In [10]:
```
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_graphviz
#import graphviz

from IPython.display import Image

%matplotlib inline
```

Load Data

In [11]:
```
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 [12]:
```
# Change target to 0-1 label
y_train_label = np.array(list(map(lambda x: 1 if x > 0 else 0,
y_train))).reshape(-1, 1)
```

Decision Tree Class

In [13]:
```
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 c
ontains points more than min_smaple
        :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
        self.split_id = 0
        self.split_value = 0
```

```
    def fit(self, X, y=None):
        '''
        This should fit the tree classifier by setting the values self.i
s_leaf,
        self.split_id (the index of the feature we want ot split on, if
 we're splitting),
        self.split_value (the corresponding value of that feature where
 the split is),
        and self.value, which is the prediction value if the tree is a l
eaf node.  If we are
        splitting the node, we should also init self.left and self.right
 to be Decision_Tree
        objects corresponding to the left and right subtrees. These subt
rees should be fit on
        the data that fall to the left and right,respectively, of self.s
plit_value.
        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
        '''

        # Your code goes here
        m = X.shape[1]
        n = X.shape[0]

        if n<=self.min_sample or self.depth==self.max_depth:
            self.is_leaf=True
            self.value=self.leaf_value_estimator(y)
        else: # not a leaf node
            # for each feature
            self.is_leaf=False
            min_loss =1e7
            best_idx = None
            split_row=None
            for i in range(m):
                # sort the matrix based on features
                idx = np.argsort(X[:,i])
                X_sorted = X[idx]
                y_sorted = y[idx]
                for split in range(n-1): # for each split

                    loss =
(split+1)*self.split_loss_function(y_sorted[:split+1]) + (n-
split)*self.split_loss_function(y_sorted[split+1:])

                    if(loss<min_loss):
                        min_loss = loss
                        self.split_id = i
                        self.split_value = (X_sorted[split,i] +
X_sorted[split+1,i])/2
```

```
                        split_row = split
                        best_idx = idx

            if not split_row is None:
                self.left = Decision_Tree(self.split_loss_function,
self.leaf_value_estimator,self.depth+1, self.min_sample, self.max_depth)
                self.right = Decision_Tree(self.split_loss_function, sel
f.leaf_value_estimator,self.depth+1, self.min_sample, self.max_depth)
                X_sorted = X[best_idx]
                y_sorted = y[best_idx]
                self.left.fit(X_sorted[:self.split_row+1], y_sorted[:sel
f.split_row+1])
                self.right.fit(X_sorted[self.split_row+1:], y_sorted[sel
f.split_row+1:])


        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 co
ntaining instance
        '''
        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 [15]:
```
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
    # Your code goes here
    classes, counts = np.unique(label_array, return_counts=True)
    s = np.size(label_array)

    p = counts/s
    entropy = -np.dot(p, np.log2(p))
    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
    classes, count = np.unique(label_array, return_counts=True)
    s = np.size(label_array)
    gini = np.dot(count/s, 1-(count/s))

    return gini
```

In [16]:
```
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 [17]:
```
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 [19]:
```
# 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_trai
n_label, alpha=0.8)
    axarr[idx[0], idx[1]].set_title(tt)

plt.show()
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-19-d2d031b178b5> in <module>()
      1 # Training classifiers with different depth
      2 clf1 = Classification_Tree(max_depth=1)
----> 3 clf1.fit(x_train, y_train_label)
      4 
      5 clf2 = Classification_Tree(max_depth=2)

<ipython-input-17-19774630efe6> in fit(self, X, y)
     16 
     17     def fit(self, X, y=None):
---> 18         self.tree.fit(X,y)
     19         return self
     20 

<ipython-input-13-169d11af26af> in fit(self, X, y)
     74                 X_sorted = X[best_idx]
     75                 y_sorted = y[best_idx]
---> 76                 self.left.fit(X_sorted[:self.split_row+1], y_sorted[:self.split_row+1])
     77                 self.right.fit(X_sorted[self.split_row+1:], y_sorted[self.split_row+1:])
     78 

AttributeError: 'Decision_Tree' object has no attribute 'split_row'
```

Compare decision tree with tree model in sklearn

In [20]:

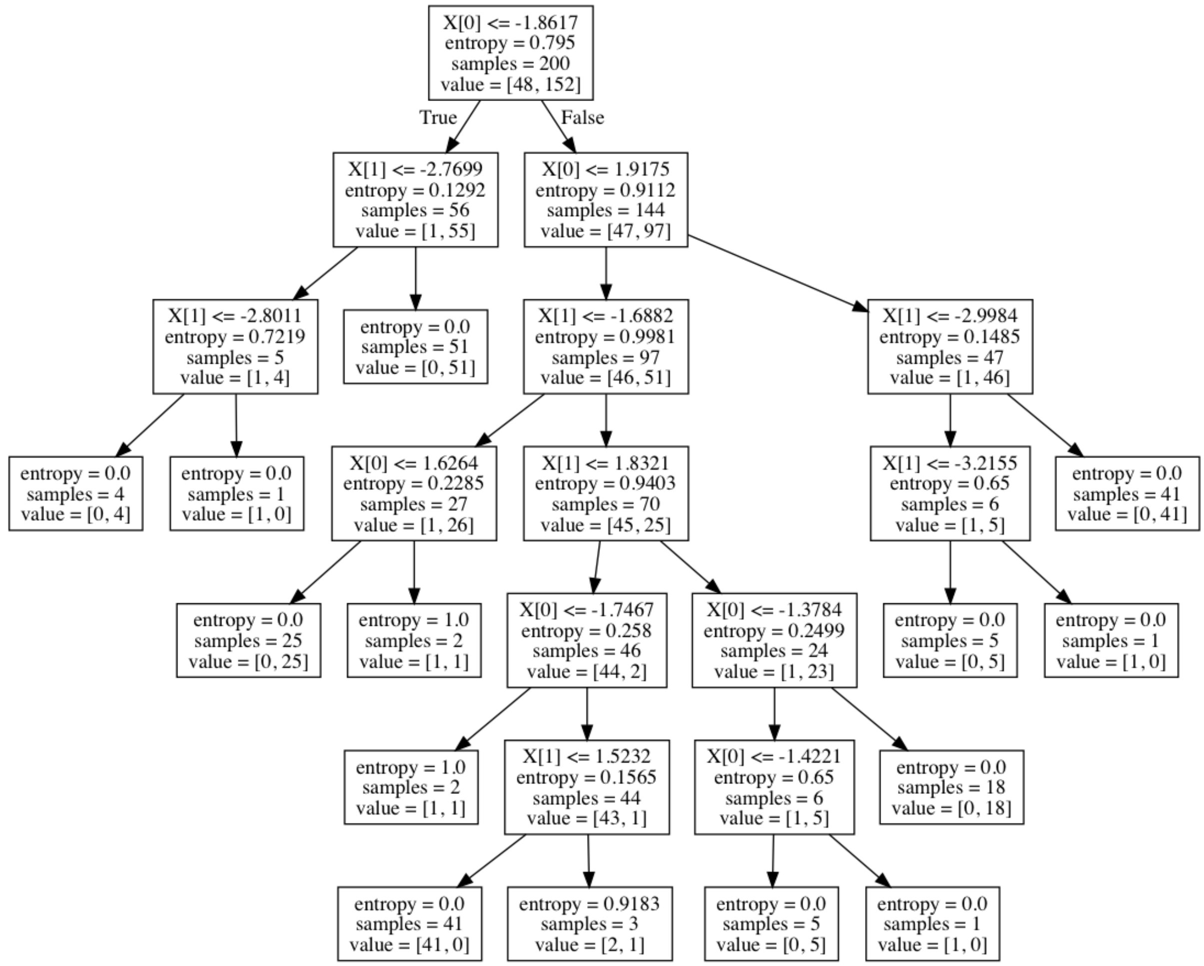
```
clf = DecisionTreeClassifier(criterion='entropy', max_depth=10, min_samples_split=5)
clf.fit(x_train, y_train_label)
export_graphviz(clf, out_file='tree_classifier.dot')
```

In [21]:

```
# Visualize decision tree
!dot -Tpng tree_classifier.dot -o tree_classifier.png
```

In [22]:
```
Image(filename='tree_classifier.png')
```

Out[22]:



Decision Tree Regressor

In [23]:
```
# Regression Tree Specific Code
def mean_absolute_deviation_around_median(y):
    '''
    Calulate the mean absolute deviation around the median of a given ta
rget list

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

In [24]:
```
class Regression_Tree():
    '''
    :attribute loss_function_dict: dictionary containing the loss functi
ons used for splitting
    :attribute estimator_dict: dictionary containing the estimation func
tions used in leaf nodes
    '''

    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_depth=10):
        '''
        Initialize Regression_Tree
        :param loss_function(str): loss function used for splitting inte
rnal nodes
        :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 [25]:
```
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_tr
ain[:,1].reshape(-1,1)
x_krr_test, y_krr_test =
data_krr_test[:,0].reshape(-1,1),data_krr_test[:,1].reshape(-1,1)

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

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

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

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

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

clf6 = Regression_Tree(max_depth=6,  min_sample=1, loss_function='mae',
estimator='median')
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]).reshape(-1, 1)

    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-25-fb0845488eb1> in <module>()
      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 
     10 clf2 = Regression_Tree(max_depth=2,  min_sample=1, loss_function='mae', estimator='median')

<ipython-input-24-d2bb9fbe5cf0> in fit(self, X, y)
     27 
     28     def fit(self, X, y=None):
---> 29         self.tree.fit(X,y)
     30         return self
     31 

<ipython-input-13-169d11af26af> in fit(self, X, y)
     60                 for split in range(n-1): # for each split
     61 
---> 62                     loss = (split+1)*self.split_loss_function(y_sorted[:split+1]) + (n-split)*self.split_loss_function(y_sorted[split+1:])
     63 
     64                     if(loss<min_loss):

<ipython-input-23-603c6fd699e9> in mean_absolute_deviation_around_median(y)
      8     '''
      9     # Your code goes here
---> 10     return mae

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

Gradient Boosting Method

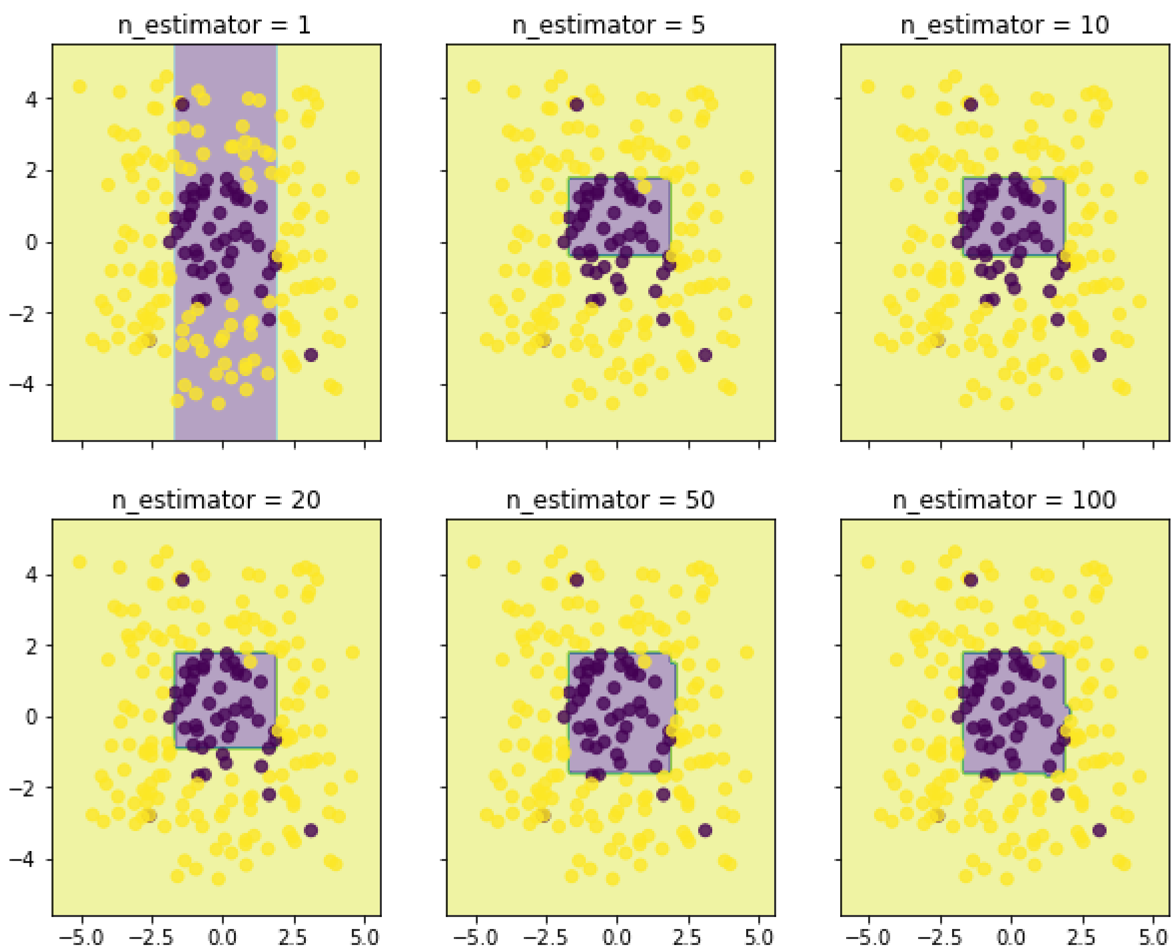
In [26]:

```
#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 [27]:
```
class gradient_boosting():
    '''
    Gradient Boosting regressor class
    :method fit: fitting model
    '''
    def __init__(self, n_estimator, pseudo_residual_func,
learning_rate=0.1, min_sample=5, max_depth=3):
        '''
        Initialize gradient boosting class

        :param n_estimator: number of estimators (i.e. number of rounds
 of gradient boosting) #M
        :pseudo_residual_func: function used for computing pseudo-residu
al
        :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
        self.basis_functions = list()

    def fit(self, train_data, train_target):
        '''
        Fit gradient boosting model
        '''
        prediction_vector = np.zeros(train_target.shape[0])
        for step in range(self.n_estimator): # 1 to M
            # Compute residual
            residual =
self.pseudo_residual_func(train_target.reshape(-1), prediction_vector)
            # Fit regression Model
            h_m = DecisionTreeRegressor(max_depth=self.max_depth, min_sa
mples_leaf=self.min_sample)
            h_m.fit(train_data,residual)
            # Update prediction_vector for next step
            prediction_vector= prediction_vector + self.learning_rate*h_
m.predict(train_data)
            # append to the list of basis functions
            self.basis_functions.append(h_m)

    def predict(self, test_data):
        '''
        Predict value
        '''
        prediction_vector = np.zeros(test_data.shape[0])
        for i in range(len(self.basis_functions)):
            #Get fm(x) by summation of base predictions
            prediction_vector += self.learning_rate*\
            self.basis_functions[i].predict(test_data)
        return prediction_vector
```

2-D GBM visualization - SVM data

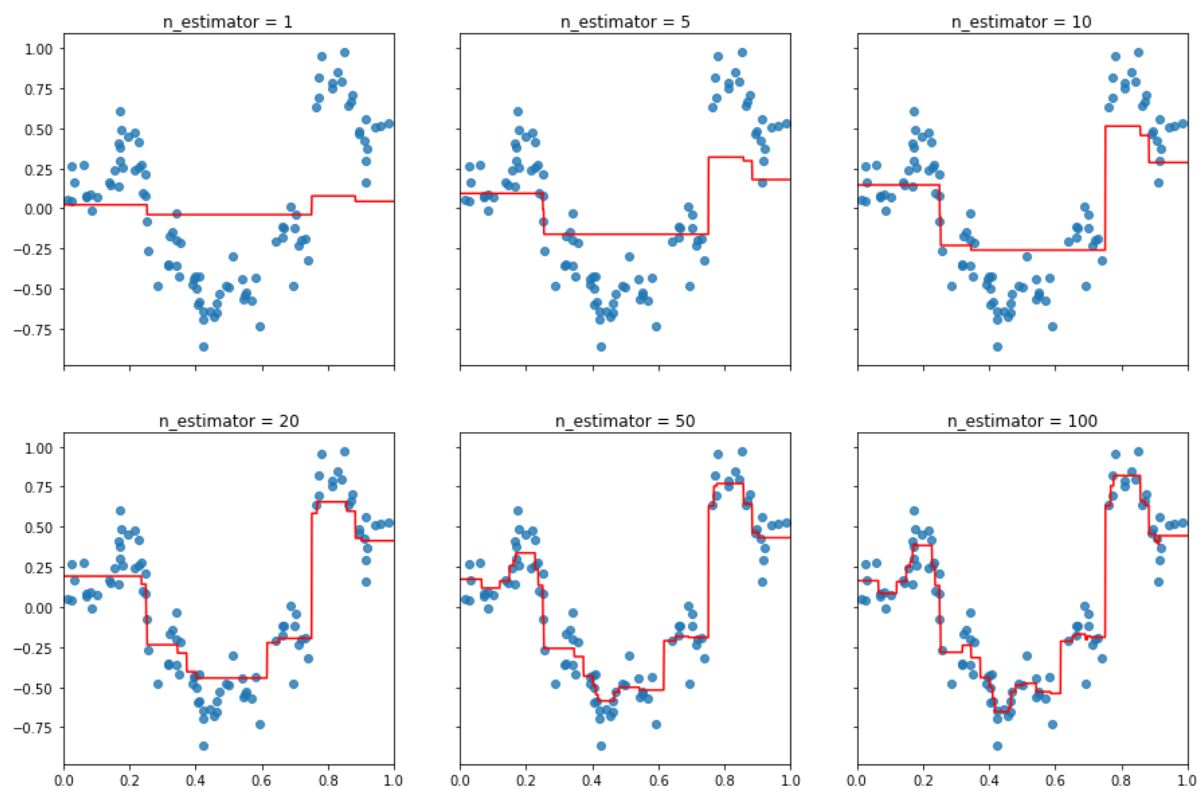
In [28]:
```
# Plotting decision regions
x_min, x_max = x_train[:, 0].min() - 1, x_train[:, 0].max() + 1
y_min, y_max = x_train[:, 1].min() - 1, x_train[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.1),
                     np.arange(y_min, y_max, 0.1))

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, 100]]):

    gbt = gradient_boosting(n_estimator=i, pseudo_residual_func=pseudo_r
esidual_L2, max_depth=2)
    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_trai
n_label, alpha=0.8)
    axarr[idx[0], idx[1]].set_title(tt)
```



1-D GBM visualization - KRR data

In [29]:
```
# Load Data
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_tr
ain[:,1].reshape(-1,1)
x_krr_test, y_krr_test =
data_krr_test[:,0].reshape(-1,1),data_krr_test[:,1].reshape(-1,1)
```

In [30]:
```
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, 100]]):

    gbm_1d = gradient_boosting(n_estimator=i, pseudo_residual_func=pseud
o_residual_L2, max_depth=2)
    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 [32]:
```
def logistic_pseudo_residual(train_target, train_predict):
    return train_target/(1+np.exp(train_target*train_predict))
```

1.1) $l(\hat{y}, y) = \frac{1}{2}(\hat{y}-y)^2$

$$\Rightarrow \partial_{\hat{y}} l(\hat{y}, y) = \partial_{\hat{y}} (\hat{y}-y)^2/2 = -(y-a).$$

$$\Rightarrow -\partial_2 l(y_i, f_{m-1}(x_i)) = (y_i - f_{m-1}(x_i))$$

$$\Rightarrow -g_m = \sum_{i=1}^{n} (y_i - f_{m-1}(x_i)).$$

$$\Rightarrow \boxed{h_m = \underset{h \in H}{\operatorname{argmin}} \sum_{1}^{n} [(y_i - f_{m-1}(x_i)) - h(x_i)]^2}$$

1.2) $l = \ln(1 + e^{-y\hat{y}})$

$$\Rightarrow -g_m = -\left[\frac{d}{d f_{m-1}(x_j)} (\ln(1 + e^{-y_j \hat{y}}))\right]_{j=1}^{n}$$

$$= \frac{y_1 \cdot e^{-y_1 \hat{y}}}{1 + e^{-y_1 f_{m-1}(x_1)}} + \cdots \frac{y_n \cdot e^{-y_n \hat{y}}}{1 + e^{-y_n f_{m-1}(x_n)}}$$

$$\cancel{\Rightarrow h_m = \left[\frac{y_i e^{-m}}{1 + e^{-m}} * - h(x_i)\right]^2}$$

$$\boxed{h_m = \underset{h \in H}{\operatorname{argmin}} \sum_{i=1}^{n} \left[\frac{y_i e^{-m}}{1 + e^{-m}} - h(x_i)\right]^2}$$

$$= \underset{h \in H}{\operatorname{argmin}} \sum_{i=1}^{n} \left[\frac{y_i}{1 + e^{m}} - h(x_i)\right]^2$$

2.1) $E_y\left[\,l(yf(x))\,|\,x\,\right]$

$$= l(f(x))\cdot\pi(x) + l(-f(x))\,(1-\pi(x))$$

2.2) For Exponential Loss:

$$E_y\left[e^{-yf(x)}|x\right] = e^{-f(x)}\,\cancel{\pi(x)}\,\pi(x) + e^{f(x)}\cdot(1-\pi(x)).$$

$$= \pi(x)\left(e^{-f(x)} - e^{f(x)}\right) + e^{f(x)}$$

Baye's Prediction Function:

$$\underset{\hat{y}}{\operatorname{argmin}}\ \pi(x)\left(e^{-\hat{y}} - e^{\hat{y}}\right) + e^{\hat{y}}$$

$$\frac{d}{d\hat{y}} E_y' = \pi(x)\left(-e^{-\hat{y}} - e^{\hat{y}}\right) + e^{\hat{y}}$$

$$= 0$$

$$\Rightarrow \boxed{\hat{y}_* = \frac{1}{2}\ln\left(\frac{\pi(x)}{1-\pi(x)}\right)}$$

$$\Rightarrow \boxed{f^*(x) = \frac{1}{2}\ln\left(\frac{\pi(x)}{1-\pi(x)}\right)}$$

2.3) $f^*(x) = \underset{\hat{y}}{\operatorname{argmin}}\ \pi(x)\ln\left(1+e^{-\hat{y}}\right) + (1-\pi(x))\left(\ln\left(1+e^{\hat{y}}\right)\right)$

$$\Rightarrow \partial_{\hat{y}}\left(\pi(x)\ln\left(1+e^{-\hat{y}}\right) + (1-\pi(x))\ln\left(1+e^{\hat{y}}\right)\right) = 0$$

$$\Rightarrow e^{\hat{y}} = \frac{\pi(x)}{1-\pi(x)}$$

$$\Rightarrow \hat{y} = \ln\left(\frac{p}{1-p}\right) = \cancel{\ln\left(\frac{1-\pi(x)\ \pi(x)}{1-\pi(x)}\right)}$$

$$\boxed{\hat{y} = \ln\left(\frac{\pi(x)}{1-\pi(x)}\right)}$$

(3.1)

Case 1

$y = g(x) \Rightarrow$ LHS $= 0$

RHS $= \exp(+ve) > 0$ (LHS)

Case 2

$y \neq g(x) \Rightarrow$ LHS $= 1$

RHS $= \cancel{\exp(-y g(x))}$

$\exp(-y g(x)) > 1$

RHS $>$ LHS

$$\Rightarrow \boxed{1_{\{g(x) \neq y\}} < \exp(-y\, g(x))}$$

3.2) ~~Case~~

$$L(G, D) = \frac{1}{n} \sum_{1}^{n} 1_{(G(x_i) \neq y_i)}$$

$$Z_T = \frac{1}{n} \sum_{1}^{m} \exp(-y_i f_T(x_i))$$

$$\Rightarrow 1_{(G(x_i) \neq y_i)} < \exp(-y_i f_T(x_i))$$

$$\Rightarrow \sum_{1}^{m} 1_{(G(x_i) \neq y_i)} < \sum_{1}^{m} \exp(-y_i f_T(x_i))$$

$$\Rightarrow L(G, D) < Z_T$$

$$\Rightarrow \boxed{\pi(x) = \frac{1}{1 + e^{-f^*(x)}}}$$

and

$$\boxed{f^*(x) = \ln\left(\frac{\pi(x)}{1-\pi(x)}\right)}$$

(4.) Hinge Loss :

$$E_y\left((1 - yf(x))_+ \middle| x\right) = \pi(x)\left(1 - f(x)\right)_+ + (1-\pi(x))(1+f(x))_+$$

$$= \cancel{\pi(x)} - \pi(x)f(x) + 1 + f(x) - \cancel{\pi(x)} - \pi(x)f(x)$$

$$= 1 + f(x) - 2\pi(x)f(x) \quad = 0$$

$$E_y = 1 + f(x)(1 - 2\pi(x)) \quad \rightarrow \hat{y} \in [-1, 1]$$

$$\Rightarrow f^*(x) = - \text{sign}(1 - 2\pi(x))$$

3.5)

$g(a) = a(1-a)$

$g'(a) = 1-2a \geq 0$ for $a \in [0, \frac{1}{2}]$

$p = (1-a) - \exp(-a)$

$\frac{dp}{da} = -1 + \exp(-a) \leq 0 \quad \forall a \in [0, \frac{1}{2}]$

so at 0 $1-a = \exp(-a)$

so unorasing function $\Rightarrow$ $\boxed{\exp(-a) \geq 1-a \quad \forall a \geq 0}$

$$\frac{Z_{t+1}}{Z_t} = 2\sqrt{err_{t+1}(1-err_{t+1})}$$

$$\leq 2\sqrt{\left(\frac{1}{2}-\gamma\right)\left(\frac{1}{2}+\gamma\right)}$$

$$\leq 2\sqrt{\frac{1}{4}-\gamma^2}$$

$$\leq 2\sqrt{1-4\gamma^2}$$

$$\leq \exp(-2\gamma^2)$$

3.5 $L(G,D) < Z_T$

$$\frac{Z_T}{Z_{T-1}} \leq \exp(-2\gamma^2)$$

$$Z_T \leq Z_{T-1} \exp(-2\gamma^2)$$

So $Z_T \leq \exp(-2\gamma^2)$

$\Rightarrow$ $Z_T \leq \exp(-2\gamma^2 T)$

So $L(G,D) < Z_T$

$$L(G,D) < \exp(-2\gamma^2 T)$$

4.1)

$$(\beta_t, h_t) = \underset{\beta, h \in H}{\operatorname{argmin}} \sum_1^n \exp\left(-y_i\left(f_{t-1}(x_i) + \beta h(x_i)\right)\right)$$

$$= \underset{\beta \in R, h \in H}{\operatorname{argmin}} \sum_1^n \exp(-y_i f_{t-1}(x_i)) \cdot \exp(-y_i \beta h(x_i))$$

$$= \underset{\beta \in R, h \in H}{\operatorname{argmin}} \sum_1^n w_i^T \exp(-y_i \beta h(x_i))$$

rescaling each w_i^t by $\frac{1}{W}$ where $W = \sum_1^n w_i^t$

So, $$\beta_t, h_t = \underset{\beta \in R, h \in H}{\operatorname{argmin}} \frac{\sum_{i=1}^n w_i^t \exp(-y_i \beta h(x_i))}{\sum_{i=1}^n w_i^t}$$

4.2)

$$\sum_1^n w_i^t \exp(-\beta y_i h(x_i))$$

$$= \sum_1^n w_i^t \exp(-\beta y_i h(x_i)) \, 1(y_i \neq h(x_i)) + \sum_{i=1}^n w_i^t \exp(-\beta) \, 1(y_i = h(x_i))$$

$$\Rightarrow \sum_1^n w_i^t \exp(-\beta y_i h(x_i))$$

$$err_t(h_t) \cdot \exp(B) + \frac{\sum w_i^t \exp(-B)(1 - 1(y_i \neq h(x_i)))}{\sum_1^n w_i^t}$$

$$= \exp(-B) + (\exp(B) - \exp(-B)) \, err_t(h)$$

4.3) if $\beta \geq 0$

$$\underbrace{e^{-\beta}}_{>0} + \underbrace{(e^{\beta} - e^{-\beta})}_{>0} err_t(h)$$

← constants

$$\Rightarrow \operatorname{argmin}(\quad)$$

$$= \operatorname{argmin}(err_t(h))$$

if $\beta < 0$

$$\Rightarrow e^{-\beta} > 0 \qquad e^{\beta} - e^{-\beta} < 0$$

$$\Rightarrow \operatorname{argmin}(\quad) = \underline{\operatorname{argmax} err_t(h)}$$

$$\Rightarrow \operatorname{argmax}\left(\frac{\sum_{i}^{n} w_i^t \, 1(y_i \neq h(x_i))}{\sum_{i=1}^{n} w_i^t}\right)$$

$$\Rightarrow \operatorname{argmin} \sum_{i=1}^{n} \frac{w_i^t \, 1(y_i \neq h(x_i))}{\sum_{i=1}^{n} w_i^t}$$

$$= \boxed{\operatorname{argmin} err_t(-h)}$$

4.4)

(1.8)

FSAM

$$\alpha_t = \frac{1}{2} \log \left(\frac{1 - err_t}{err_t} \right)$$

$$\text{score} = \sum_{s \leq T} \alpha_s h_s(x)$$

$$\text{exact Adaboost} = \sum_{s \leq T} \alpha_s G_s(x_i)$$

$$\alpha_T = \log \left(\frac{1 - err_T}{err_T} \right)$$

$\Rightarrow$ score of FSAM $= \frac{1}{2}$ score of exact adaboost

$\beta \geq 0 \Rightarrow \underset{h \in H}{\operatorname{argmin}}\ err_t(h)$

$\beta < 0 \Rightarrow \operatorname{argmin}\ err_t(-h)$

H is symmetric

$\Rightarrow \underset{h \in H}{\operatorname{argmin}}\ err_t(h),$

So in both cases $\Rightarrow$

$$\boxed{h_t = \underset{h \in H}{\operatorname{argmin}}\ err_t(h)}$$

) Fix h_t + $err_t(h) \rightarrow$ convex

$= \Rightarrow e^{-\beta}, e^{\beta} \rightarrow$ convex

$\Rightarrow e^{-\beta} + (e^{\beta} - e^{-\beta})\ err_t(h) \rightarrow$ Convex

$\Rightarrow Z = e^{\beta} - e^{-\beta}$

5) $Z = e^{-\beta_t} + (e^{\beta_t} - e^{-\beta_t})\ err_t(h)$

$= \frac{1}{\sum w_i^t} \underbrace{\exp(-\beta y_i h(x_i))}_{\rightarrow \text{ convex}}$

$w_i^t \rightarrow +ve$

$\Rightarrow Z$ is non-negative sum of convex $f^n \Rightarrow$ Convex

$\Rightarrow \frac{\partial Z}{\partial \beta} = 0 \Rightarrow \boxed{\beta_t = \frac{1}{2} \log\left(\frac{1 - err_t}{err_t}\right)}$

1.6) $w_i^{t+1} = \exp(-y_i f_t(x_i))$

$w_i^t = \exp(-y_i f_{t-1}(x_i))$

$f_t(x_i) = f_{t-1}(x_i) + \beta_t h_t(x_i)$

$\Rightarrow w_i^{t+1} = \exp(-y_i (f_{t-1}(x_i) + \beta_t h_t(x_i)))$

$= \exp(-y_i f_{t-1}(x_i)) \cdot \exp(-y_i \beta_t h_t(x_i))$

$w_i^{t+1} = w_i^t \cdot \exp(-y_i \beta_t h_t(x_i))$

$= w_i^t e^{-\beta_t} \cdot w_i^t e^{2\beta_t}$

$\Rightarrow w_i^{t+1} = e^{-\beta_t} w_i^t$ if $h_t(x_i) = y_i$

$e^{-\beta_t} w_i^t e^{2\beta_t}$ if $y_i \neq h_t(x_i)$

1.7) Exact $\Rightarrow$ $\alpha_t = \log\left(\frac{1 - err_t}{err_t}\right)$.

$f = \sum_1^T \alpha_t h_t$

Adaboost $\Rightarrow$ $\alpha_t = \frac{1}{2} \log\left(\frac{1 - err_t}{err_t}\right)$

$\Rightarrow$ differ only by a constant factor

3.3)

$$w_i^{t+1} = w_i^t \exp(-\alpha_t y_i G_t(x_i))$$

$$w_t^t = w_i^{t-1} \exp(-\alpha_{t-1} y_i G_{t-1}(x_i))$$

$$w_i^{t+1} = \exp(-y_i f_t(x_i))$$