In [69]:
```
import matplotlib.pyplot as plt
import seaborn as sns
from tqdm import tqdm
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import Pipeline
import numpy as np
from sklearn import svm
from sklearn.svm import SVC, SVR
from sklearn.metrics import accuracy_score
from sklearn.model_selection import GridSearchCV, train_test_split
import pandas as pd
from sklearn.model_selection import cross_val_score
```

In [2]:
```
np.random.seed(666)
```

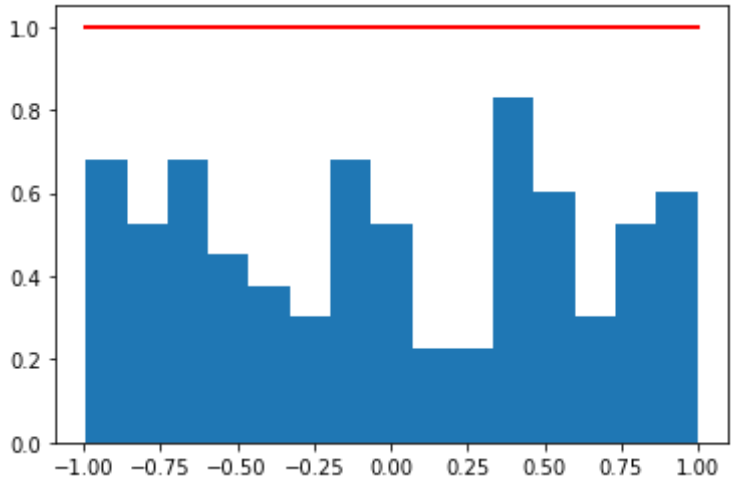
Non-linear separation

(15 points) 1. Generate a simulated two-class data set with 100 observations and two features in which there is a visible (clear) but still non-linear 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 [3]:
```
#Generating values for X1
X1 = np.random.uniform(-1,1,100)
count, bins, ignored = plt.hist(X1, 15, normed=True)
plt.plot(bins, np.ones_like(bins), linewidth=2, color='r')
plt.show()
```

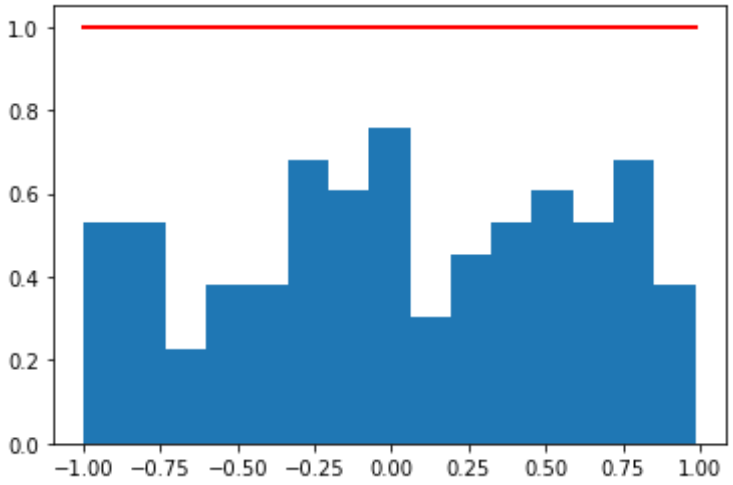
```
/Users/Sruti/opt/anaconda3/lib/python3.7/site-packages/ipykernel_launch
er.py:3: MatplotlibDeprecationWarning:
The 'normed' kwarg was deprecated in Matplotlib 2.1 and will be removed
```

```
in 3.1. Use 'density' instead.
  This is separate from the ipykernel package so we can avoid doing imp
orts until
```



In [4]:
```
#Generating values for X2
X2 = np.random.uniform(-1,1,100)
count, bins, ignored = plt.hist(X2, 15, normed=True)
plt.plot(bins, np.ones_like(bins), linewidth=2, color='r')
plt.show()
```

```
/Users/Sruti/opt/anaconda3/lib/python3.7/site-packages/ipykernel_launch
er.py:3: MatplotlibDeprecationWarning:
The 'normed' kwarg was deprecated in Matplotlib 2.1 and will be removed
in 3.1. Use 'density' instead.
  This is separate from the ipykernel package so we can avoid doing imp
orts until
```



In [5]:
```
e = np.random.normal(0, .5, 100)

Y = X1 + (X1**2) + X2 + (X2**2)  + e
```

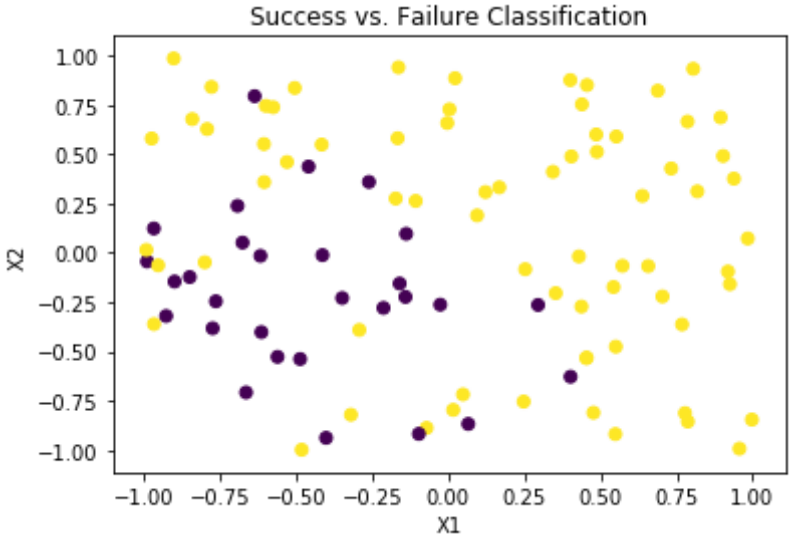
In [6]:
```
#Output = logodds, converting to probabilities
Y_prob = np.exp(Y)/(1+np.exp(Y))
Y_prob

#Using probabilites to classify (0.5 threshold)
Y_input = np.where(Y_prob > 0.5, 1, 0)
```

In [79]:
```
#Scatterplot of X1, X2
plt.scatter(X1, X2, c = Y_input)
plt.title("Success vs. Failure Classification")
plt.xlabel('X1')
plt.ylabel('X2')
plt.show()

#Had some problems plotting legend- manually generated:
plot_legend = [["Success", " Yellow"], ["Failure", "Purple"]]
plot_legend = pd.DataFrame(plot_legend)
```

```
plot_legend.rename(columns={0:'Type', 1: 'Color'}, inplace=True)
plot_legend
```



Out[79]:

	Type	Color
0	Success	Yellow
1	Failure	Purple

In [80]:
```
X_mod = np.stack((X1,X2), axis=1)
X_mod = pd.DataFrame(X_mod)
```

In [81]:
```
X_train, X_test, y_train, y_test = train_test_split(X_mod, Y_input, tes
t_size=0.2, random_state=42)
```

In [82]:
```
svm_radial = svm.SVC().fit(X_train, y_train)
```

```
/Users/Sruti/opt/anaconda3/lib/python3.7/site-packages/sklearn/svm/bas
e.py:193: FutureWarning: The default value of gamma will change from 'a
uto' to 'scale' in version 0.22 to account better for unscaled feature
```

```
s. Set gamma explicitly to 'auto' or 'scale' to avoid this warning.
  "avoid this warning.", FutureWarning)
```

In [83]:
```
svm_linear = svm.SVC(kernel='linear').fit(X_train, y_train)
```

In [87]:
```
print("Training data SVM radial kerneling acc:", svm_radial.score(X_tra
in, y_train))
print("Test data SVM radial kerneling acc:", svm_radial.score(X_test, y
_test))
print("Training data SVM linear kernel acc:", svm_linear.score(X_train,
 y_train))
print("Test data SVM linear kernel acc:", svm_linear.score(X_test, y_te
st))
```

```
Training data SVM radial kerneling acc: 0.825
Test data SVM radial kerneling acc: 0.85
Training data SVM linear kernel acc: 0.75
Test data SVM linear kernel acc: 0.85
```

Based on this output, SVM with radial kerneling performs better than SVM linear kernel for the training data (which we expected), and equally well on the test data - which is surprising, because I would've expected SVM with radial kerneling to perform better on the test set as well. This would have made more sense given that there's a non-linear separation present, making a linear classifier an inappropriate fit.

SVM vs. logistic regression

We have seen that we can fit an SVM with a non-linear kernel in order to perform classification using a non-linear decision boundary. We will now see that we can also obtain a non-linear decision boundary by performing logistic regression using non-linear transformations of the features. Your goal here is to compare different approaches to estimating non-linear decision boundaries, and thus assess the benefits and drawbacks of each.

1. (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 [153]:
```
np.random.seed(666)
x1 = np.random.uniform(-1,1,500)
x2 = np.random.uniform(-1,1,500)
e = np.random.normal(0, 0.5, 500)
```

In [154]:
```
y = x1 + (x1**2) + x2 + (x2**3)  + e
```

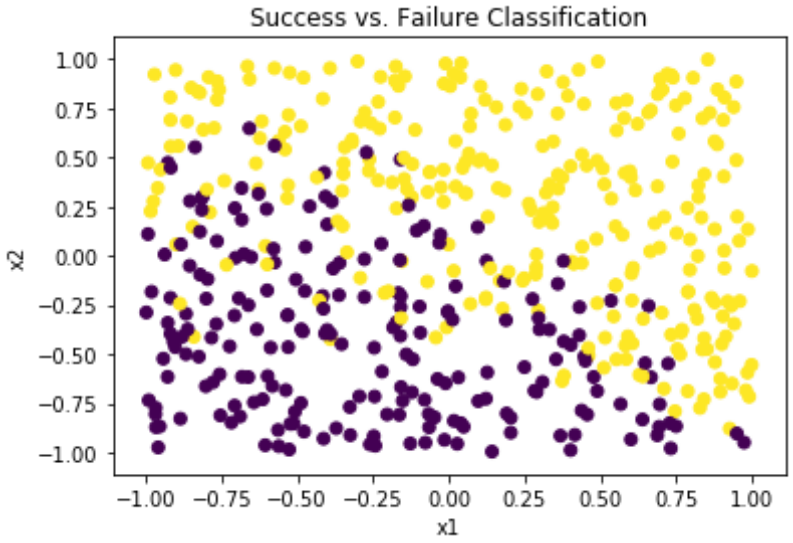
In [155]:
```
import math

p_success = math.e**y/(1+math.e**y)
s_class = p_success > 0.5 #create boolean array for success
f_class = p_success <= 0.5 #create boolean array for failures
```

1. (5 points) Plot the observations with colors according to their class labels (y). Your plot should display X_1 on the x-axis and X_2 on the y-axis.

In [99]:
```
#Scatterplot of X1, X2
plt.scatter(x1, x2, c = s_class)
plt.title("Success vs. Failure Classification")
plt.xlabel('x1')
plt.ylabel('x2')
plt.show()

#Had some problems plotting legend- manually generated:
plot_legend = [["Success", " Yellow"], ["Failure", "Purple"]]
plot_legend = pd.DataFrame(plot_legend)
plot_legend.rename(columns={0:'Type', 1: 'Color'}, inplace=True)
plot_legend
```



Out[99]:

	Type	Color
0	Success	Yellow
1	Failure	Purple

1. (5 points) Fit a logistic regression model to the data, using X_1 and X_2 as predictors.

In [105]:
```
x_linreg = np.stack((x1, x2), axis=1)
```

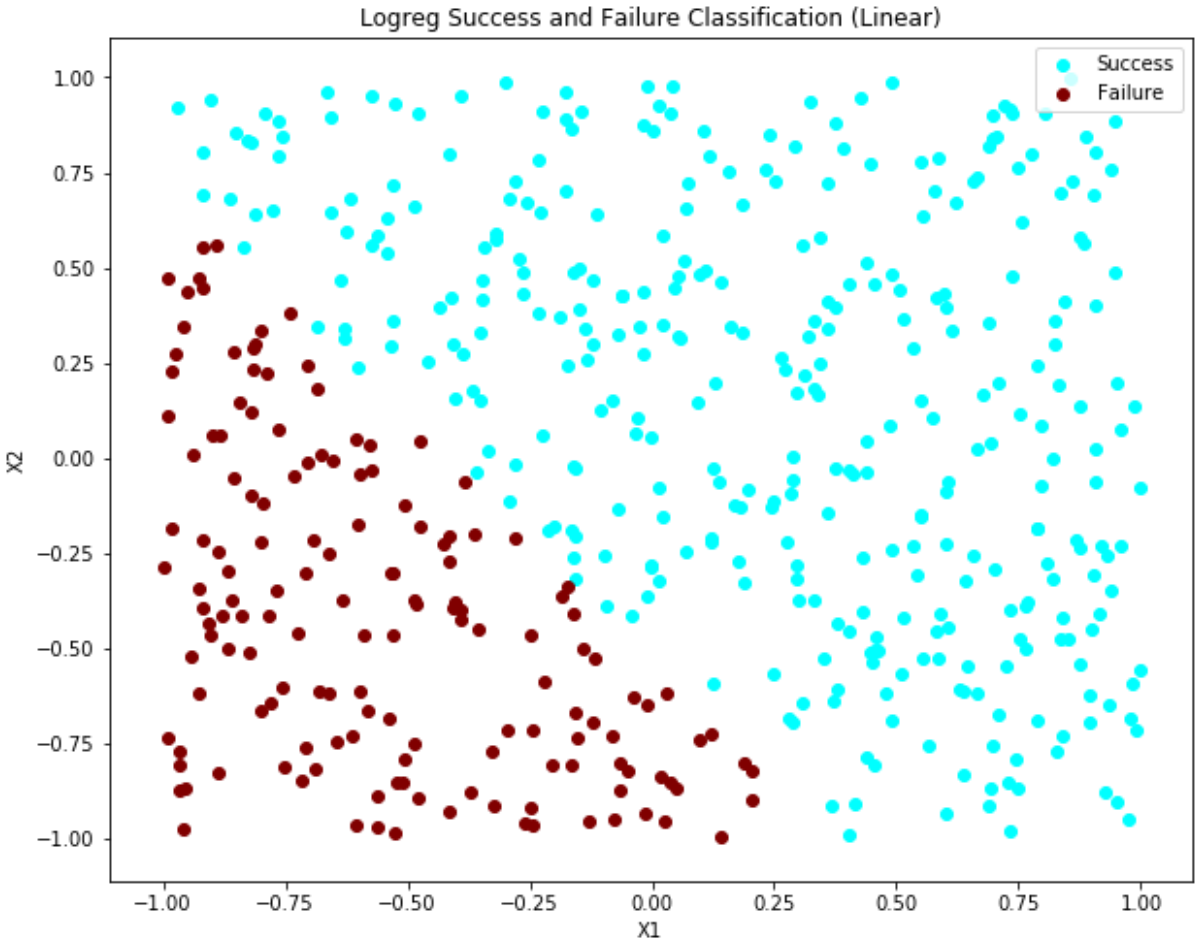
In [106]:
```
logreg = LogisticRegression().fit(x_linreg, y_input)
```

```
/Users/Sruti/opt/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:432: FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
```

1. (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 [115]:
```
logreg_predict = lr_linear.predict(x_linreg)
plt.figure(figsize=(10,8))
plt.scatter(x1[logreg_predict],x2[logreg_predict], color='aqua')
plt.scatter(x1[~logreg_predict],x2[~logreg_predict], color='maroon')
plt.xlabel('X1')
plt.ylabel('X2')
plt.legend(['Success','Failure'], loc=1)
plt.title('Logreg Success and Failure Classification (Linear)');
```



1. (5 points) Now fit a logistic regression model to the data, but this time using some non-linear function of both X_1 and X_2 as predictors (e.g. $X_1^2, X_1 \times X_2, \log(X_2)$, and so on).

In [156]:
```
interx = x1*x2
x1cube = x1**4
x2cube = x2**4
```

In [157]:
```
nonlin_3 = np.column_stack((interx, x1cube, x2cube))
```

```
logreg_nonlin = LogisticRegression().fit(nonlin_3, s_class)
```

```
/Users/Sruti/opt/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:432: FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
```

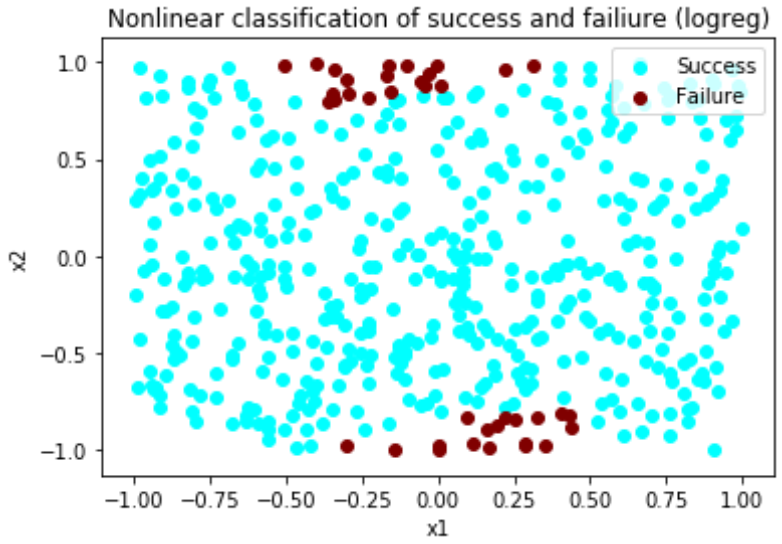
1. (5 points) Now, obtain a predicted class label for each observation based on the fitted model with non-linear transformations of the X features in the previous question. Plot the observations, colored according to the new predicted class labels from the non-linear model (the decision boundary should 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 [158]:
```
logreg_nonlin_pred = logreg_nonlin.predict(nonlin_3)
```

In [ ]:
```
plt.scatter(x1[logreg_nonlin_pred], x2[logreg_nonlin_pred], color = 'aqua')
plt.scatter(x1[~logreg_nonlin_pred], x2[~logreg_nonlin_pred], color = 'maroon')
```

In [ ]:
```
plt.xlabel('x1')
plt.ylabel('x2')
plt.title('Nonlinear classification of success and failiure (logreg)')
plt.legend(['Success', 'Failure'], loc=1);

#tried a few times until we got a more non-linear looking boundary
```

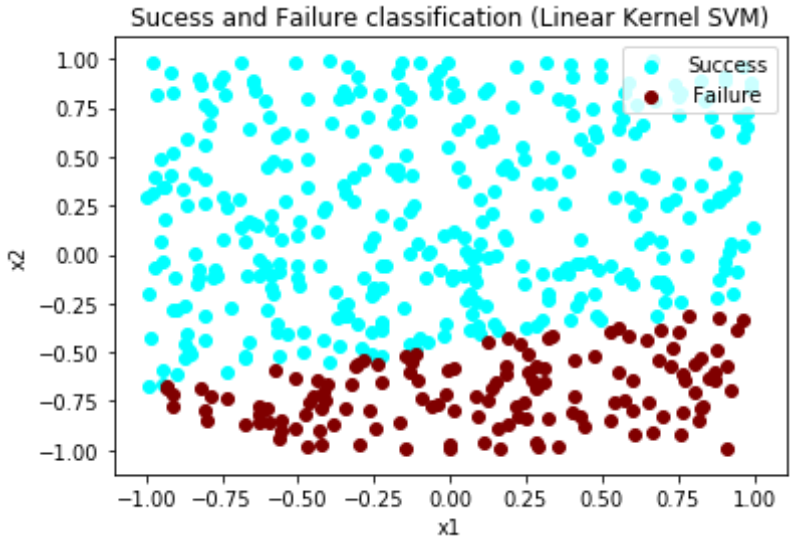


1. (5 points) Now, fit a support vector classifier (linear kernel) to the data with original X_1 and X_2 as predictors. Obtain a class prediction for each observation. Plot the observations, colored according to the predicted class labels.

In [ ]:
```
X_svm = np.column_stack((x1,x2))
X_svm = pd.DataFrame(X_svm)

svm_linear = svm.SVC(kernel='linear').fit(X_svm, s_class)
svm_lpred = svm_linear.predict(X_svm)
plt.scatter(x1[svm_lpred], x2[svm_lpred], color = 'aqua')
plt.scatter(x1[~svm_lpred], x2[~svm_lpred], color = 'maroon')
```

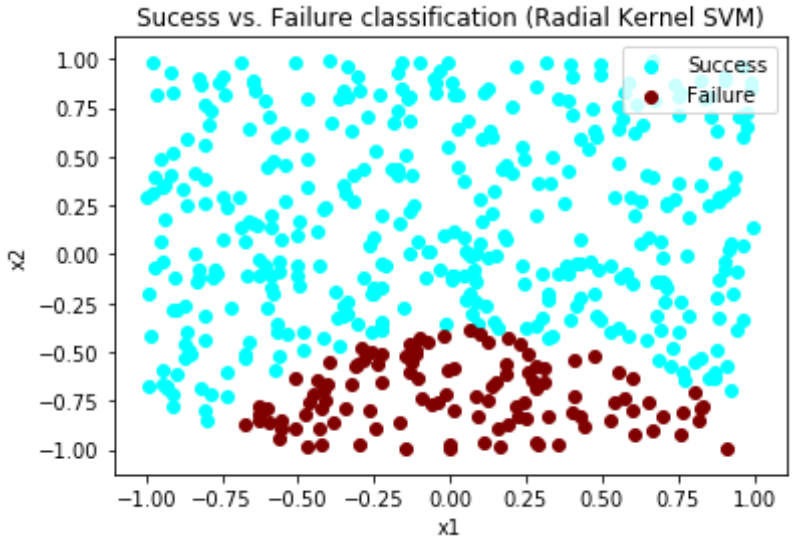
In [ ]:
```
plt.xlabel('x1')
plt.ylabel('x2')
plt.title('Sucess and Failure classification (Linear Kernel SVM)')
plt.legend(['Success', ' Failure'], loc=1);
```



1. (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 [ ]:
```
svm_r = svm.SVC().fit(X_svm, s_class)
svm_pred_r = svm_r.predict(X_svm)
plt.scatter(x1[svm_pred_r], x2[svm_pred_r], color = 'aqua')
plt.scatter(x1[~svm_pred_r], x2[~svm_pred_r], color = 'maroon')
```

In [ ]:
```
plt.xlabel('x1')
plt.ylabel('x2')
plt.title('Sucess vs. Failure classification (Radial Kernel SVM)')
plt.legend(['Success', 'Failure'], loc=1);
```



1. (5 points) Discuss your results and specifically the tradeoffs between estimating non-linear decision boundaries using these two different approaches.

In [ ]:

In [ ]:
```
lr = LogisticRegression().fit(X, success)

print('(linear) Logistic regression accuracy:', lr.score(X, success))
print('(nonlinear) Logistic regression accuracy):',lr_non_linear.score(
X_non_linear, success))
print('SVM linear kernel accuracy::',svm_linear.score(X, success))
print('SVM radial kernel accuracy:',svm_r.score(X, success))
```

Had some issues running the code a second time, here the output from the first run that worked:

- (linear) Logistic regression accuracy: 0.725
- (nonlinear) Logistic regression accuracy): 0.760
- SVM linear kernel accuracy:: 0.722
- SVM radial kernel accuracy: 0.764

Based on the graph outputs and the accuracy scores above, the SVM radial kernel performed the best. This makes sense, given that the underlying relationship of the data is non-linear, which explains why the linear approaches didn't do as well (and the nonlinear logistic regression did second best).

1. (5 points) Generate two-class data with $p = 2$ in such a way that the classes are just barely linearly separable.

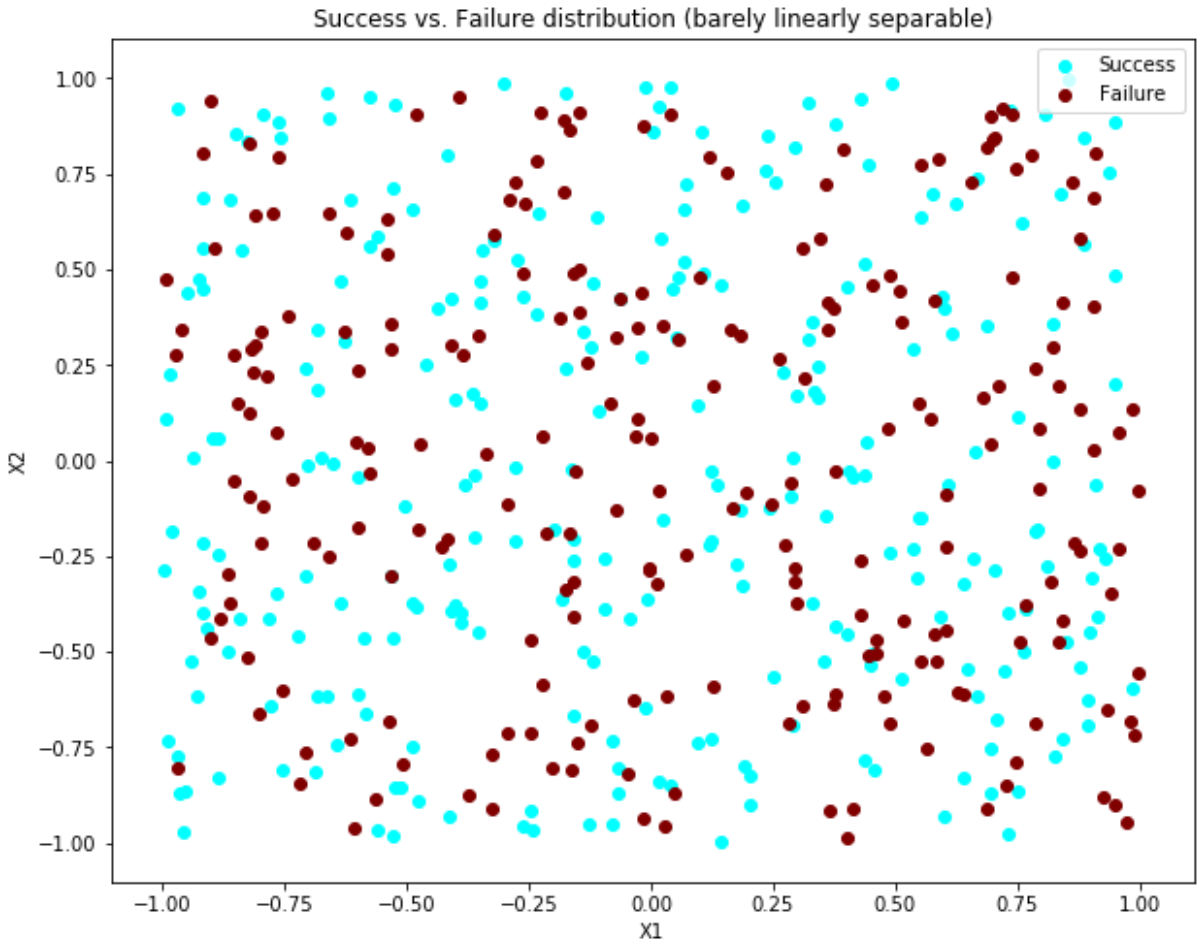
In [163]:
```
np.random.seed(666)
x1 = np.random.uniform(-1,1,500)
x2 = np.random.uniform(-1,1,500)
e = np.random.normal(0,0.6,500)

#generating y
y = x1 + x2 + e

#establishing success vs. failure
s_class = y >= 0.01 #barely discernable, margin
```

In [ ]:
```
plt.figure(figsize=(10,8))
plt.scatter(x1[s_class],x2[s_class], color='aqua')
plt.scatter(x1[~s_class],x2[~s_class], color='maroon')
```

In [165]:
```
plt.xlabel('X1')
plt.ylabel('X2')
plt.legend(['Success','Failure'], loc=1)
plt.title('Success vs. Failure distribution (barely linearly separabl
e)');
```



In [ ]:

1. (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 [170]:
```
X_tr = np.stack((x1[:200], x2[:200]), axis=1)
```

```
success_tr = s_class[:200]
Cost = [0.1, 1, 10, 100, 300]
for c in Cost:
    print(1 - accuracy_score(SVC(Cost=c, kernel='linear').fit(X_tr,succ
ess_train).predict(X_train), success_train))

#the output below corresponds to training error for c=0.1,
#c=1, c=10, c=100, and c=300 respectively
```

```
0.19499999999999995
0.17500000000000004
0.18000000000000005
0.18000000000000005
0.18000000000000005
```

In [171]:
```
for c in Cost:
    print(1 - np.mean(cross_val_score(SVC(Cost=c, kernel='linear'), X_t
rain, success_train, cv=10, scoring='accuracy')))


#the output below corresponds to the cv error for
#c=0.1,
#c=1, c=10, c=100, and c=300 respectively
```

```
0.20500000000000007
0.19499999999999995
0.18499999999999994
0.17999999999999994
0.17999999999999994
```

We can see that as C increases, the training error values decrease (they're actually the same for the last three C's in the sequence- 10, 100, and 300). Similiarly, the CV decreases as C increases, with the same value for C=100 and C=300

Seemingly 1 training error value per c value is venerated; same for cv error and test error.

Overall, I'd say accuracy seems to improve as cost increases (given that training error goes down -> accuracy is higher).

1. (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 [173]:
```
X_te = np.stack((x1[200:], x2[200:]), axis=1)
success_te = s_class[200:]
for c in Cost:
    print(1 - accuracy_score(SVC(Cost=c, kernel='linear').fit(X_tr,succ
ess_tr).predict(X_te), success_te))


#the output below corresponds to the test error for
#c=0.1,
#c=1, c=10, c=100, and c=300 respectively


    #lowest test errors correspond to c = 10, c=100, and c=300 (equal)
```

```
0.20333333333333337
0.20333333333333337
0.19666666666666666
0.19666666666666666
0.19666666666666666
```

In [ ]:
```

```

1. (5 points) Discuss your results.

As with the training error and cv values generated earlier, the test error decreases as C increases. Further, when the underlying relationship in our data is linear, a relatively lower C value might suffice

Application: Predicting attitudes towards racist college professors

1. (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 [179]:
```
gss_train = pd.read_csv('gss_train.csv')
```

In [180]:
```
gss_train.head(5)
```

Out[180]:

	age	attend	authoritarianism	black	born	childs	colath	colrac	colcom	colmil	...	partyid_
0	21	0	4	0	0	0	1	1	0	1	...	
1	42	0	4	0	0	2	0	1	1	0	...	
2	70	1	1	1	0	3	0	1	1	0	...	
3	35	3	2	0	0	2	0	1	0	1	...	
4	24	3	6	0	1	3	1	1	0	0	...	

5 rows × 56 columns

In [181]:
```
X = gss_train.drop(['colrac'], axis=1)
y = gss_train['colrac']
```

In [182]:
```
Cost = [0.2, 0.6, 1, 15, 20]
for c in Cost:
    print('The cv error for C = {} is'.format(c), 1 - np.mean(cross_val
_score(SVC(C=c, kernel='linear'), X, y, cv=10, scoring='accuracy')))
```

```
The cv error for C = 0.2 is 0.20460217099718536
The cv error for C = 0.6 is 0.20392655701859919
The cv error for C = 1 is 0.20595818047879422
The cv error for C = 15 is 0.20663385615446983
The cv error for C = 20 is 0.20663385615446983
```

The best tuning parameter is c=0.6, which corresponds to a cv value of .2039. However, there isn't much variation in the outputs as I vary c.

In [ ]:

In [ ]:

1. (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 [183]:
```
C = [0.1, 0.2, .5, 1, 5]
Gamma = ['scale', 'auto', 0.05]

for c in C:
    for gamma in Gamma: print('The cv error for C = {}'.format(c), 'gam
ma = {}'.format(gamma), 1 - np.mean(cross_val_score(SVC(C=c, kernel='rb
f', gamma = gamma), X, y, cv=10, scoring='accuracy')))
```

```
The cv error for C = 0.1 gamma = scale 0.3139772596867516
The cv error for C = 0.1 gamma = auto 0.47399732974996645
The cv error for C = 0.1 gamma = 0.05 0.4746776018588099
The cv error for C = 0.2 gamma = scale 0.2862741251044223
The cv error for C = 0.2 gamma = auto 0.38358277466618795
The cv error for C = 0.2 gamma = 0.05 0.4746776018588099
The cv error for C = 0.5 gamma = scale 0.26132823345191514
The cv error for C = 0.5 gamma = auto 0.28501868187870105
```

```
The cv error for C = 0.5 gamma = 0.05 0.4726505131346934
The cv error for C = 1 gamma = scale 0.24916570110721592
The cv error for C = 1 gamma = auto 0.27283329076234153
The cv error for C = 1 gamma = 0.05 0.3605590558370999
The cv error for C = 5 gamma = scale 0.223567054247783
The cv error for C = 5 gamma = auto 0.26411280819238636
The cv error for C = 5 gamma = 0.05 0.34841497092216167
```

The lowest cv error, .22356, corresponds to c=5 and gamma = auto (these parameters shape a fit that could most accurately predict colrac as a function of all available predictors). The gamma parameter tells us how near or far a sample's radius is (in terms of support vector). Higher gamma tends to spike the CV, based on the output. The radial kernel seems to do best with c values of 1 and 2, or decimal values that have gamma set to scale or auto.

In [185]:
```
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)

C = [0.1, 0.2, 0.5, 1]
Degree = [1, 3, 5, 10]
for c in C:
    for degree in Degree: print('The cv error for C = {}'.format(c), 'd
egree = {}'.format(degree), 1 - np.mean(cross_val_score(SVC(C=c, kernel
='poly', degree = degree), X, y, cv=10, scoring='accuracy')))
```

```
The cv error for C = 0.1 degree = 1 0.22890878825682281
The cv error for C = 0.1 degree = 3 0.2573102721952194
The cv error for C = 0.1 degree = 5 0.2701577347790196
The cv error for C = 0.1 degree = 10 0.29981055908668575
The cv error for C = 0.2 degree = 1 0.2106971647719491
The cv error for C = 0.2 degree = 3 0.2627250555582291
The cv error for C = 0.2 degree = 5 0.2701577347790196
The cv error for C = 0.2 degree = 10 0.29981055908668575
The cv error for C = 0.5 degree = 1 0.20594451457346952
The cv error for C = 0.5 degree = 3 0.26205385292154226
The cv error for C = 0.5 degree = 5 0.2701577347790196
The cv error for C = 0.5 degree = 10 0.29981055908668575
The cv error for C = 1 degree = 1 0.20189033712523652
The cv error for C = 1 degree = 3 0.26205385292154226
```

```
The cv error for C = 1 degree = 5 0.2701577347790196
The cv error for C = 1 degree = 10 0.29981055908668575
```

The lowest cv error is .2019, which corresponds to a degree of 1 and c of 1 (optimal value) = best fit for predicting colrac as a function of all available predictors. This also suggests that the data might have a linear pattern (degree = 1).

In [ ]: