In [1]:
import pandas as pd
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
from sklearn import tree
from sklearn.ensemble import AdaBoostClassifier, GradientBoostingClassifier
%matplotlib inline

## AdaBoost

### 9.1 Exponential loss vs misclassification loss

In boosting, we label (for convenience) the classes as $+1$ and $-1$, respectively. We furthermore let our classifier $\hat{y}(x)$ be on the form $\hat{y}(x) = \text{sign}(C(x))$, i.e., a thresholding at 0 of some real-valued function $C(x)$. Assume that we have learned some function $\hat{C}(x)$ from training data such that we can make predictions $\hat{y}(x) = \text{sign}(\hat{C}(x))$. Fill in the missing columns in the table below:

<table>
<thead>
<tr>
<th>$\hat{C}(x_*)$</th>
<th>$\hat{y}_*$</th>
<th>Exponential loss $\exp(-y_<em>\hat{C}(x_</em>))$</th>
<th>Misclassification loss $I(y_* \neq \hat{y}_*)$</th>
<th>$y_*$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.3</td>
<td>1</td>
<td>$\exp(0.3)=1.35$</td>
<td>1</td>
<td>-1</td>
</tr>
<tr>
<td>-0.2</td>
<td>-1</td>
<td>$\exp(-0.2)=0.82$</td>
<td>0</td>
<td>-1</td>
</tr>
<tr>
<td>1.5</td>
<td>1</td>
<td>$\exp(-1.5)=0.22$</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>-4.3</td>
<td>-1</td>
<td>$\exp(4.3)=73.70$</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

In what sense is the exponential loss more informative than the misclassification loss, and how can that information be used when training a classifier?

Whereas the misclassification loss only gives the information correct or misclassified, the exponential loss contains the same information ($< 1$: correct, $> 1$: misclassified) as well as information about the margin $y_*C(x_*)$, which in a sense gives an idea of "how much" wrong or correct the classifier is.

### 9.2 AdaBoost for spam classification

Consider the very same setting data set for the data set email.csv as in problem 8.2, but now use AdaBoost instead. AdaBoost is available in sklearn.ensemble as the command AdaBoostClassifier(). What test error do you achieve?
In [2]:

```python
email = pd.read_csv('Data/email.csv')

np.random.seed(1)
trainIndex = np.random.choice(email.shape[0], size=int(len(email)*.75), replace=False)
train = email.iloc[trainIndex]  # training set
test = email.iloc[-email.index.isin(trainIndex)]  # test set

X_train = train.copy().drop(columns=['Class'])
y_train = train['Class']
X_test = test.copy().drop(columns=['Class'])
y_test = test['Class']

model = AdaBoostClassifier()
model.fit(X=X_train, y=y_train)
y_predict = model.predict(X_test)
print('Test error rate is %.3f' % np.mean(y_predict != y_test))
```

Test error rate is 0.054

### 9.3 Exploring the AdaBoost algorithm

The script below illustrates the AdaBoost algorithm in the same way as was done in lecture 7. Familiarize yourself with the code and compare it to the pseudocode given in the course literature (Chapter 5 in the lecture notes). Explore what happens if you make changes in the input data and the number of trees (stumps), B, used!
In [3]:

X1 = np.array([0.2358, 0.1252, 0.4278, 0.6398, 0.6767, 0.8733, 0.3648, 0.6336, 0.3433, 0.8410])
X2 = np.array([0.1761, 0.4465, 0.7539, 0.9037, 0.7111, 0.8414, 0.6060, 0.5107, 0.1430, 0.1994])
X = np.column_stack([X1, X2])
y = np.array([1, 1, 1, 1, 1, -1, -1, -1, -1, -1])
n = len(y)

# note: the class labels are 1 and -1

fig, ax = plt.subplots()
ax.set_xlim((0, 1))
ax.set_ylim((0, 1))

# plot the data
ax.plot(X1[y==1], X2[y==1], 'x', color='red')
ax.plot(X1[y==-1], X2[y==-1], 'o', color='blue')
ax.set_xlabel('$X_1$')
ax.set_ylabel('$X_2$')

# define some variables we will fill out in the loop below
stumps = []
alpha = []
w = np.repeat(1/n, n)  # the boosting weights, start with equal weights
B = 3

# learn the boosting classifier
for b in range(B):
    fig, ax = plt.subplots()
    ax.set_xlim((0, 1))
    ax.set_ylim((0, 1))

    # use a decision stump (tree with depth 1) as base classifier
    model = tree.DecisionTreeClassifier(max_depth=1)
    model.fit(X, y, sample_weight=w)
stumps.append(model)  # Saving the model in each step

    # plot the decision boundary
    split_variable = model.tree_.feature[0]
    split_value = model.tree_.threshold[0]
    if split_variable == 0:
x1 = [0, split_value, split_value, 0]
x2 = [0, 0, 1, 1]
    ax.fill(x1, x2, 'lightsalmon', alpha=0.5)
x1 = [split_value, 1, 1, split_value]
x2 = [0, 0, split_value, split_value]
    ax.fill(x1, x2, 'skyblue', alpha=0.5)
    ax.set_xlabel('$x_1$')
    ax.set_ylabel('$x_2$')

    # plot the data
    ax.scatter(X1[y==1], X2[y==1], s=50, facecolors='red')
    ax.scatter(X1[y==-1], X2[y==-1], s=50, facecolors='blue')

    # compute and plot the boosting weights for the next iteration
    correct = model.predict(X) == y
    ax.scatter(X1[~correct], X2[~correct], s=300, facecolors='none', edgecolors='k')
9.4 Deriving the AdaBoost weights
The AdaBoost classifier can be written as
\[ \hat{y}_{\text{boost}}(x) = \text{sign}(C^B(x)) \]
where the functions \( C^1(x), \ldots, C^B(x) \) are constructed sequentially as
\[ C^b(x) = C^{b-1}(x) + \alpha_b \hat{y}^b(x), \]
initialized with \( C^0(x) \equiv 0 \). The \( b \)th ensemble member \( \hat{y}^b(x) \) is found by applying the chosen base classifier to a weighted version of the training data. Once this has been found, we also need to compute the corresponding "confidence" coefficient \( \alpha_b \). This is done by minimizing the weighted exponential loss of the training data,
\[ \alpha_b = \arg \min_{\alpha} \left\{ \sum_{i=1}^{N} w_i^b \exp(-\alpha y_i \hat{y}^b(x_i)) \right\} \]
where \( w_i^b = \exp(-y_i C^{b-1}(x_i)) \).

Show that the optimal solution is given by
\[ \alpha_b = \frac{1}{2} \log \left( \frac{1 - \tilde{\epsilon}_{\text{err}}}{\tilde{\epsilon}_{\text{err}}} \right) \]
where
\[ \tilde{\epsilon}_{\text{err}} = \sum_{i=1}^{N} \frac{w_i^b}{\sum_{j=1}^{N} w_j^b} I(y_i \neq C^b(x_i)). \]

_Hint 1_: We use class labels \(-1\) and \(1\), i.e. \( y_i \in \{-1, 1\} \) and \( \hat{y}^b(x_i) \in \{-1, 1\} \). Using this fact, divide the sum in the objective function in the equation for \( \alpha_b \) into one sum ranging over all correctly classified training data points and one sum ranging over all misclassified training data points.

_Hint 2_: You are allowed to use the fact that the objective function in the equation for \( \alpha_b \) has a single stationary point corresponding to the global minima.
We want to minimize the expression

\[ F(\alpha) := \sum_{i=1}^{N} w_i \exp(-\alpha y_i \hat{y}_i(x_i)) \]

with respect to \( \alpha \). Using the second hint of the problem formulation, we can do this by differentiating the expression, setting the derivative to zero, and solving for \( \alpha \).

Before we do this, however, we make use of the first hint of the problem formulation to simplify the expression under study. Note first that using the class labels \(-1 \) and \( 1 \) implies that

\[ y_i \hat{y}_i^b(x_i) = \begin{cases} 
1 & \text{if } \hat{y}_i^b(x_i) = y_i, \\
-1 & \text{if } \hat{y}_i^b(x_i) \neq y_i.
\end{cases} \]

Thus, the expression for \( F(\alpha) \) can be written as

\[ F(\alpha) = e^{-\alpha} \sum_{i=1}^{N} w_i I(\hat{y}_i^b(x_i) = y_i) + e^{\alpha} \sum_{i=1}^{N} w_i I(\hat{y}_i^b(x_i) \neq y_i), \]

where we have used the indicator function to split the sum into two sums: the first ranging over all correctly classified data points, and the second ranging over all erroneously classified data points. Furthermore, for notational simplicity we define \( W_c \) and \( W_e \) for the sum of weights of correctly classified data points and erroneously classified data points, respectively.

Now, we compute the derivative and set to zero:

\[ \frac{dF}{d\alpha} = -e^{-\alpha} W_c + e^{\alpha} W_e = 0 \iff e^{2\alpha} = \frac{W_c}{W_e} \iff \alpha = \frac{1}{2} \log \left( \frac{W_c}{W_e} \right). \]

It remains to write the solution on the format asked for in the problem formulation. With \( W := \sum_{i=1}^{N} w_i \) denoting the sum of all weights, we note first that \( W_c = W - W_e \) and thus,

\[ \alpha = \frac{1}{2} \log \left( \frac{W - W_e}{W_e} \right) = \frac{1}{2} \log \left( \frac{1 - W_e}{W_e} \right). \]

Finally, noting that \( \text{err}_b = \frac{W_e}{W} \) completes the proof.

### 9.5 Gradient boosting

Explore gradient boosting by using `GradientBoostingClassifier()` on the spam data set `email.csv`

```python
In [4]:
   : model = GradientBoostingClassifier()
   : model.fit(X=X_train, y=y_train)
   : y_predict = model.predict(X_test)
   : print('Test error rate is %.3f' % np.mean(y_predict != y_test))

Test error rate is 0.052`