Optional

1.1

① $g_m = \left(\frac{\partial}{\partial f_{m-1}(x_i)} \left[\frac{1}{2} (f_{m-1}(x_i) - y_i)^2 \right] \right)_{i=1}^{n}$

$= \left(f_{m-1}(x_i) - y_i \right)_{i=1}^{n}$

$h_m = \underset{h \in \mathcal{F}}{\operatorname{argmin}} \sum_{i=1}^{n} \left((-g_m)_i - h(x_i) \right)^2$

$= \underset{h \in \mathcal{F}}{\operatorname{argmin}} \sum_{i=1}^{n} \left(y_i - f_{m-1}(x_i) - h(x_i) \right)^2$

1.2

(2) $g_m = \left(\frac{\partial}{\partial f_{m-1}(x_i)} \left[\ln (1 + e^{-y_i f_{m-1}(x_i)}) \right] \right)_{i=1}^{n}$

$= \left(\frac{e^{-y_i f_{m-1}(x_i)} \times -y_i}{1 + e^{-y_i f_{m-1}(x_i)}} \right)_{i=1}^{n}$

$= \left(-y_i \cdot \frac{1}{e^{y_i f_{m-1}(x_i)} + 1} \right)_{i=1}^{n}$

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

2.1

2.

(c) $E_y[\ell(yf(x))|x] = \pi(x) \cdot E[\ell(f(x))|x] + (1-\pi(x)) E[\ell(-f(x))|x]$

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

2.2

② $\frac{\partial}{\partial f(x)} E_y[\ell(y f(x)) \| x]$

$= \pi(x)\, \ell'(f(x)) + (1-\pi(x))\, \ell'(-f(x))$.

$= \pi(x) \left[e^{-f(x)} \right] + (1-\pi(x)) \left[e^{f(x)} \right] = 0$.

$(1-\pi(x))\, e^{f(x)} = \pi(x)\, \frac{1}{e^{f(x)}}$

$e^{2f(x)} = \pi(x)\, \frac{1}{(1-\pi(x))}$

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

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

$e^{f(x)} - e^{f(x)}\pi(x) = \frac{1}{e^{f(x)}}\pi(x)$

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

$\pi(x) = \frac{e^{f(x)} \times e^{-f(x)}}{e^{f(x)} + e^{-f(x)}} = \frac{1}{1+e^{-2f(x)}}$.

2.3

③ $\partial_{f(x)} E_y[l(y, f(x)) \mid x]$

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

$= \pi(x) \frac{1}{1+e^{-f(x)}} (e^{-f(x)}) \times (-1) + (1-\pi(x)) \frac{1}{1+e^{f(x)}} e^{f(x)} = 0.$

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

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

$$\frac{\pi(x)}{1-\pi(x)} = e^{f(x)}$$

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

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

$\pi(x)[1+e^{f(x)}] = e^{f(x)}$

$\pi(x) = \frac{e^{f(x)}}{1+e^{f(x)}}$

$= \frac{1}{e^{-f(x)}+1}$

3.1 & 3.2

3. ① if $g(x)=y$

$1(g(x)\neq y)=0$ $\qquad$ $\exp(-y\,g(x)) = \exp(-1) = \frac{1}{e}$

$\therefore\ 1(g(x)\neq y) < \exp(-y\,g(x))$.

if $g(x)\neq y$

$1(g(x)\neq y)=1$ $\qquad$ $\exp(-y\,g(x)) = \exp(1) = e$

$\therefore\ 1(g(x)\neq y) < \exp(-y\,g(x))$.

② $Z_t = \frac{1}{n}\sum_{i=1}^{n} \exp(-y_i f_t(x_i)) \geq \frac{1}{n}\sum_{i=1}^{n} 1(f_t(x_i)\neq y_i) = L(G,D)$

6.1

Gradient Boosting Method

In [5]:
```
from sklearn import tree
```

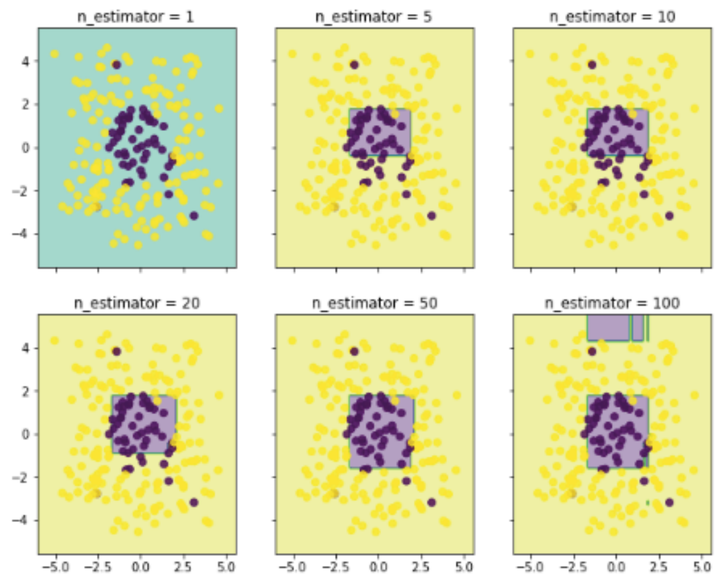
In [6]:
```
#Pseudo-residual function.
#Here you can assume that we are using L2 loss

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

In [7]:
```
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)
        :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

    def fit(self, train_data, train_target):
        '''
        Fit gradient boosting model
        '''
        h = list()
        m = 0
        length = train_data.shape[0]
        residual = np.zeros(length)
        while m <= self.n_estimator:
            if m == 0:
                for i in range(length):
                    residual[i] = self.pseudo_residual_func(train_target[i],0)
            else:
                f = np.sum(self.learning_rate * h[i].predict(train_data) for i in range(len(h)))
                for i in range(length):
                    residual[i] = self.pseudo_residual_func(train_target[i],f[i])
            clf = DecisionTreeRegressor(min_samples_split=self.min_sample,max_depth=self.max_depth)
            clf_fit = clf.fit(train_data,residual)
            h.append(clf_fit)
            m+=1
        self.h = h
        return self

    def predict(self, test_data):
        '''
        Predict value
        '''
        pred = np.sum(self.learning_rate*self.h[i].predict(test_data) for i in range(len(self.h)))
        return pred
```

2-D GBM visualization - SVM data

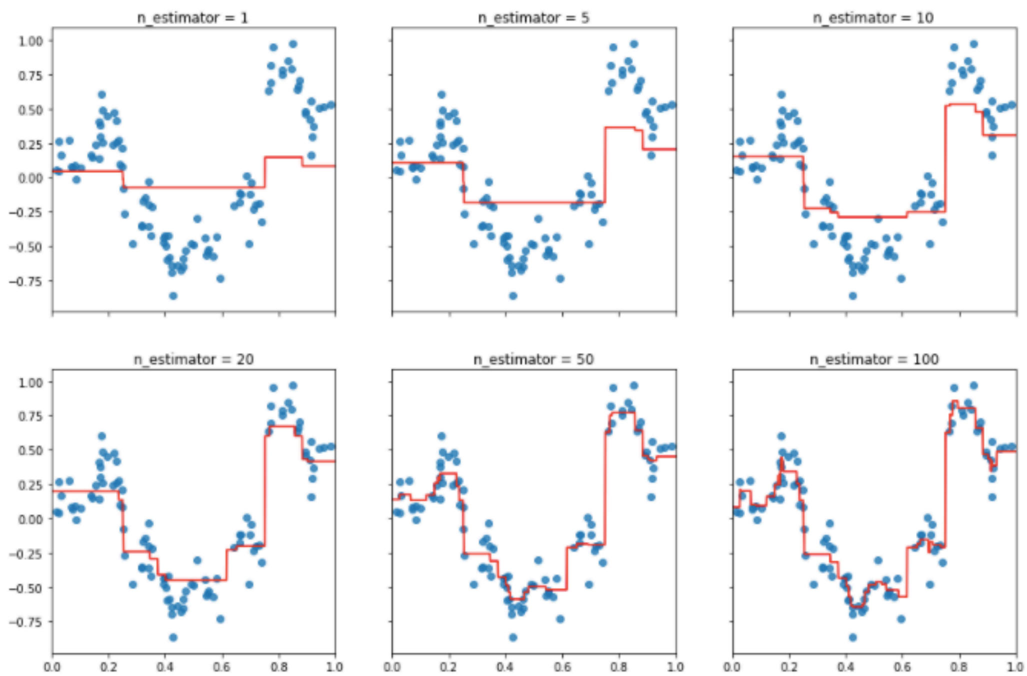
In [8]:

```
# 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_residual_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_train_label, alpha=0.8)
    axarr[idx[0], idx[1]].set_title(tt)
```



1-D GBM visualization - KRR data

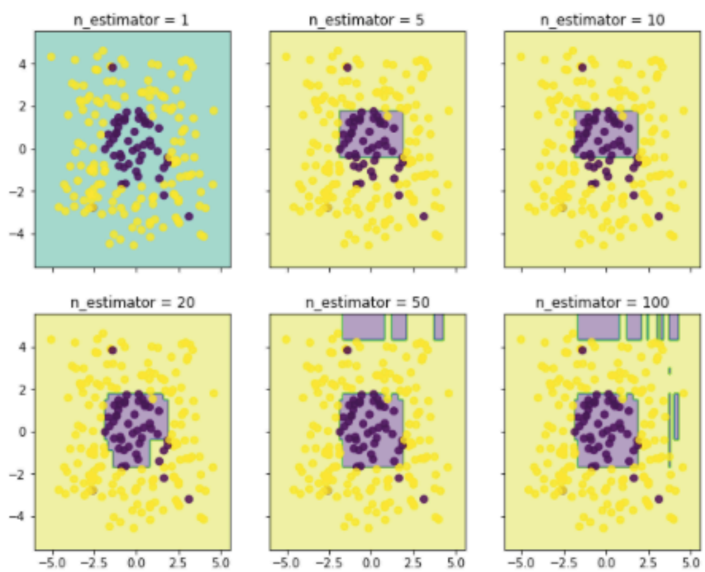
In [9]:
```
plot_size = 0.001
x_range = np.arange(0., 1., plot_size).reshape(-1, 1)

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=pseudo_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)
```



6.2

In [11]:
```
def pseudo_residual_L1(train_target, train_predict):
    '''
    Compute the pseudo-residual based on current predicted value.
    '''
    return np.sign(train_target - train_predict)
```

In [12]:
```
# 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_residual_L1, 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_train_label, alpha=0.8)
    axarr[idx[0], idx[1]].set_title(tt)
```



In [13]:
```
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=pseudo_residual_L1, 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)
```