree(max_depth=3)
clf3.fit(x_train, y_train_label)

clf4 = Classification_Tree(max_depth=4)
clf4.fit(x_train, y_train_label)

clf5 = Classification_Tree(max_depth=5)
clf5.fit(x_train, y_train_label)

clf6 = Classification_Tree(max_depth=6)
clf6.fit(x_train, y_train_label)

# Plotting decision regions
x_min, x_max = x_train[:, 0].min() - 1, x_train[:, 0].max() + 1
y_min, y_max = x_train[:, 1].min() - 1, x_train[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.1),
                     np.arange(y_min, y_max, 0.1))

f, axarr = plt.subplots(2, 3, sharex='col', sharey='row', figsize=(10, 8))

for idx, clf, tt in zip(product([0, 1], [0, 1, 2]),
                        [clf1, clf2, clf3, clf4, clf5, clf6],
                        ['Depth = {}'.format(n) for n in range(1, 7)]):

    Z = np.array([clf.predict_instance(x) for x in np.c_[xx.ravel(), yy.ravel()]]
    Z = Z.reshape(xx.shape)

    axarr[idx[0], idx[1]].contourf(xx, yy, Z, alpha=0.4)
    axarr[idx[0], idx[1]].scatter(x_train[:, 0], x_train[:, 1], c=y_train_label,
    axarr[idx[0], idx[1]].set_title(tt)

plt.show()
```

Compare decision tree with tree model in sklearn

In [ ]:
```
clf = DecisionTreeClassifier(criterion='entropy', max_depth=10, min_samples_split
clf.fit(x_train, y_train_label)
export_graphviz(clf, out_file='tree_classifier.dot')
```

In [ ]:
```
# Visualize decision tree
!dot -Tpng tree_classifier.dot -o tree_classifier.png
```

In [ ]:
```
Image(filename='tree_classifier.png')
```

Decision Tree Regressor

In [ ]:
```python
# Regression Tree Specific Code
def mean_absolute_deviation_around_median(y):
    '''
    Calulate the mean absolute deviation around the median of a given target list

    :param y: a numpy array of targets shape = (n, 1)
    :return mae
    '''
    # Your code goes here
    return mae
```

In [ ]:
```python
class Regression_Tree():
    '''
    :attribute loss_function_dict: dictionary containing the loss functions used
    :attribute estimator_dict: dictionary containing the estimation functions use
    '''

    loss_function_dict = {
        'mse': np.var,
        'mae': mean_absolute_deviation_around_median
    }

    estimator_dict = {
        'mean': np.mean,
        'median': np.median
    }

    def __init__(self, loss_function='mse', estimator='mean', min_sample=5, max_d
        '''
        Initialize Regression_Tree
        :param loss_function(str): loss function used for splitting internal node
        :param estimator(str): value estimator of internal node
        '''

        self.tree = Decision_Tree(self.loss_function_dict[loss_function],
                                  self.estimator_dict[estimator],
                                  0, min_sample, max_depth)

    def fit(self, X, y=None):
        self.tree.fit(X,y)
        return self

    def predict_instance(self, instance):
        value = self.tree.predict_instance(instance)
        return value
```

Fit regression tree to one-dimensional regression data

In [21]:
```
data_krr_train = np.loadtxt('krr-train.txt')
data_krr_test = np.loadtxt('krr-test.txt')
x_krr_train, y_krr_train = data_krr_train[:,0].reshape(-1,1),data_krr_train[:,1].
x_krr_test, y_krr_test = data_krr_test[:,0].reshape(-1,1),data_krr_test[:,1].resh

# Training regression trees with different depth
clf1 = Regression_Tree(max_depth=1,  min_sample=1, loss_function='mae', estimator
clf1.fit(x_krr_train, y_krr_train)

clf2 = Regression_Tree(max_depth=2,  min_sample=1, loss_function='mae', estimator
clf2.fit(x_krr_train, y_krr_train)

clf3 = Regression_Tree(max_depth=3,  min_sample=1, loss_function='mae', estimator
clf3.fit(x_krr_train, y_krr_train)

clf4 = Regression_Tree(max_depth=4,  min_sample=1, loss_function='mae', estimator
clf4.fit(x_krr_train, y_krr_train)

clf5 = Regression_Tree(max_depth=5,  min_sample=1, loss_function='mae', estimator
clf5.fit(x_krr_train, y_krr_train)

clf6 = Regression_Tree(max_depth=6,  min_sample=1, loss_function='mae', estimator
clf6.fit(x_krr_train, y_krr_train)

plot_size = 0.001
x_range = np.arange(0., 1., plot_size).reshape(-1, 1)

f2, axarr2 = plt.subplots(2, 3, sharex='col', sharey='row', figsize=(15, 10))

for idx, clf, tt in zip(product([0, 1], [0, 1, 2]),
                        [clf1, clf2, clf3, clf4, clf5, clf6],
                        ['Depth = {}'.format(n) for n in range(1, 7)]):

    y_range_predict = np.array([clf.predict_instance(x) for x in x_range]).reshap

    axarr2[idx[0], idx[1]].plot(x_range, y_range_predict, color='r')
    axarr2[idx[0], idx[1]].scatter(x_krr_train, y_krr_train, alpha=0.8)
    axarr2[idx[0], idx[1]].set_title(tt)
    axarr2[idx[0], idx[1]].set_xlim(0, 1)
plt.show()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-21-d534932f835c> in <module>
      5
      6 # Training regression trees with different depth
----> 7 clf1 = Regression_Tree(max_depth=1,  min_sample=1, loss_function='mae',
estimator='median')
      8 clf1.fit(x_krr_train, y_krr_train)
      9

NameError: name 'Regression_Tree' is not defined
```

8 Gradient Boosting Implementations

8.1 Gradient Boosting Method

```
In [13]: #Pseudo-residual function.
         #Here you can assume that we are using L2 loss

         def pseudo_residual_L2(train_target, train_predict):
             '''
             Compute the pseudo-residual based on current predicted value.
             '''
             return train_target - train_predict
```

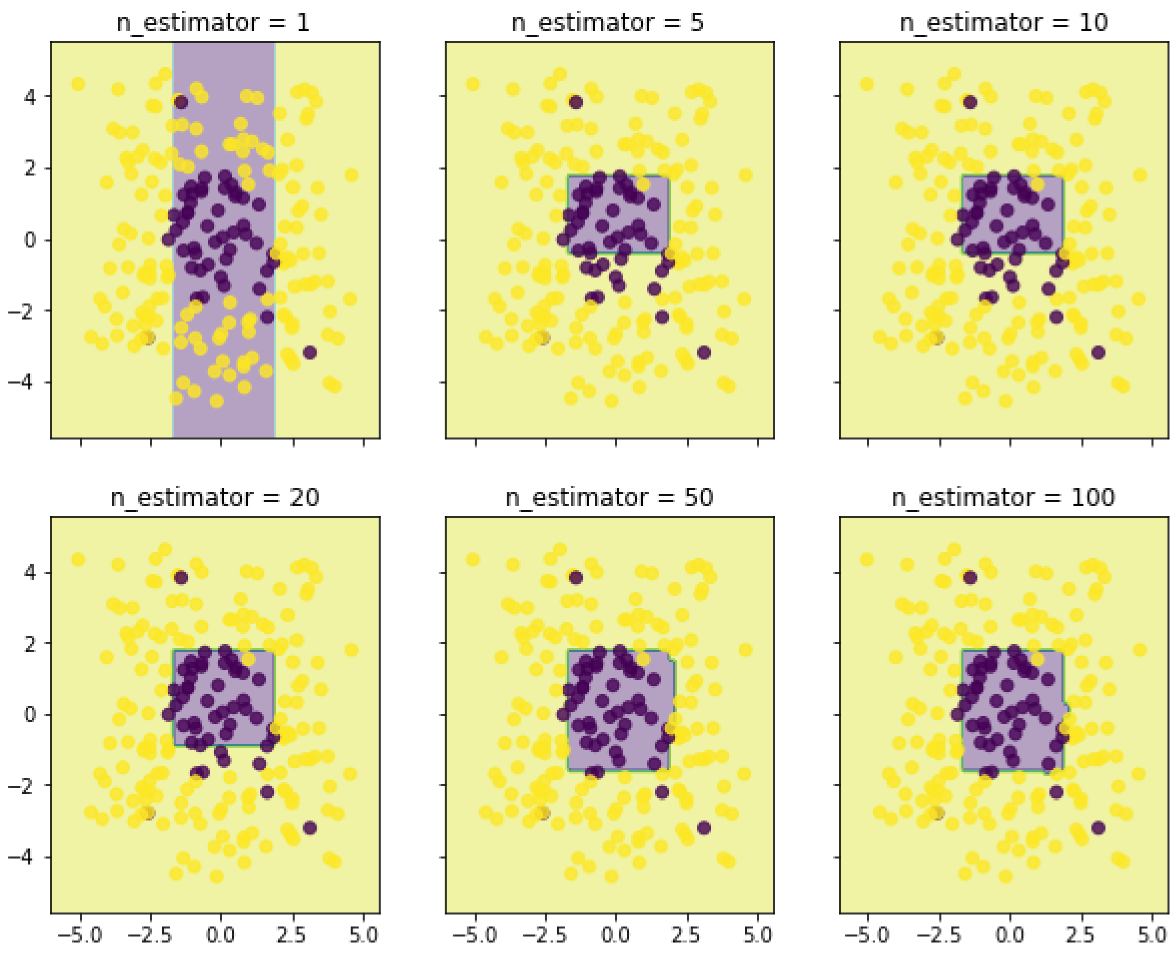
In [50]:
```
class gradient_boosting():
    '''
    Gradient Boosting regressor class
    :method fit: fitting model
    '''
    def __init__(self, n_estimator, pseudo_residual_func, learning_rate=0.1, min_
        '''
        Initialize gradient boosting class

        :param n_estimator: number of estimators (i.e. number of rounds of gradie
        :pseudo_residual_func: function used for computing pseudo-residual
        :param learning_rate: step size of gradient descent
        '''
        self.n_estimator = n_estimator
        self.pseudo_residual_func = pseudo_residual_func
        self.learning_rate = learning_rate
        self.min_sample = min_sample
        self.max_depth = max_depth
        #create param to save base models
        self.base_models = []
        self.h_0 = DecisionTreeRegressor(max_depth=self.max_depth,min_samples_lea

    def fit(self, train_data, train_target):
        '''
        Fit gradient boosting model
        '''
        # Your code goes here
        #NOTE: HERE I START FROM m=0, USE "n_estimator" as upper bound M(m=0,1,..
        #initialize first prediction f0(x) as 0
        train_prediction = 0

        #for step m 1 to M:
        for step in range(1,self.n_estimator+1):
            #calculate (-gm)
            residual = self.pseudo_residual_func(train_target.reshape(-1),train_p
            #fit regression to residual(-gm)
            hm = DecisionTreeRegressor(max_depth=self.max_depth,min_samples_leaf=
            hm.fit(train_data,residual)
            #save hm to base_models
            self.base_models.append(hm)
            #compute fm(x)
            train_prediction += self.learning_rate * hm.predict(train_data)

        return self


    def predict(self, test_data):
        '''
        Predict value
        '''
        # Your code goes here
        prediction = 0

        for i in range(len(self.base_models)):
            prediction += self.learning_rate * self.base_models[i].predict(test_d
        return prediction
```

In [51]:
```
y_train_label.shape
```

Out[51]:
```
(200, 1)
```

2-D GBM visualization - SVM data

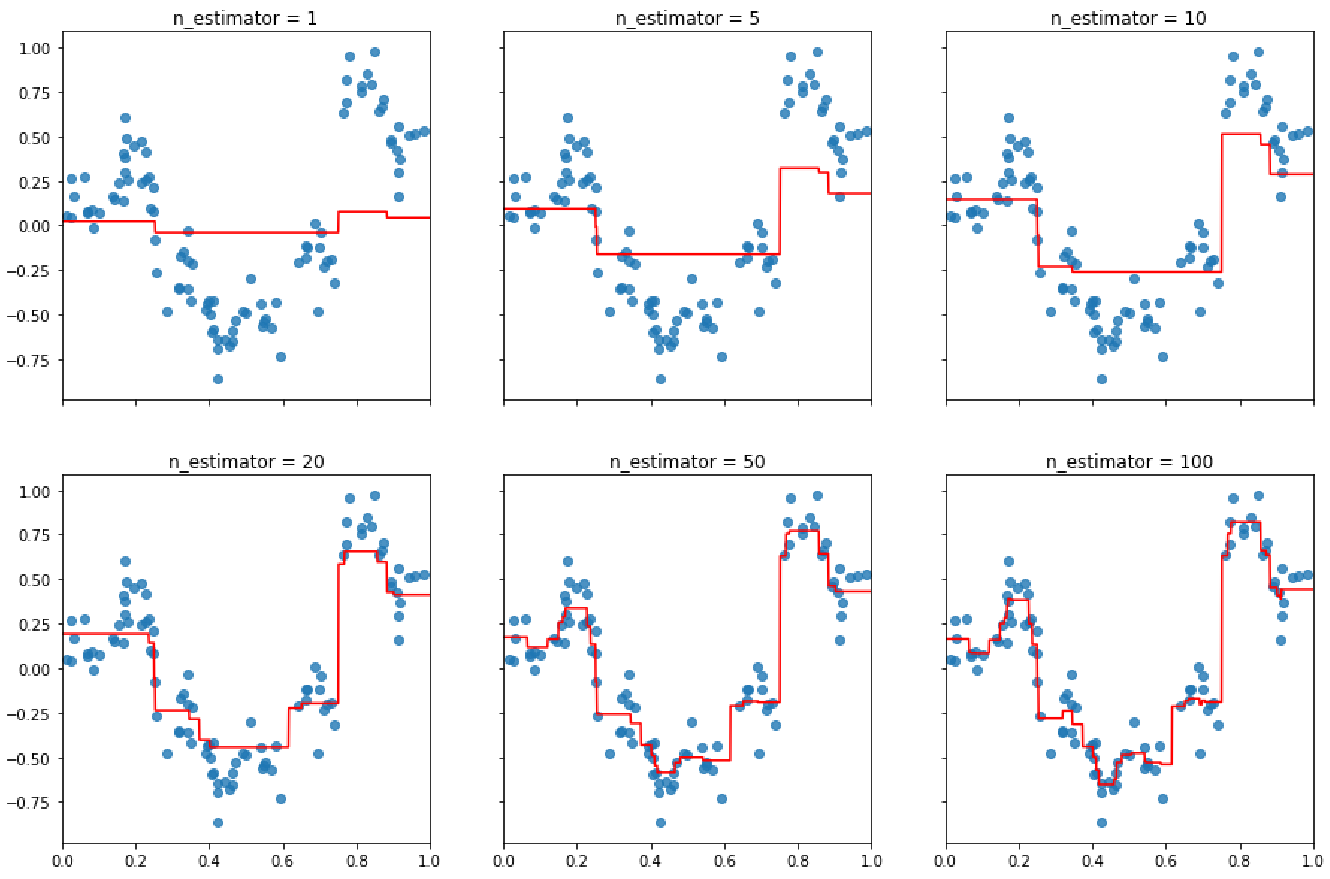
In [52]:
```python
# Plotting decision regions
x_min, x_max = x_train[:, 0].min() - 1, x_train[:, 0].max() + 1
y_min, y_max = x_train[:, 1].min() - 1, x_train[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.1),
                     np.arange(y_min, y_max, 0.1))

f, axarr = plt.subplots(2, 3, sharex='col', sharey='row', figsize=(10, 8))

for idx, i, tt in zip(product([0, 1], [0, 1, 2]),
                      [1, 5, 10, 20, 50, 100],
                      ['n_estimator = {}'.format(n) for n in [1, 5, 10, 20, 50,

    gbt = gradient_boosting(n_estimator=i, pseudo_residual_func=pseudo_residual_L
    gbt.fit(x_train, y_train)

    Z = np.sign(gbt.predict(np.c_[xx.ravel(), yy.ravel()]))
    Z = Z.reshape(xx.shape)

    axarr[idx[0], idx[1]].contourf(xx, yy, Z, alpha=0.4)
    axarr[idx[0], idx[1]].scatter(x_train[:, 0], x_train[:, 1], c=y_train_label.r
    axarr[idx[0], idx[1]].set_title(tt)
```



1-D GBM visualization - KRR data

In [53]:
```python
data_krr_train = np.loadtxt('krr-train.txt')
data_krr_test = np.loadtxt('krr-test.txt')
x_krr_train, y_krr_train = data_krr_train[:,0].reshape(-1,1),data_krr_train[:,1].
x_krr_test, y_krr_test = data_krr_test[:,0].reshape(-1,1),data_krr_test[:,1].resh
```

In [54]:
```python
plot_size = 0.001
x_range = np.arange(0., 1., plot_size).reshape(-1, 1)

f2, axarr2 = plt.subplots(2, 3, sharex='col', sharey='row', figsize=(15, 10))

for idx, i, tt in zip(product([0, 1], [0, 1, 2]),
                      [1, 5, 10, 20, 50, 100],
                      ['n_estimator = {}'.format(n) for n in [1, 5, 10, 20, 50,

    gbm_1d = gradient_boosting(n_estimator=i, pseudo_residual_func=pseudo_residua
    gbm_1d.fit(x_krr_train, y_krr_train)

    y_range_predict = gbm_1d.predict(x_range)

    axarr2[idx[0], idx[1]].plot(x_range, y_range_predict, color='r')
    axarr2[idx[0], idx[1]].scatter(x_krr_train, y_krr_train, alpha=0.8)
    axarr2[idx[0], idx[1]].set_title(tt)
    axarr2[idx[0], idx[1]].set_xlim(0, 1)
```

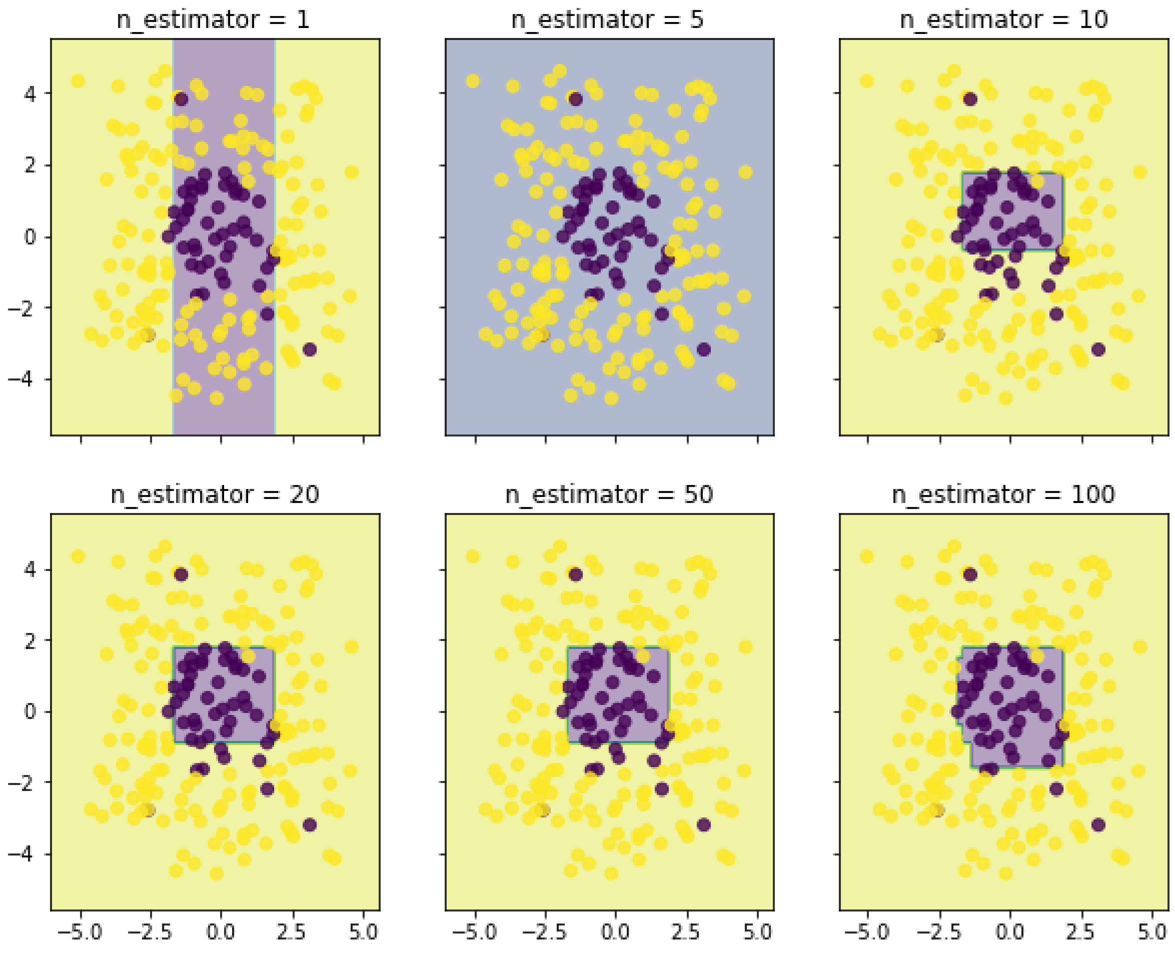


8.2 Logistic loss

In [55]:
```python
#Pseudo-residual function.
#Here you can assume that we are using Logistic loss

def pseudo_residual_logistic(train_target,train_predict):
    y = train_target
    f = train_predict
    m = y*f
    return (y*np.exp(-m))/(1+np.exp(-m))
```

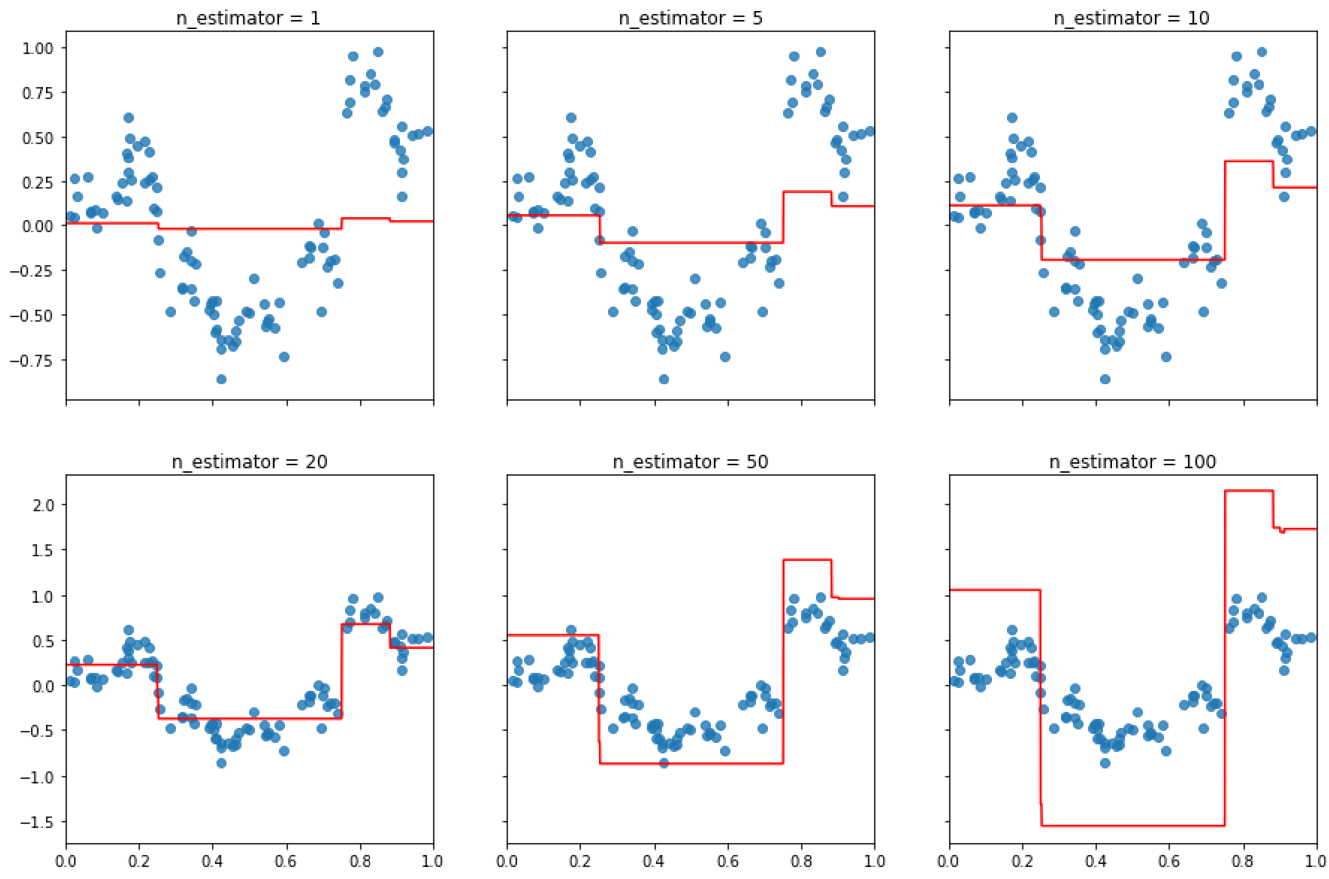
2-D GBM visualization - SVM data

In [61]:
```
# Plotting decision regions
x_min, x_max = x_train[:, 0].min() - 1, x_train[:, 0].max() + 1
y_min, y_max = x_train[:, 1].min() - 1, x_train[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.1),
                     np.arange(y_min, y_max, 0.1))

f, axarr = plt.subplots(2, 3, sharex='col', sharey='row', figsize=(10, 8))

for idx, i, tt in zip(product([0, 1], [0, 1, 2]),
                       [1, 5, 10, 20, 50, 100],
                       ['n_estimator = {}'.format(n) for n in [1, 5, 10, 20, 50,

    gbt = gradient_boosting(n_estimator=i, pseudo_residual_func=pseudo_residual_l
    gbt.fit(x_train, y_train)

    Z = np.sign(gbt.predict(np.c_[xx.ravel(), yy.ravel()]))
    Z = Z.reshape(xx.shape)

    axarr[idx[0], idx[1]].contourf(xx, yy, Z, alpha=0.4)
    axarr[idx[0], idx[1]].scatter(x_train[:, 0], x_train[:, 1], c=y_train_label.r
    axarr[idx[0], idx[1]].set_title(tt)
```



1-D GBM visualization - KRR data

In [57]:
```
plot_size = 0.001
x_range = np.arange(0., 1., plot_size).reshape(-1, 1)

f2, axarr2 = plt.subplots(2, 3, sharex='col', sharey='row', figsize=(15, 10))

for idx, i, tt in zip(product([0, 1], [0, 1, 2]),
                      [1, 5, 10, 20, 50, 100],
                      ['n_estimator = {}'.format(n) for n in [1, 5, 10, 20, 50,

    gbm_1d = gradient_boosting(n_estimator=i, pseudo_residual_func=pseudo_residua
    gbm_1d.fit(x_krr_train, y_krr_train)

    y_range_predict = gbm_1d.predict(x_range)

    axarr2[idx[0], idx[1]].plot(x_range, y_range_predict, color='r')
    axarr2[idx[0], idx[1]].scatter(x_krr_train, y_krr_train, alpha=0.8)
    axarr2[idx[0], idx[1]].set_title(tt)
    axarr2[idx[0], idx[1]].set_xlim(0, 1)
```



In [ ]: