Xiong_Yinjiang_HW6

March 4, 2020

[3]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn.svm import LinearSVC
from sklearn.model_selection import cross_val_score
```

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.

Generate 2 features, x1 and x2

[145]:
```
x1 = np.random.uniform(-1,1,100)
x2 = np.random.uniform(-1,1,100)
```

Generate $Y = X_1 + X_1^2 + X_2 + X_2^2 + \epsilon$, where $\epsilon \sim N(\mu = 0, \sigma^2 = 0.25)$.

[146]:
```
error = np.random.normal(0,0.5,100)
y = x1 + x1 * x1 + x2 + x2 * x2 + error
```

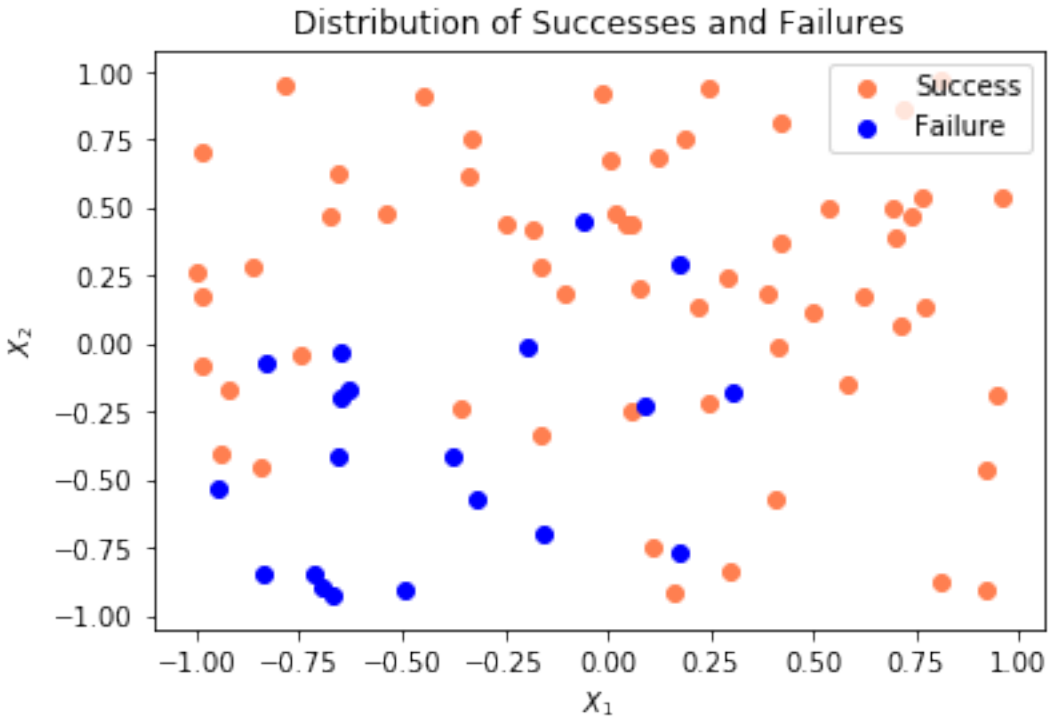
Define results

[147]:
```
train_result = y[:80] >= 0
test_result = y[80:] >= 0
```

Visualize

[148]:
```
x1_train = x1[:80]
x2_train = x2[:80]
x1_test = x1[80:]
x2_test = x2[80:]
plt.scatter(x1_train[train_result],x2_train[train_result], color='coral')
plt.scatter(x1_train[~train_result],x2_train[~train_result], color='blue')
plt.xlabel('$X_1$')
```

```
plt.ylabel('$X_2$')
plt.legend(['Success','Failure'], loc=1)
plt.title('Distribution of Successes and Failures');
```



Train svm and svc

[149]:
```
svm = SVC(kernel='rbf', gamma='scale')
svc = SVC(kernel='linear')
X_train = np.stack((x1_train, x2_train), axis=1)
svm.fit(X_train, train_result)
svc.fit(X_train, train_result)
```

[149]:
```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='auto_deprecated',
    kernel='linear', max_iter=-1, probability=False, random_state=None,
    shrinking=True, tol=0.001, verbose=False)
```

Predict training set

[150]:
```
print('The accuracy for svm using radial kernel in training set is',␣
 ↪accuracy_score(svm.predict(X_train), train_result))
print('The accuracy for svc using linear kernel in training set is',␣
 ↪accuracy_score(svc.predict(X_train), train_result))
```

```
The accuracy for svm using radial kernel in training set is 0.85
The accuracy for svc using linear kernel in training set is 0.8
```

Predict test set

[151]:
```
X_test = np.stack((x1_test, x2_test), axis=1)
```

[152]:
```
print('The accuracy for svm using radial kernel in test set is',␣
 ↪accuracy_score(svm.predict(X_test), test_result))
print('The accuracy for svc using linear kernel in test set is',␣
 ↪accuracy_score(svc.predict(X_test), test_result))
```

```
The accuracy for svm using radial kernel in test set is 0.85
The accuracy for svc using linear kernel in test set is 0.8
```

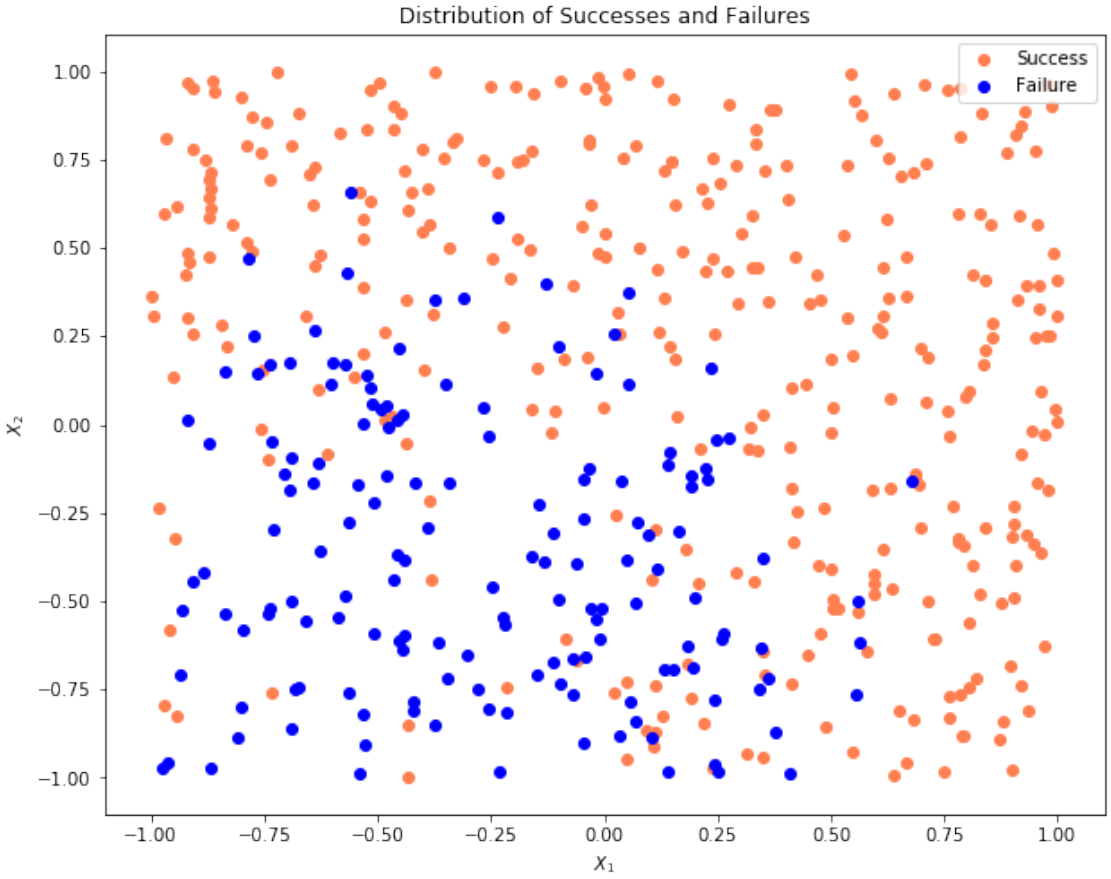
As the result suggests, when the separation is clear but not linear, svm with radial kernal consistently outperforms svc with linear kernel in both training and test sets.

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.

2.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.

3.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.

[86]:
```
x1 = np.random.uniform(-1,1,500)
x2 = np.random.uniform(-1,1,500)
error = np.random.normal(0,2,500)
y = x1 + x1 * x1 + x2 + x2 * x2 + error
success = y >= 0
plt.figure(figsize=(10,8))
plt.scatter(x1[success],x2[success], color='coral')
plt.scatter(x1[~success],x2[~success], color='blue')
plt.xlabel('$X_1$')
plt.ylabel('$X_2$')
plt.legend(['Success','Failure'], loc=1)
plt.title('Distribution of Successes and Failures');
```

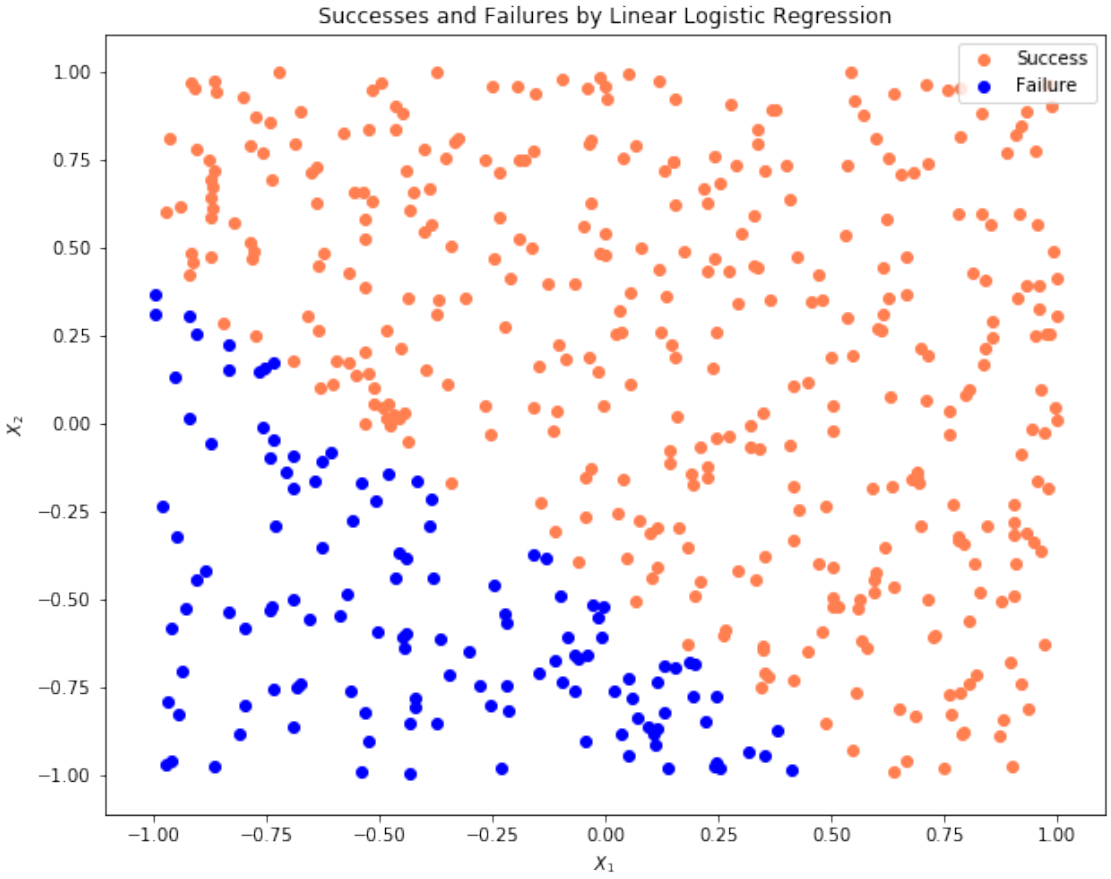


4.Fit a logistic regression model to the data, using X_1 and X_2 as predictors.

[87]:
```
X = np.stack((x1, x2), axis=1)
X_linear = np.stack((x1, x2), axis=1)
lr_linear = LogisticRegression().fit(X_linear, success)
```

5.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).

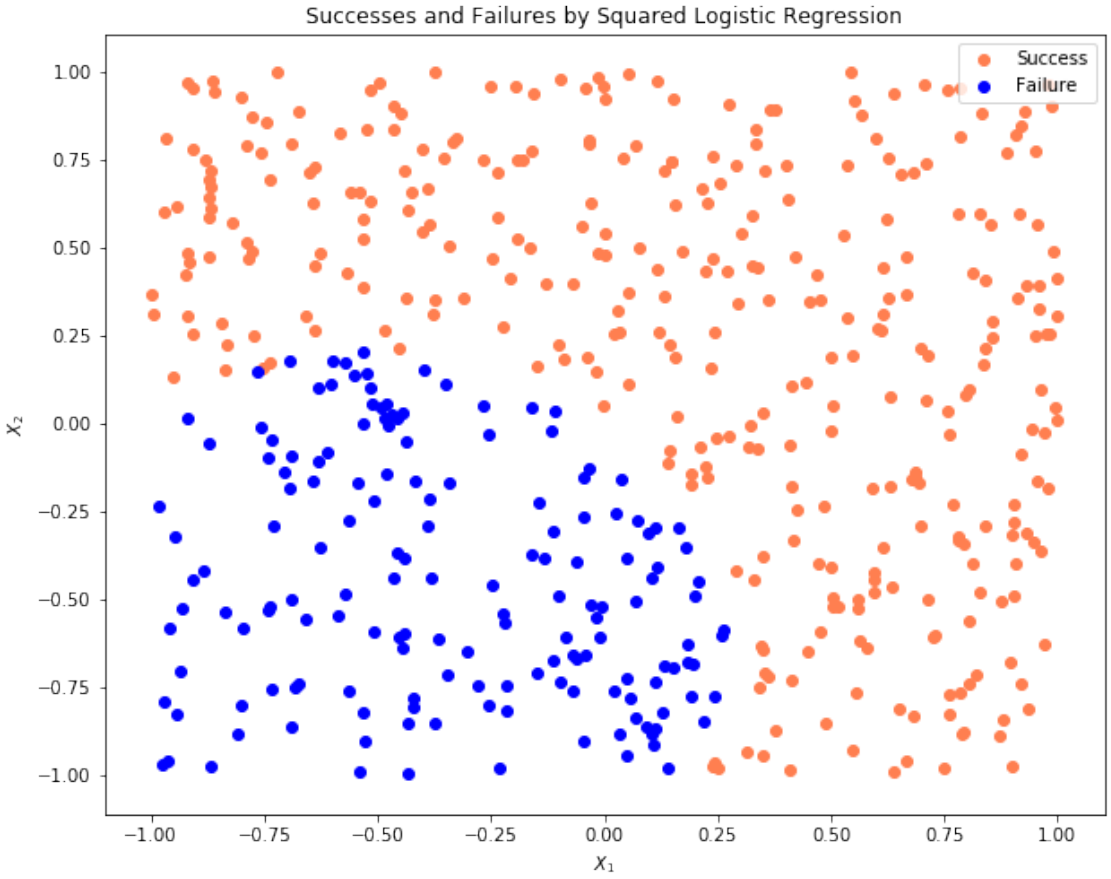
[88]:
```
lr_linear_predict = lr_linear.predict(X_linear)
plt.figure(figsize=(10,8))
plt.scatter(x1[lr_linear_predict],x2[lr_linear_predict], color='coral')
plt.scatter(x1[~lr_linear_predict],x2[~lr_linear_predict], color='blue')
plt.xlabel('$X_1$')
plt.ylabel('$X_2$')
plt.legend(['Success','Failure'], loc=1)
plt.title('Successes and Failures by Linear Logistic Regression');
```



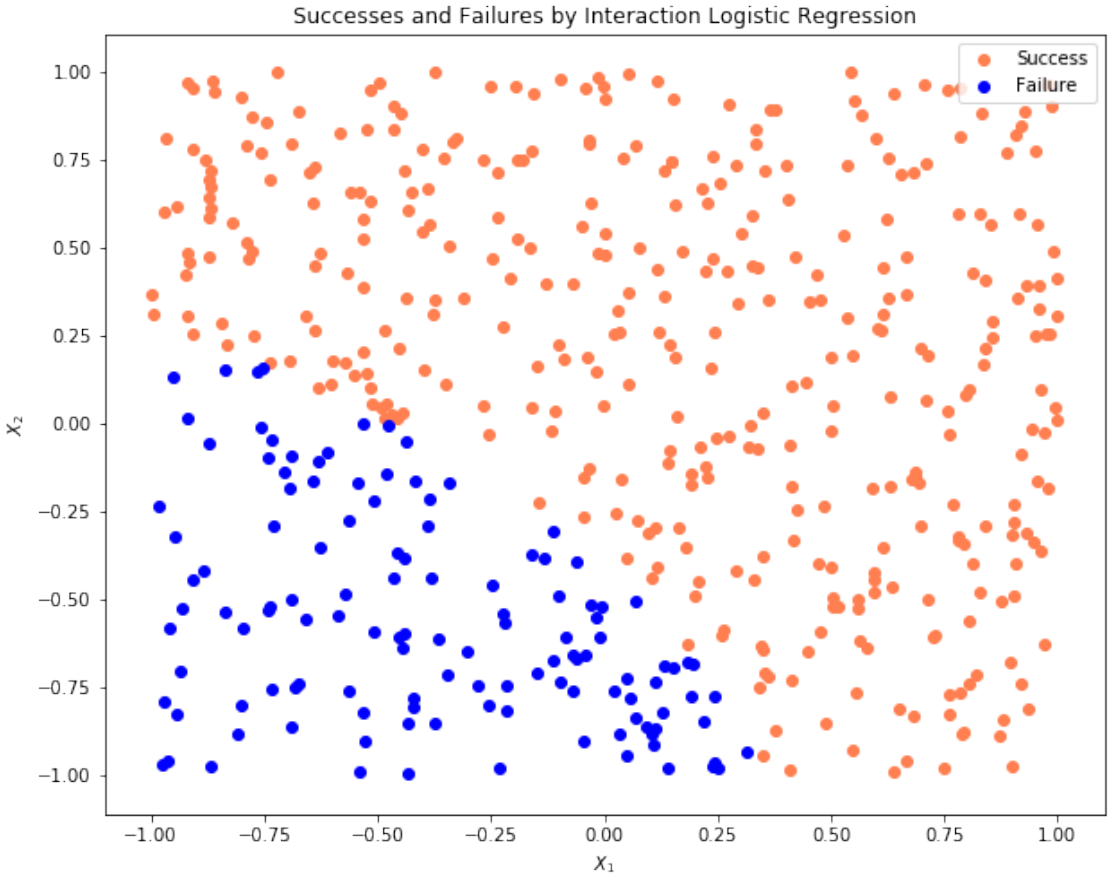
6.Now fit a logistic regression model to the data, but this time using some non-linear function of both X_1 and X_2 as predictors (e.g. $X_1^2, X_1 \times X_2, \log(X_2)$, and so on).

[89]:
```
X_square = np.stack((x1, x2, x1**2, x2**2), axis=1)
X_inter = np.stack((x1, x2, x1*x2), axis=1)
lr_square = LogisticRegression().fit(X_square, success)
lr_inter = LogisticRegression().fit(X_inter, success)
```

[91]:
```
lr_square_predict = lr_square.predict(X_square)
plt.figure(figsize=(10,8))
plt.scatter(x1[lr_square_predict],x2[lr_square_predict], color='coral')
plt.scatter(x1[~lr_square_predict],x2[~lr_square_predict], color='blue')
plt.xlabel('$X_1$')
plt.ylabel('$X_2$')
plt.legend(['Success','Failure'], loc=1)
plt.title('Successes and Failures by Squared Logistic Regression');
```

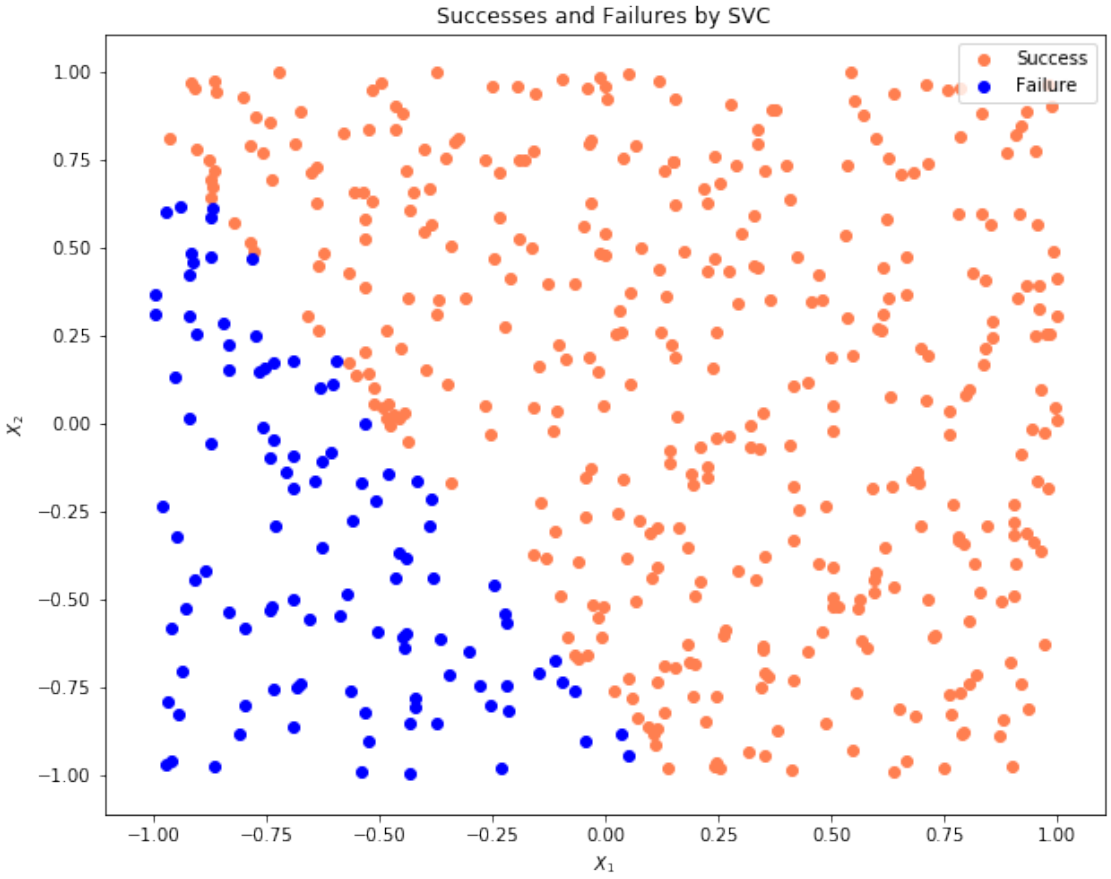


[92]:
```
lr_inter_predict = lr_inter.predict(X_inter)
plt.figure(figsize=(10,8))
plt.scatter(x1[lr_inter_predict],x2[lr_inter_predict], color='coral')
plt.scatter(x1[~lr_inter_predict],x2[~lr_inter_predict], color='blue')
plt.xlabel('$X_1$')
plt.ylabel('$X_2$')
plt.legend(['Success','Failure'], loc=1)
plt.title('Successes and Failures by Interaction Logistic Regression');
```



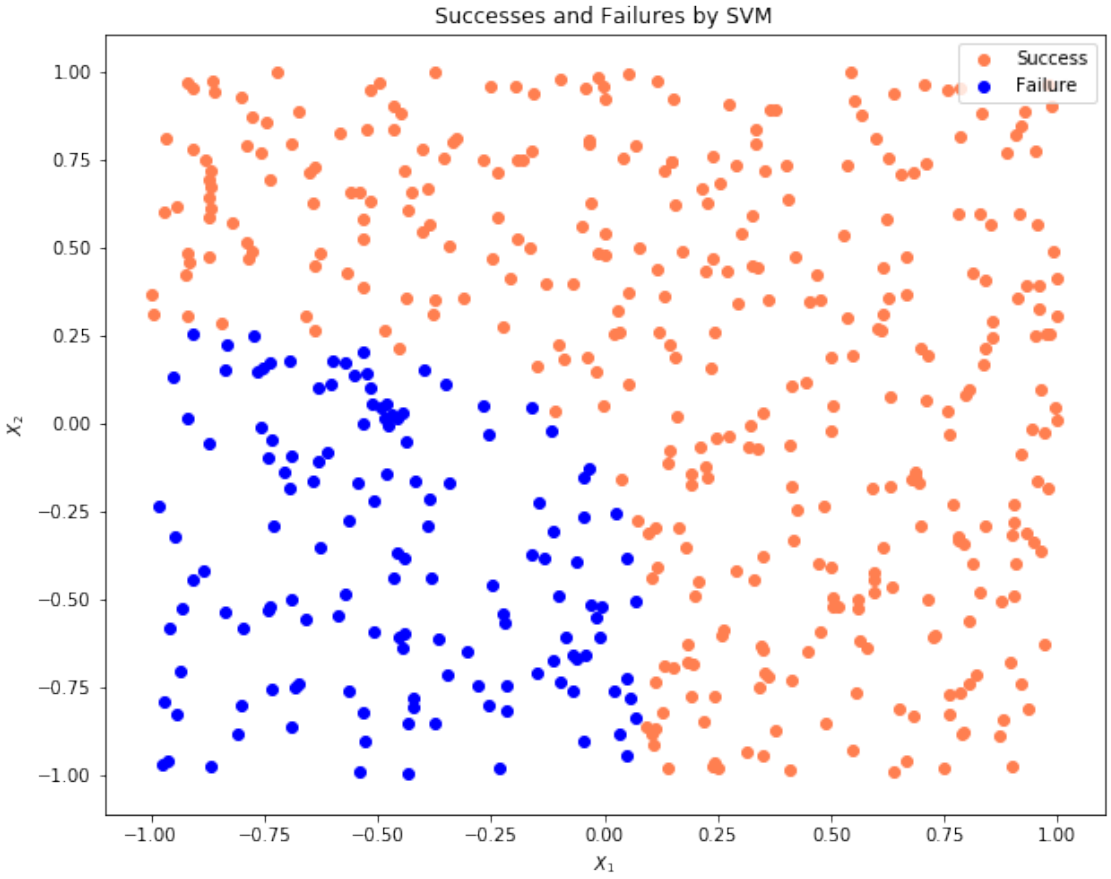
8.Now, fit a support vector classifier (linear kernel) to the data with original X_1 and X_2 as predictors. Obtain a class prediction for each observation. Plot the observations, colored according to the predicted class labels.

[93]:
```
svc_predict = svc.predict(X)
plt.figure(figsize=(10,8))
plt.scatter(x1[svc_predict],x2[svc_predict], color='coral')
plt.scatter(x1[~svc_predict],x2[~svc_predict], color='blue')
plt.xlabel('$X_1$')
plt.ylabel('$X_2$')
plt.legend(['Success','Failure'], loc=1)
plt.title('Successes and Failures by SVC');
```



9.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.

[94]:
```
svm_predict = svm.predict(X)
plt.figure(figsize=(10,8))
plt.scatter(x1[svm_predict],x2[svm_predict], color='coral')
plt.scatter(x1[~svm_predict],x2[~svm_predict], color='blue')
plt.xlabel('$X_1$')
plt.ylabel('$X_2$')
plt.legend(['Success','Failure'], loc=1)
plt.title('Successes and Failures by SVM');
```



10.Discuss your results and specifically the tradeoffs between estimating non-linear decision boundaries using these two different approaches.

[95]:
```
print('Accuracy for logistic regression(linear):
 ↪',accuracy_score(lr_linear_predict, success))
print('Accuracy for logistic regression(squared):
 ↪',accuracy_score(lr_square_predict, success))
print('Accuracy for logistic regression(interaction):
 ↪',accuracy_score(lr_inter_predict, success))
print('Accuracy for svc:',accuracy_score(svc_predict, success))
print('Accuracy for svm:',accuracy_score(svm_predict, success))
```

```
Accuracy for logistic regression(linear): 0.794
Accuracy for logistic regression(squared): 0.84
Accuracy for logistic regression(interaction): 0.81
Accuracy for svc: 0.774
Accuracy for svm: 0.834
```

Accuracy wise, we see the squared logistic regression performs as well as svm, and this is because we train the true model. If we don't know the true model, logistic regression could lose to svm

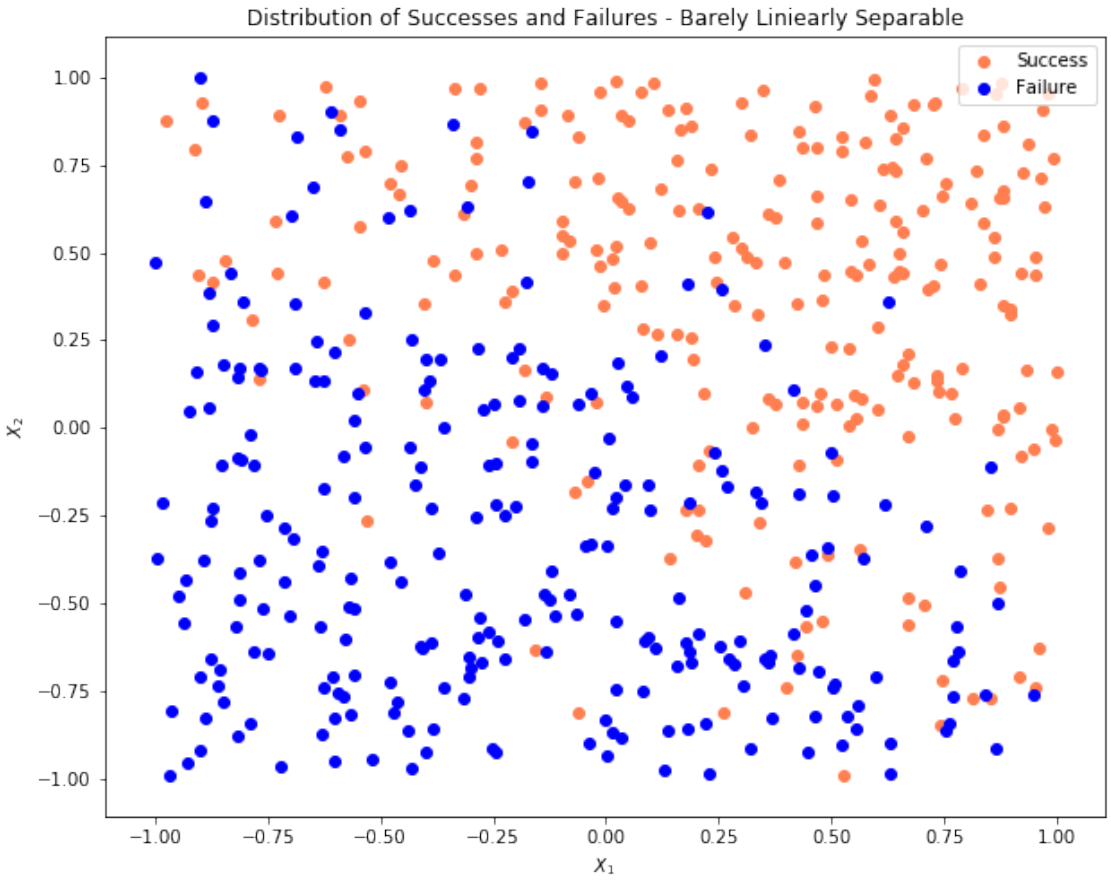
in accuracy. On the other hand, logistic regression is more interpretable, as we can report the probability of each point, whereas svm only returns the result.

Tuning cost

In class we learned that in the case of data that is just barely linearly separable, a support vector classifier with a small value of cost that misclassifies a couple of training observations may perform better on test data than one with a huge value of cost that does not misclassify any training observations. You will now investigate that claim.

11.Generate two-class data with $p = 2$ in such a way that the classes are just barely linearly separable.

[170]:
```
x1 = np.random.uniform(-1,1,500)
x2 = np.random.uniform(-1,1,500)
error = np.random.normal(0,0.6,500)
y = x1 + x2 + error
success = y >= 0
plt.figure(figsize=(10,8))
plt.scatter(x1[success],x2[success], color='coral')
plt.scatter(x1[~success],x2[~success], color='blue')
plt.xlabel('$X_1$')
plt.ylabel('$X_2$')
plt.legend(['Success','Failure'], loc=1)
plt.title('Distribution of Successes and Failures - Barely Liniearly Separable');
```



12.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?

[179]:
```
# training error
X_train = np.stack((x1[:400], x2[:400]), axis=1)
success_train = success[:400]
C = [0.1, 1, 10, 100, 300]
for c in C:
    print('Training error for C = {} is'.format(c),
          1 - accuracy_score(SVC(C=c, kernel='linear').fit(X_train,success_train).
 ↪predict(X_train), success_train))
```

```
Training error for C = 0.1 is 0.15749999999999997
Training error for C = 1 is 0.16249999999999998
Training error for C = 10 is 0.15249999999999997
Training error for C = 100 is 0.15249999999999997
Training error for C = 300 is 0.15249999999999997
```

[180]:
```
# cv error
for c in C:
    print('The cv error for C = {} is'.format(c),
          1 - np.mean(cross_val_score(SVC(C=c, kernel='linear'), X_train,␣
 ↪success_train, cv=10, scoring='accuracy')))
```

```
The cv error for C = 0.1 is 0.1576626016260162
The cv error for C = 1 is 0.16022670419011875
The cv error for C = 10 is 0.1551016260162601
The cv error for C = 100 is 0.1551016260162601
The cv error for C = 300 is 0.1551016260162601
```

We can see for training error, as c increases, the model is more flexible and the training error decreases. But for cv error, as c increases, the model overfits and cv error increases.

13.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?

[181]:
```
# test error
X_test = np.stack((x1[400:], x2[400:]), axis=1)
success_test = success[400:]
for c in C:
    print('Test error for C = {} is'.format(c),
          1 - accuracy_score(SVC(C=c, kernel='linear').fit(X_train,success_train).
 ↪predict(X_test), success_test))
```

```
Test error for C = 0.1 is 0.18000000000000005
Test error for C = 1 is 0.18000000000000005
Test error for C = 10 is 0.18000000000000005
Test error for C = 100 is 0.18000000000000005
Test error for C = 300 is 0.18000000000000005
```

Test error is consistent with CV result in that when C=10, the error is the lowest.

15.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.

[4]:
```
gss_train = pd.read_csv('gss_train.csv')
```

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

[5]:

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

```
   colmil  ...  partyid_3_Ind  partyid_3_Rep  relig_CATHOLIC  relig_NONE  \
0       1  ...              1              0               1           0
1       0  ...              1              0               0           0
2       0  ...              0              0               0           0
3       1  ...              1              0               0           0
4       0  ...              1              0               1           0

   relig_other  social_cons3_Mod  social_cons3_Conserv  spend3_Mod  \
0            0                 1                     0           0
1            0                 0                     0           1
2            0                 0                     0           0
3            1                 0                     0           0
4            0                 1                     0           0

   spend3_Liberal  zodiac_other
0               0             1
1               0             1
2               0             1
3               1             1
4               0             1

[5 rows x 56 columns]
```

[6]:
```
X = gss_train.drop(['colrac'], axis=1)
y = gss_train['colrac']
```

[7]:
```
C = [0.1, 0.5, 1, 10, 15]
for c in C:
    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 = 0.1 is 0.2039447576600022
The cv error for C = 0.5 is 0.20325088134292357
The cv error for C = 1 is 0.20595818047879422
The cv error for C = 10 is 0.2073141282633133
The cv error for C = 15 is 0.20663385615446983
```

The cv result shows that when C = 0.5, the cv error is the lowest.

16.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.).

[20]:
```
# radial kernal
C = [0.1, 0.5, 1, 5]
Gamma = ['scale', 'auto', 0.05, 0.1]
```

```
for c in C:
    for gamma in Gamma:
        print('The cv error for C = {}'.format(c), 'gamma = {}'.format(gamma),
              1 - np.mean(cross_val_score(SVC(C=c, kernel='rbf', 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 = 0.1 gamma = 0.1 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 = 0.5 gamma = 0.1 0.4746776018588099
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 = 1 gamma = 0.1 0.4699432139988228
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 cv error for C = 5 gamma = 0.1 0.46187126038516246
```

For radial kernal svm, the optimal C is 5 or greater (computationally taxing for C > 5) and gamma as scale. Scale means that it uses 1 / (n_features * X.var()) as value of gamma. We do see a pattern that as C gets larger, the result gets better, showing the data require a more flexible model. The cv error gets worse as gamma gets larger.

[22]:
```
# poly kernal
C = [0.1, 0.5, 1, 5]
Degree = [1, 2, 3, 4]
for c in C:
    for degree in Degree:
        print('The cv error for C = {}'.format(c), 'degree = {}'.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 = 2 0.2086472172761722
The cv error for C = 0.1 degree = 3 0.2573102721952194
The cv error for C = 0.1 degree = 4 0.2627205208221508
The cv error for C = 0.5 degree = 1 0.20594451457346952
The cv error for C = 0.5 degree = 2 0.22558072385493289
The cv error for C = 0.5 degree = 3 0.26205385292154226
The cv error for C = 0.5 degree = 4 0.2627205208221508
The cv error for C = 1 degree = 1 0.20189033712523652
The cv error for C = 1 degree = 2 0.2255988011021568
The cv error for C = 1 degree = 3 0.26205385292154226
```

```
The cv error for C = 1 degree = 4 0.2627205208221508
The cv error for C = 5 degree = 1 0.20190853776663942
The cv error for C = 5 degree = 2 0.2397333883671371
The cv error for C = 5 degree = 3 0.26205385292154226
The cv error for C = 5 degree = 4 0.2627205208221508
```

The optimal poly svm is when C = 1 and degree = 1 and is better than the best model achieved by radial. This shows that the data have a linear pattern.

[ ]: