HW6

In [83]:

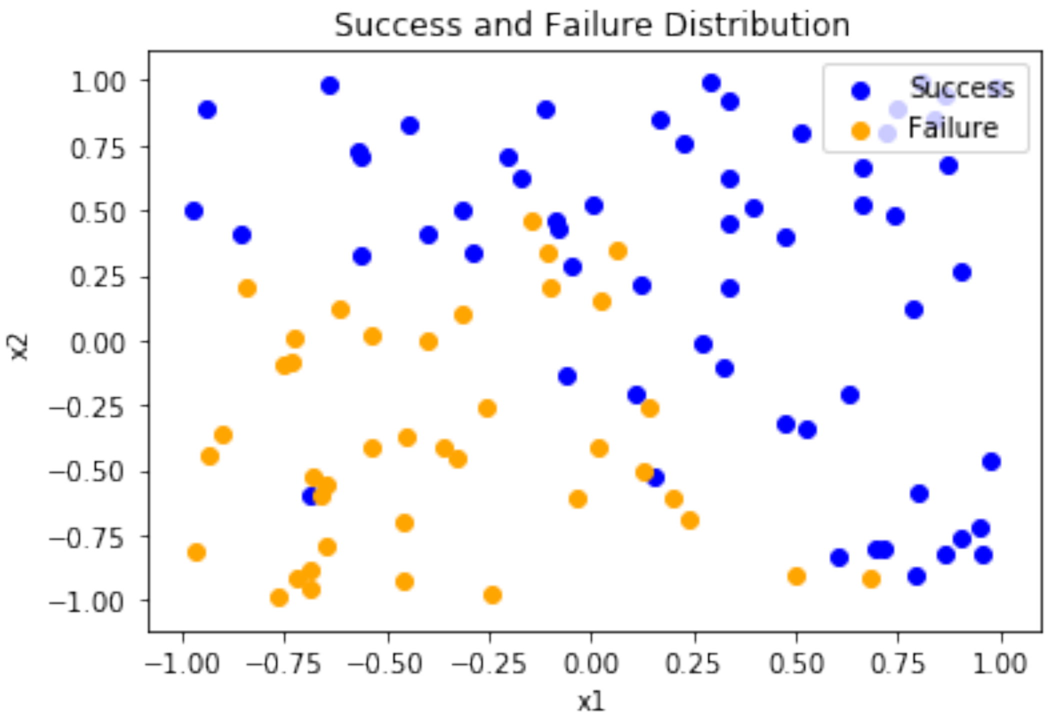
```
import random
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sb
import math
from sklearn import svm
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
#disable future warning
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
```

Conceptual exercises(1-14)

Q1.(15 points) Generate a simulated two-class data set with 100 observations and two features in which there is a visible (clear) but still nonlinear separation between the two classes. Show that in this setting, a support vector machine with a radial kernel will outperform a support vector classifier (a linear kernel) on the training data. Which technique performs best on the test data? Make plots and report training and test error rates in order to support your conclusions.

In [84]:

```
np.random.seed(2020)
#build predictors x1 & x2
x1 = np.random.uniform(-1,1,100)
x2 = np.random.uniform(-1,1,100)
#build error term
err = np.random.normal(0, 0.5, 100)
#calculate Y
y = x1 + x1**2 +x1**3 + x2+ x2**2 + x2**3 + err
suc_pbb = math.e**y/(1+math.e**y)
success = suc_pbb > 0.5  #boolean value index
failure = suc_pbb <= 0.5
#plot x1 and x2
plt.scatter(x1[success], x2[success], color = 'blue')
plt.scatter(x1[failure], x2[failure], color = 'orange')
plt.xlabel('x1')
plt.ylabel('x2')
plt.title('Success and Failure Distribution')
plt.legend(['Success', 'Failure'], loc=1);
```



In [85]:

```
X = pd.DataFrame({'x1': x1, 'x2': x2})
```

In [86]:

```
X_train, X_test, y_train, y_test = train_test_split(
    X, success, test_size=0.2,
    random_state=42)
```

In [87]:

```
svm_radial = SVC(gamma='scale').fit(
    X_train, y_train)
svm_linear = SVC(kernel='linear').fit(
    X_train, y_train)
```

In [88]:

```
print("Training Error:")
print("SVM with a radial kernel has an error rate of: ",
      round(1-svm_radial.score(X_train, y_train),4))
print("SVM with a linear kernel has an error rate of: ",
      round(1-svm_linear.score(X_train, y_train),4))
print("Test Error:")
print("SVM with a radial kernel has an error rate of: ",
      round(1-svm_radial.score(X_test, y_test),4))
print("SVM with a linear kernel has an error rate of: ",
      round(1-svm_linear.score(X_test, y_test),4))
```

```
Training Error:
SVM with a radial kernel has an error rate of:  0.11
25
SVM with a linear kernel has an error rate of:  0.11
25
Test Error:
SVM with a radial kernel has an error rate of:  0.1
SVM with a linear kernel has an error rate of:  0.15
```

We can see from the result that SVM with radial kernel performs better than SVM with linear kernel in both training and testing dataset.

SVM vs. logistic regression

Q2.(5 points) Generate a data set with n = 500 and p = 2, such that the observations belong to two classes with some overlapping, non-linear boundary between them.

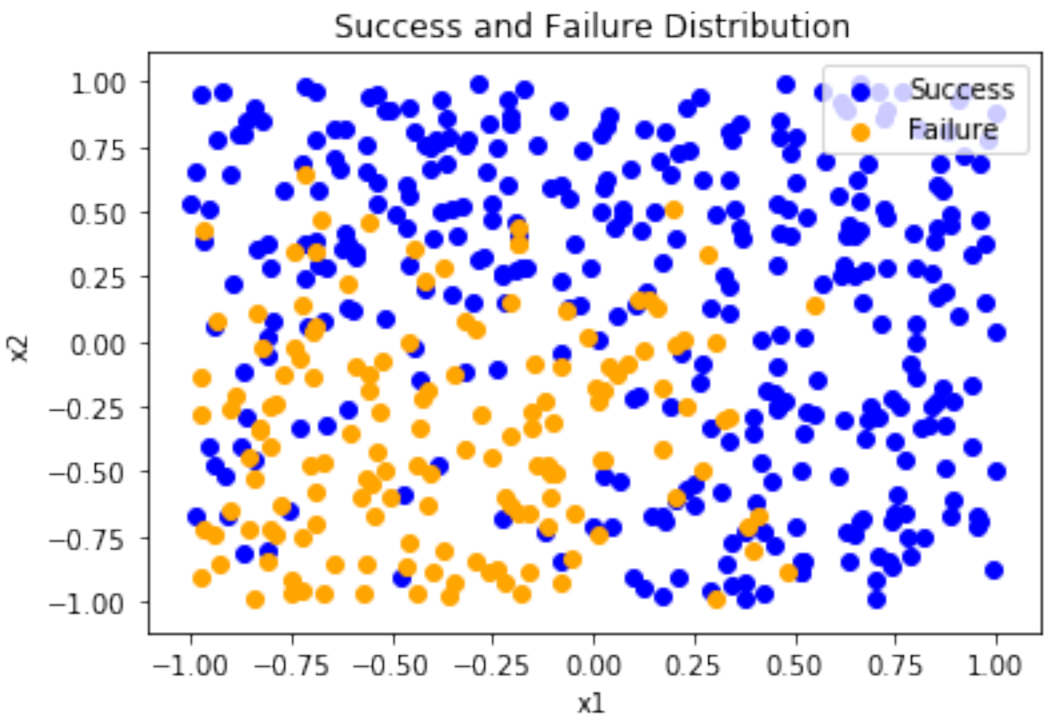
In [89]:

```
np.random.seed(2020)

x3 = np.random.uniform(-1,1,500)
x4 = np.random.uniform(-1,1,500)
# build error term
err2 = np.random.normal(0,0.5,500)
# calculate y
y2 = x3 + x3**2 + x4 + x4**2 + err2

suc_pbb = math.e**y2/(1+math.e**y2)
success2 = suc_pbb > 0.5 #boolean value index
failure2 = suc_pbb <= 0.5
```

Q3. (5 points) Plot the observations with colors according to their class labels (y). Your plot should display X1 on the x-axis and X2 on the y-axis.

In [90]:

```
plt.scatter(x3[success2], x4[success2], color = 'blue')
plt.scatter(x3[failure2], x4[failure2], color = 'orange')
plt.xlabel('x1')
plt.ylabel('x2')
plt.title('Success and Failure Distribution')
plt.legend(['Success', 'Failure'], loc=1);
```



Q4. (5 points) Fit a logistic regression model to the data, using X1 and X2 as predictors.

In [91]:

```
X = pd.DataFrame({'x1': x3, 'x2': x4})
```

In [92]:

```
lr = LogisticRegression().fit(X, success2)
```

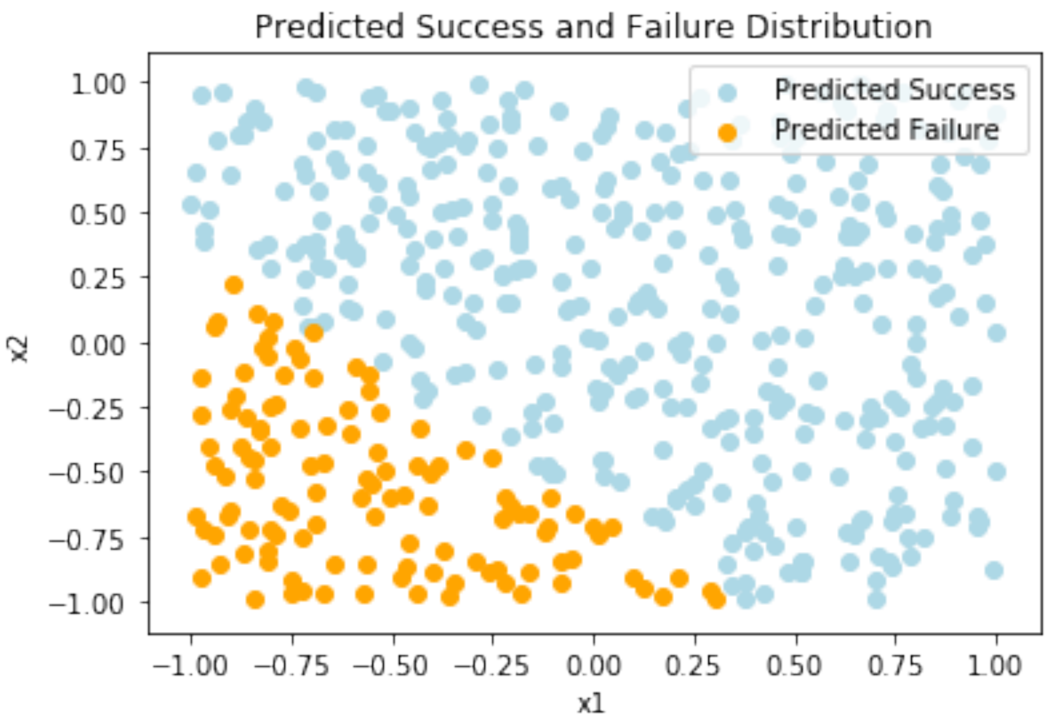
Q5.(5 points) Obtain a predicted class label for each observation based on the logistic model previously fit. Plot the observations, colored according to the predicted class labels (the predicted decision boundary should look linear).

In [93]:

```
lr_pred = lr.predict(X)
```

In [94]:

```
plt.scatter(x3[lr_pred], x4[lr_pred], color = 'lightblue')
plt.scatter(x3[~lr_pred], x4[~lr_pred], color = 'orange')
plt.xlabel('x1')
plt.ylabel('x2')
plt.title('Predicted Success and Failure Distribution')
plt.legend(['Predicted Success', 'Predicted Failure'], loc=1);
```



Q6.(5 points) Now fit a logistic regression model to the data, but this time using some non-linear function of both X1 and X2 as predictors (e.g. X12, X1 × X2, log(X2), and so on).

In [95]:

```
X_interaction = pd.DataFrame({'x1': x3*x4})
lm_interaction = LogisticRegression().fit(
    X_interaction, success2)
```

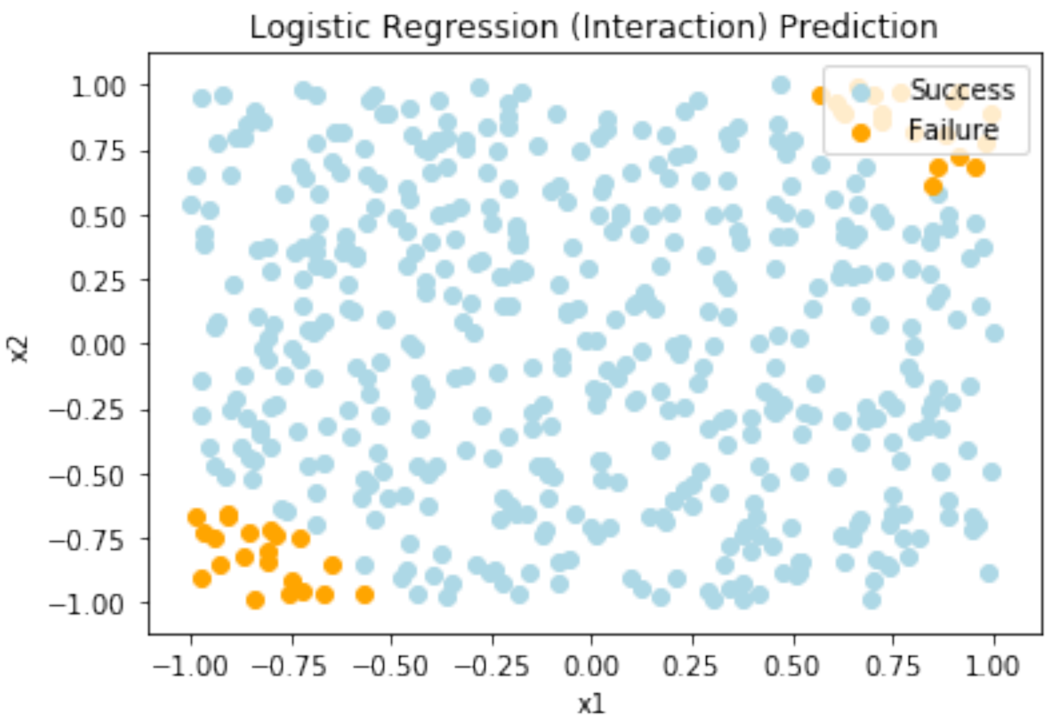
Q7. (5 points) Now, obtain a predicted class label for each observation based on the fitted model with non-linear transformations of the X features in the previous question. Plot the observations, colored according to the new predicted class labels from the non-linear model (the decision boundary should now be obviously non-linear). If it is not, then repeat earlier steps until you come up with an example in which the predicted class labels and the resultant decision boundary are clearly non-linear.

In [96]:

```
#(interaction)
lm_interaction_pred = lm_interaction.predict(X_interaction)
# plot classification (interaction)
plt.scatter(x3[lm_interaction_pred],
            x4[lm_interaction_pred], color='lightblue')
plt.scatter(x3[~lm_interaction_pred],
            x4[~lm_interaction_pred], color='orange')
plt.xlabel('x1')
plt.ylabel('x2')
plt.legend(['Success', 'Failure'],loc=1)
plt.title('Logistic Regression (Interaction) Prediction')
```

Out[96]:

```
Text(0.5,1,'Logistic Regression (Interaction) Predic
tion')
```



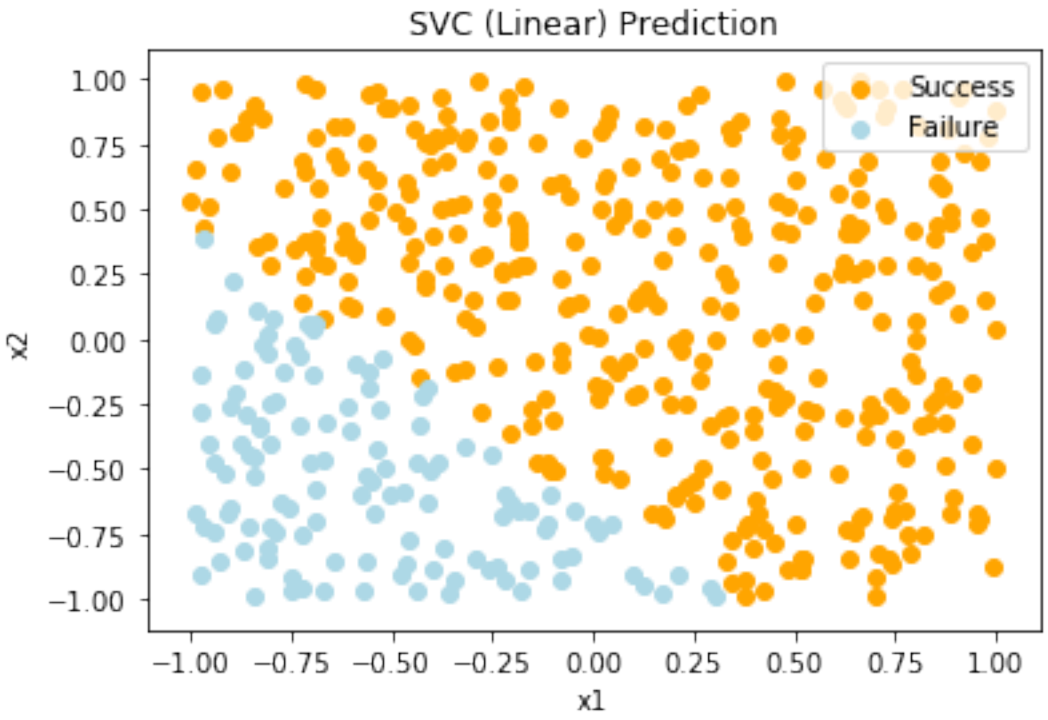
Q8. (5 points) Now, fit a support vector classifier (linear kernel) to the data with original X1 and X2 as predictors. Obtain a class prediction for each observation. Plot the observations, colored according to the predicted class labels.

In [100]:

```
svm = SVC(kernel='linear').fit(X, success2)
svm_pred = svm.predict(X)
plt.scatter(x3[svm_pred], x4[svm_pred], color='orange')
plt.scatter(x3[~svm_pred], x4[~svm_pred], color='lightblue')
plt.xlabel('x1')
plt.ylabel('x2')
plt.legend(['Success', 'Failure'],loc=1)
plt.title('SVC (Linear) Prediction')
```

Out[100]:

```
Text(0.5,1,'SVC (Linear) Prediction')
```



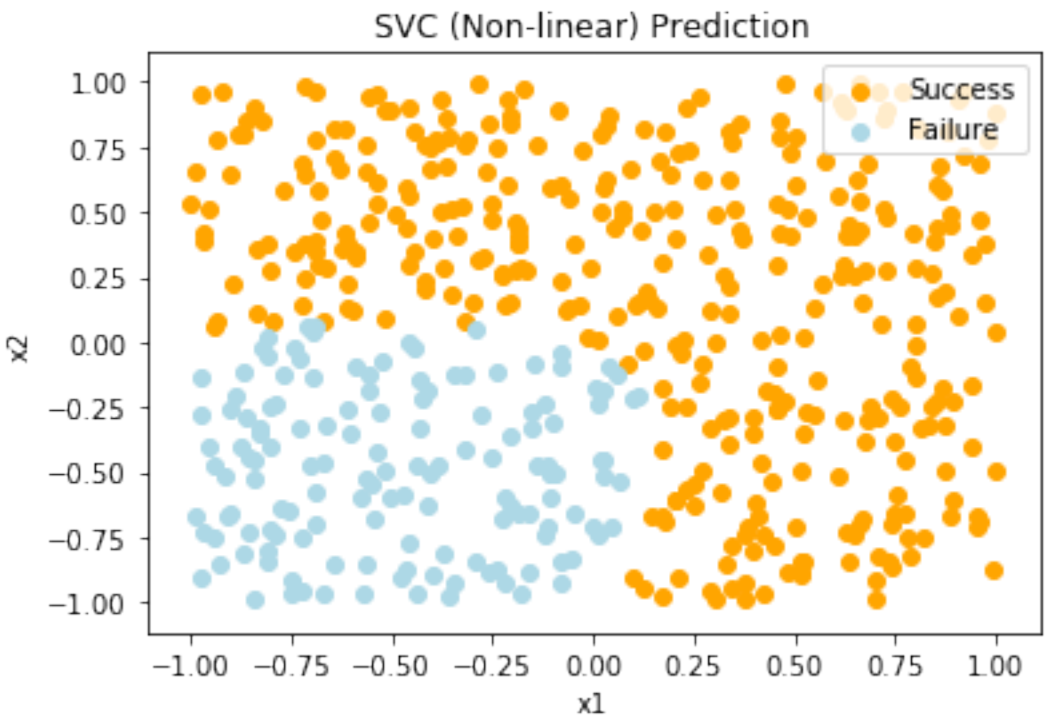
Q9. (5 points) Fit a SVM using a non-linear kernel to the data. Obtain a class prediction for each observation. Plot the observations, colored according to the predicted class labels.

In [98]:

```
svm2 = SVC(gamma='scale').fit(X, success2)
svm_pred = svm2.predict(X)
plt.scatter(x3[svm_pred], x4[svm_pred], color='orange')
plt.scatter(x3[~svm_pred], x4[~svm_pred], color='lightblue')
plt.xlabel('x1')
plt.ylabel('x2')
plt.legend(['Success', 'Failure'],loc=1)
plt.title('SVC (Non-linear) Prediction')
```

Out[98]:

```
Text(0.5,1,'SVC (Non-linear) Prediction')
```



Q10. (5 points) Discuss your results and specifically the tradeoffs between estimating non-linear decision boundaries using these two different approaches.

In [99]:

```
print("Accuracy:")
print("Logistic Regression (linear): ", lr.score(X, success2))
print("Logistic Regression (interaction): ", lm_interaction.scor
e(X_interaction, success2))
print("SVM (linear): ", svm.score(X, success2))
print("SVM (non-linear): ", svm2.score(X, success2))
```

```
Accuracy:
Logistic Regression (linear):  0.8
Logistic Regression (interaction):  0.686
SVM (linear):  0.804
SVM (non-linear):  0.844
```

The graph and the accuracy list above show that logistic regression with non-linear functions of X1 & X2 as predictors does not perform well. By comparison, SVM with non-linear kernel performs better than the rest of the the classifiers.

Tuning cost

Q11. (5 points) Generate two-class data with p = 2 in such a way that the classes are just barely linearly separable.

In [ ]:

```
np.random.seed(2020)
x1 = np.random.uniform(-1,1,500)
x2 = np.random.uniform(-1,1,500)
e = np.random.normal(0, 0.5, 500)
#calculate Y
y = x1 + x2+ e
suc_pbb = math.e**y/(1+math.e**y)

X = pd.DataFrame({'x1': x1, 'x2': x2})
success = suc_pbb > 0.5 #boolean value index
failure = suc_pbb <= 0.5
```

In [107]:

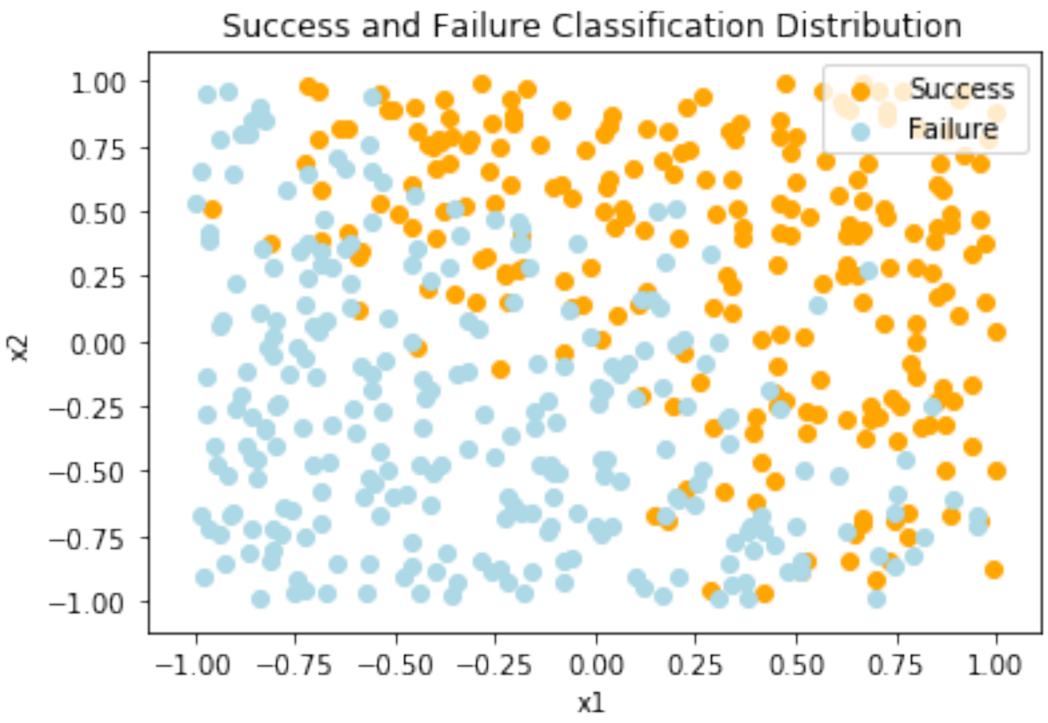
```
X_train, X_test, y_train, y_test = train_test_split(
    X, success, test_size=0.2, random_state=42)
```

In [108]:

```
plt.scatter(x1[success], x2[success], color='orange')
plt.scatter(x1[failure], x2[failure], color='lightblue')
plt.xlabel('x1')
plt.ylabel('x2')
plt.legend(['Success', 'Failure'],loc=1)
plt.title('Success and Failure Classification Distribution')
```

Out[108]:

```
Text(0.5,1,'Success and Failure Classification Distr
ibution')
```



Q.12. (5 points) Compute the cross-validation error rates for support vector classifiers with a range of cost values. How many training errors are made for each value of cost considered, and how does this relate to the cross-validation errors obtained?

In [41]:

```
X = pd.DataFrame({'x1': x1, 'x2': x2})
X_train, X_test, y_train, y_test = train_test_split(
    X, success, test_size=0.2,
    random_state=42)
```

In [42]:

```
costs = [0.001,0.01,0.1,1,3,5,7,9]
```

In [43]:

```
cv_error = []
training_error = []
for c in costs:
    model = SVC(C=c, kernel='linear', random_state=42)
    cv = 1 - np.mean(cross_val_score(model,
                                     X_train, y_train,
                                     cv=10, scoring='accuracy'))
    cv_error.append(cv)
    train = 1 - model.fit(X_train,
                          y_train).score(X_train, y_train)
    training_error.append(train)
error_rates = pd.DataFrame({'Cost': costs,
                            'CV Error': cv_error,
                            'Training Error': training_error})
```

In [44]:

```
error_rates
```

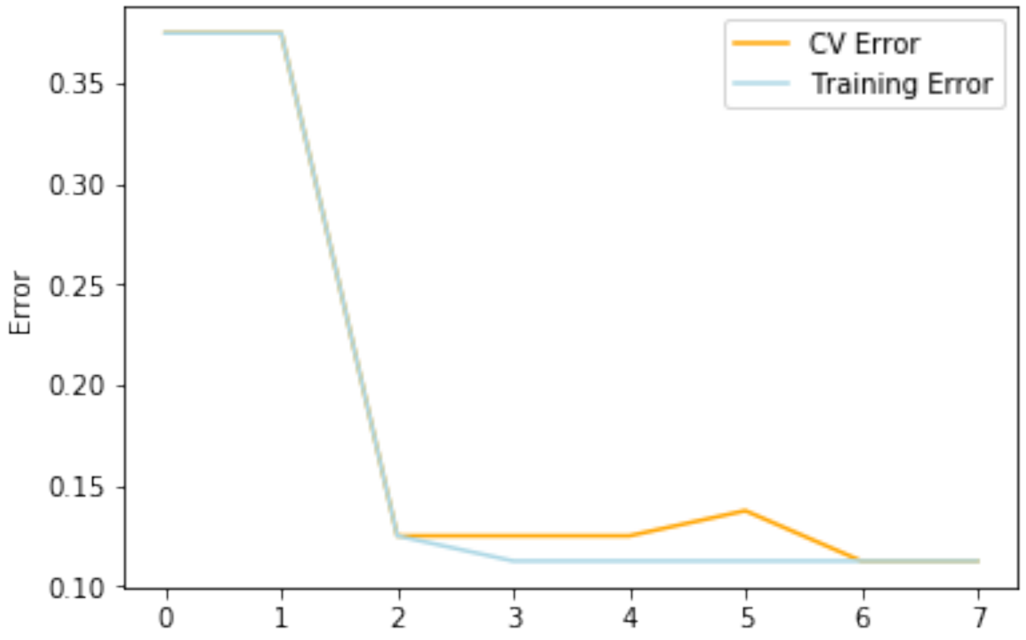
Out[44]:

	Cost	CV Error	Training Error
0	0.001	0.3750	0.3750
1	0.010	0.3750	0.3750
2	0.100	0.1250	0.1250
3	1.000	0.1250	0.1125
4	3.000	0.1250	0.1125
5	5.000	0.1375	0.1125
6	7.000	0.1125	0.1125
7	9.000	0.1125	0.1125

```
In [45]:
```

```
plt.plot(cv_error, color='orange')
plt.plot(training_error, color='lightblue')
plt.ylabel('Error')
plt.legend(['CV Error', 'Training Error'],loc=1)
```

```
Out[45]:

<matplotlib.legend.Legend at 0x1a24596d10>
```



When the cost value goes up, the training error remains to the maximum and later decreases rapidly from cv=1 to cv=2, then it almost remains to be a plateau. The CV error curve and the training error curve almost perfectly match with each other from cost value 0 to 2 and six beyond. The error range from 0.10 to 0.38.

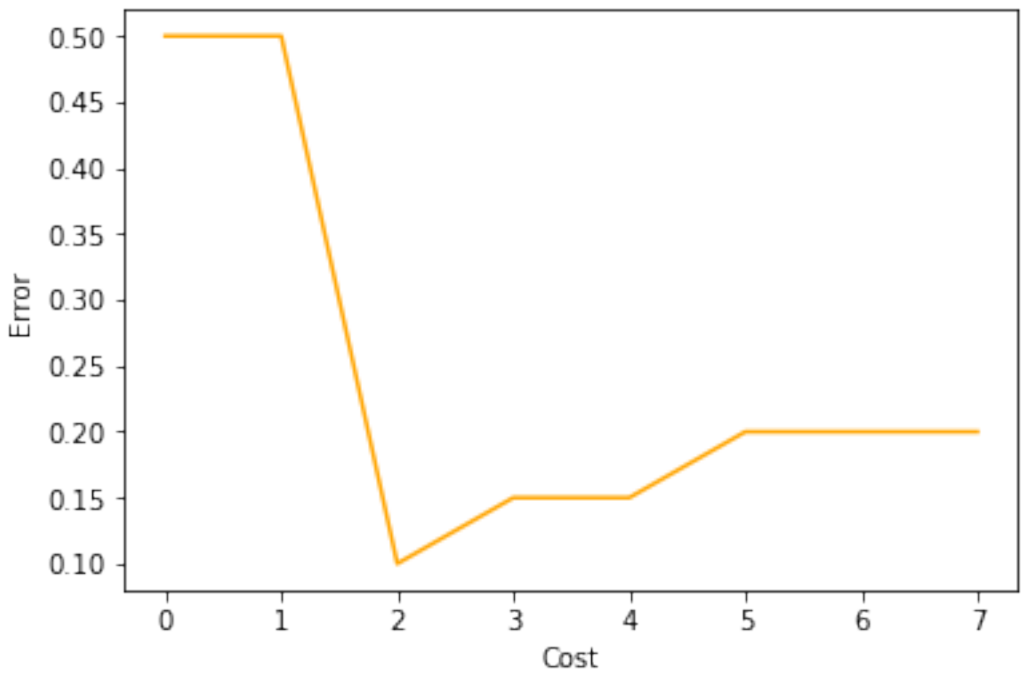
Q.13. (5 points) Generate an appropriate test data set, and compute the test errors corresponding to each of the values of cost considered. Which value of cost leads to the fewest test errors, and how does this compare to the values of cost that yield the fewest training errors and the fewest cross-validation errors?

In [46]:

```
test_error = []
for cost in costs:
    model = SVC(C=cost, kernel='linear', random_state=42)
    test = 1 - model.fit(X_train, y_train).score(X_test, y_test)
    test_error.append(test)
plt.plot(test_error, color='orange')
plt.xlabel('Cost')
plt.ylabel('Error')
```

Out[46]:

```
Text(0,0.5,'Error')
```



Q14. (5 points) Discuss your results.

The graphs above show that if predictors and the response value are close to linear, tuning parameter is not necessarily to be high to have the lowest training and test errors. Rather, we can have low tuning parameter instead.

Application: Predicting attitudes towards racist college professors

Q15. (5 points) Fit a support vector classifier to predict colrac as a function of all available predictors, using 10-fold cross-validation to find an optimal value for cost. Report the CV errors associated with different values of cost, and discuss your results.

In [47]:

```
gss_train = pd.read_csv("data/gss_train.csv")
gss_test = pd.read_csv("data/gss_test.csv")
X_train = gss_train.drop('colrac',axis=1)
y_train = gss_train['colrac']
```

In [48]:

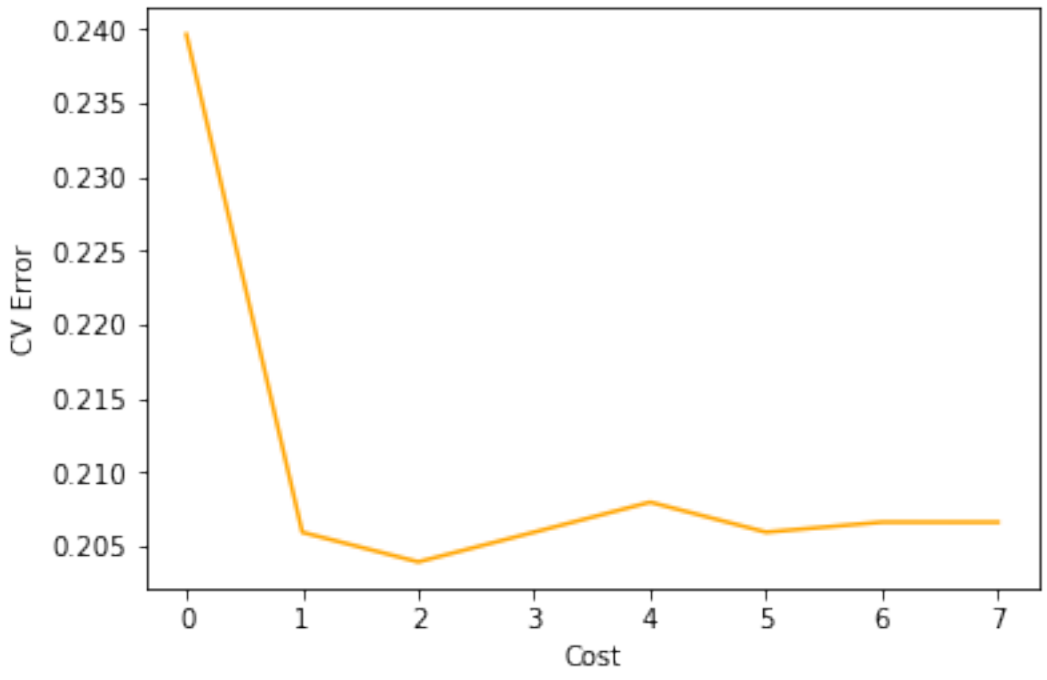
```
cv_error = []
for cost in costs:
    model = SVC(C=cost, kernel='linear', random_state=42)
    cv = 1 - np.mean(cross_val_score(model,
                                     X_train, y_train,
                                     cv=10,scoring='accuracy'))
    cv_error.append(cv)
```

In [49]:

```
plt.plot(cv_error, color='orange')
plt.xlabel('Cost')
plt.ylabel('CV Error')
```

Out[49]:

```
Text(0,0.5,'CV Error')
```



When C=2, we have the lowest CV error. The CV error fluctuates within 0.202 to 0.240.

Q16. (15 points) Repeat the previous question, but this time using SVMs with radial and polynomial basis kernels, with different values for gamma and degree and cost. Present and discuss your results (e.g., fit, compare kernels, cost, substantive conclusions across fits, etc.).

In [51]:

```
models = {SVC(kernel='rbf'):
          {'C': [0.1, 0.5, 2],
           'degree': [3, 4, 5],
           'gamma':['scale', 'auto']},
SVC(kernel='poly'): {'C':
                     [0.1, 0.5, 2],
                     'degree': [3, 4, 5],
                     'gamma':['scale', 'auto']}}
```

In [52]:

```
for model, param in models.items():
    gscv = GridSearchCV(model, param, cv=10, refit=True)
    gscv.fit(X_train, y_train)
    print(gscv.best_estimator_)
    print(gscv.best_score_)
    print('**********')
```

```
SVC(C=2, cache_size=200, class_weight=None, coef0=0.
0,
    decision_function_shape='ovr', degree=3, gamma='
scale', kernel='rbf',
    max_iter=-1, probability=False, random_state=Non
e, shrinking=True,
    tol=0.001, verbose=False)
0.761647535449021
**********
SVC(C=2, cache_size=200, class_weight=None, coef0=0.
0,
    decision_function_shape='ovr', degree=3, gamma='
scale', kernel='poly',
    max_iter=-1, probability=False, random_state=Non
e, shrinking=True,
    tol=0.001, verbose=False)
0.7629979743416611
**********
```

Using the GridSearchCV method, we see that for SVM with radial kernel, the best parameter is C=2, degree=2, and gamma=‘scale’. The mean accuracy score is 0.7616. For SVM with polynomial basis kernel, the best parameters are C=2, degree=2, and gamma=‘scale’. The mean accuracy score is 0.7629. The polynimoial kernel performs a bit better than the radial kernel. Other than accuracy, we should also consider the efficiency of training models and tuning parameters. From this perspective, SVC (linear) is pretty good.