Jiang_Luying_HW6

March 8, 2020

0.1 Conceptual exercises

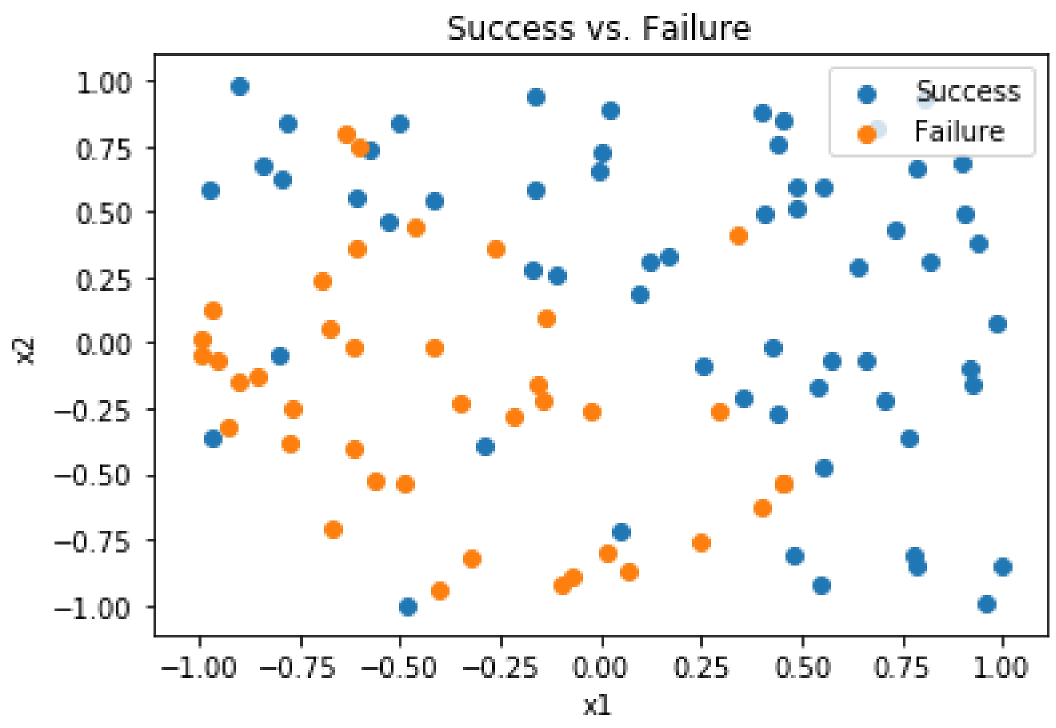
0.1.1 Non-linear separation

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.

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import sklearn
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
import math
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
warnings.simplefilter(action='ignore', category=DeprecationWarning)
```

[2]:
```
np.random.seed(666)
x1 = np.random.uniform(-1,1,100)
x2 = np.random.uniform(-1,1,100)
error = np.random.normal(0,0.5,100)
y = x1 + x1 ** 2 + x2 + x2 ** 2 + error
y_prob = np.exp(y) / (1 + np.exp(y))
success = y_prob >= 0.6
failure = y_prob < 0.6
plt.scatter(x1[success],x2[success])
plt.scatter(x1[failure],x2[failure])
plt.xlabel('x1')
plt.ylabel('x2')
plt.title('Success vs. Failure')
plt.legend(['Success', 'Failure'], loc = 1)
```

[2]:
```
<matplotlib.legend.Legend at 0x1a1941f4e0>
```



[3]:
```
X = np.column_stack((x1, x2))
X_train,X_test,y_train,y_test = train_test_split(X,success,test_size = 0.2)
```

[4]:
```
svm_radial = SVC().fit(X_train, y_train)
svm_linear = SVC(kernel='linear').fit(X_train, y_train)
```

[5]:
```
print('Training Error:')
print('Radial Kernel Error Rate:', round(1 - svm_radial.score(X_train,y_train),␣
 ↪5))
print('Linear Kernel Error Rate:', round(1 - svm_linear.score(X_train,y_train),␣
 ↪5))
```

```
Training Error:
Radial Kernel Error Rate: 0.175
Linear Kernel Error Rate: 0.225
```

[6]:
```
print('Testing Error:')
print('Radial Kernel Error Rate:', round(1 - svm_radial.score(X_test, y_test),␣
 ↪5))
print('Linear Kernel Error Rate:', round(1 - svm_linear.score(X_test, y_test),␣
 ↪5))
```

```
Testing Error:
Radial Kernel Error Rate: 0.05
Linear Kernel Error Rate: 0.15
```

From above, we can see that SVM with radial kernel has lower error rate than SVM with linear kernel on both the training and testing data set.

0.1.2 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.

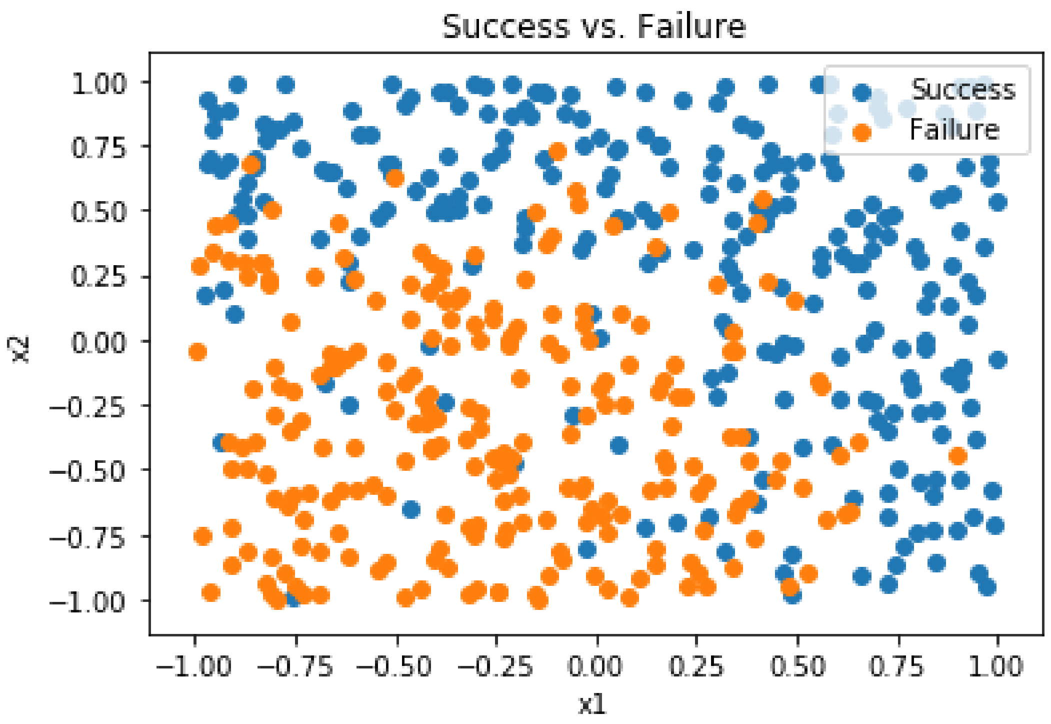
2.Generate a data set with n = 500 and p = 2, such that the observations belong to two classes with some overlapping, non-linear boundary between them.

[7]:
```
x1 = np.random.uniform(-1,1,500)
x2 = np.random.uniform(-1,1,500)
error = np.random.normal(0,0.5,500)
y = x1 + x1 ** 2 + x2 + x2 ** 2 + error
y_prob = np.exp(y)/(1 + np.exp(y))
success = y_prob >= 0.6
failure = y_prob < 0.6
```

3.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.

[8]:
```
plt.scatter(x1[success],x2[success])
plt.scatter(x1[failure],x2[failure])
plt.xlabel('x1')
plt.ylabel('x2')
plt.title('Success vs. Failure')
plt.legend(['Success', 'Failure'], loc = 1)
```

[8]:
```
<matplotlib.legend.Legend at 0x1a194a99e8>
```



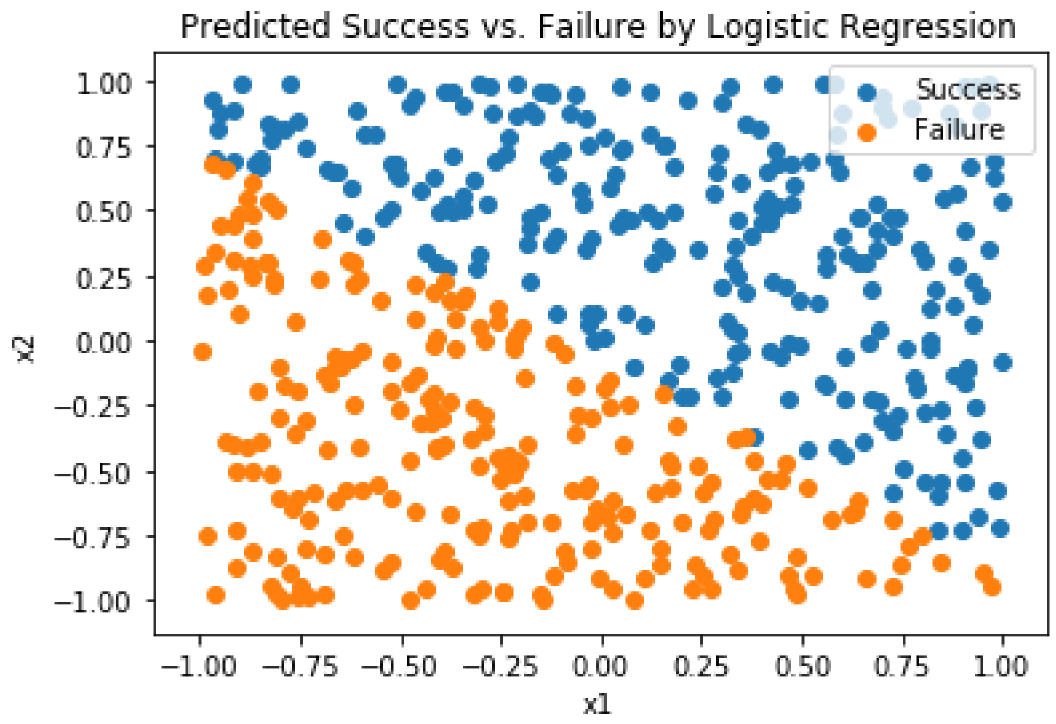
4.Fit a logistic regression model to the data, using X1 and X2 as predictors.

[9]:
```
X = np.column_stack((x1,x2))
LR = LogisticRegression().fit(X,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).

[10]:
```
y_pred = LR.predict(X)
plt.scatter(x1[y_pred],x2[y_pred])
plt.scatter(x1[~y_pred],x2[~y_pred])
plt.xlabel('x1')
plt.ylabel('x2')
plt.title('Predicted Success vs. Failure by Logistic Regression')
plt.legend(['Success', 'Failure'], loc = 1)
```

[10]:
```
<matplotlib.legend.Legend at 0x1a19626908>
```



The decision boundary looks very linear (with a negative slope).

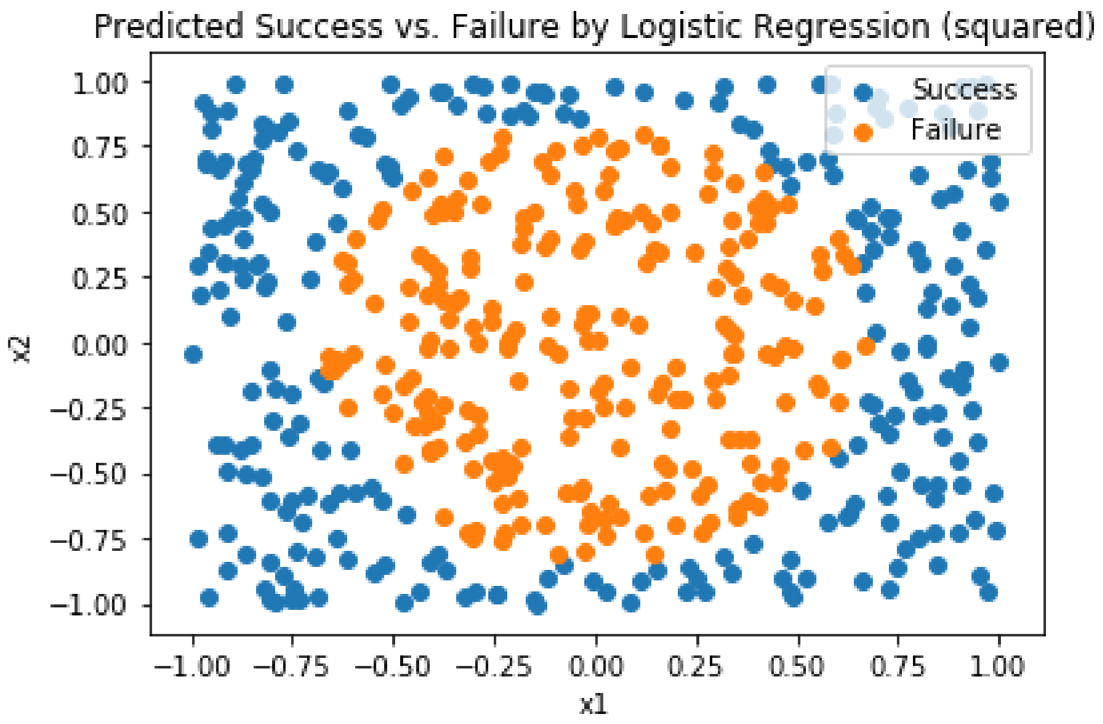
6.Now fit a logistic regression model to the data, but this time using some non-linear function of both X1 and X2 as predictors (e.g. X2 1 ,X1 Œ X2, log(X2), and so on).

[11]:
```
LR_squared = LogisticRegression().fit(X**2,success)
```

7.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.

[12]:
```
y_pred = LR_squared.predict(X**2)
plt.scatter(x1[y_pred],x2[y_pred])
plt.scatter(x1[~y_pred],x2[~y_pred])
plt.xlabel('x1')
plt.ylabel('x2')
plt.title('Predicted Success vs. Failure by Logistic Regression (squared)')
plt.legend(['Success', 'Failure'], loc = 1)
```

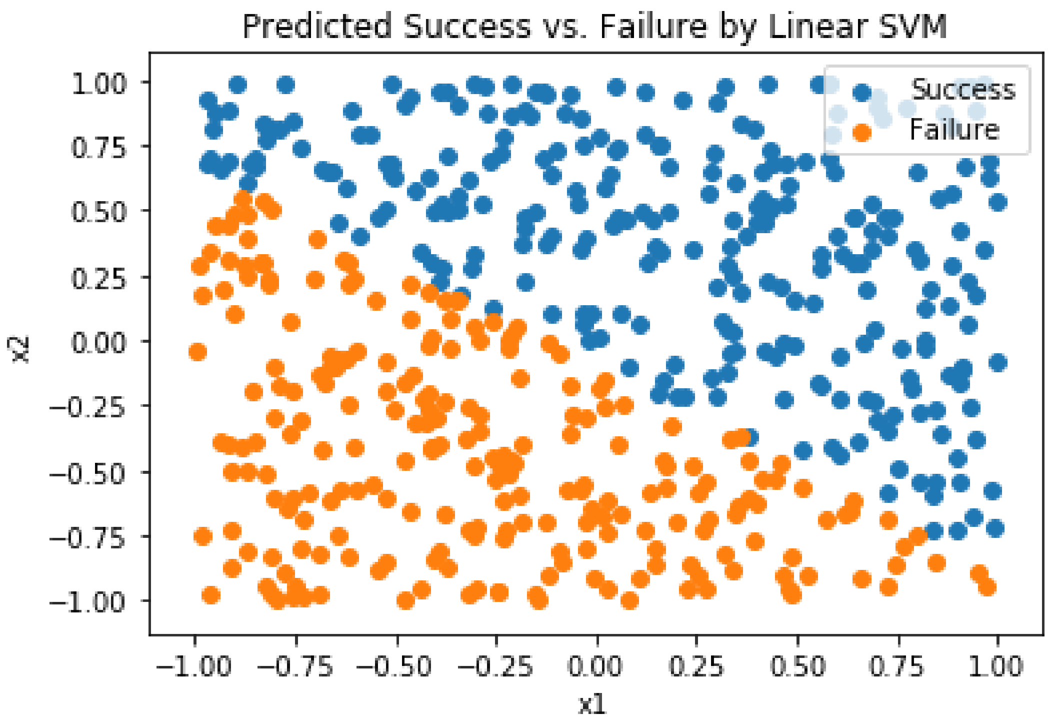
[12]:
```
<matplotlib.legend.Legend at 0x1a194e5668>
```



8.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.

[13]:
```
svm_linear = SVC(kernel='linear').fit(X, success)
y_pred = svm_linear.predict(X)
plt.scatter(x1[y_pred],x2[y_pred])
plt.scatter(x1[~y_pred],x2[~y_pred])
plt.xlabel('x1')
plt.ylabel('x2')
plt.title('Predicted Success vs. Failure by Linear SVM')
plt.legend(['Success', 'Failure'], loc = 1)
```

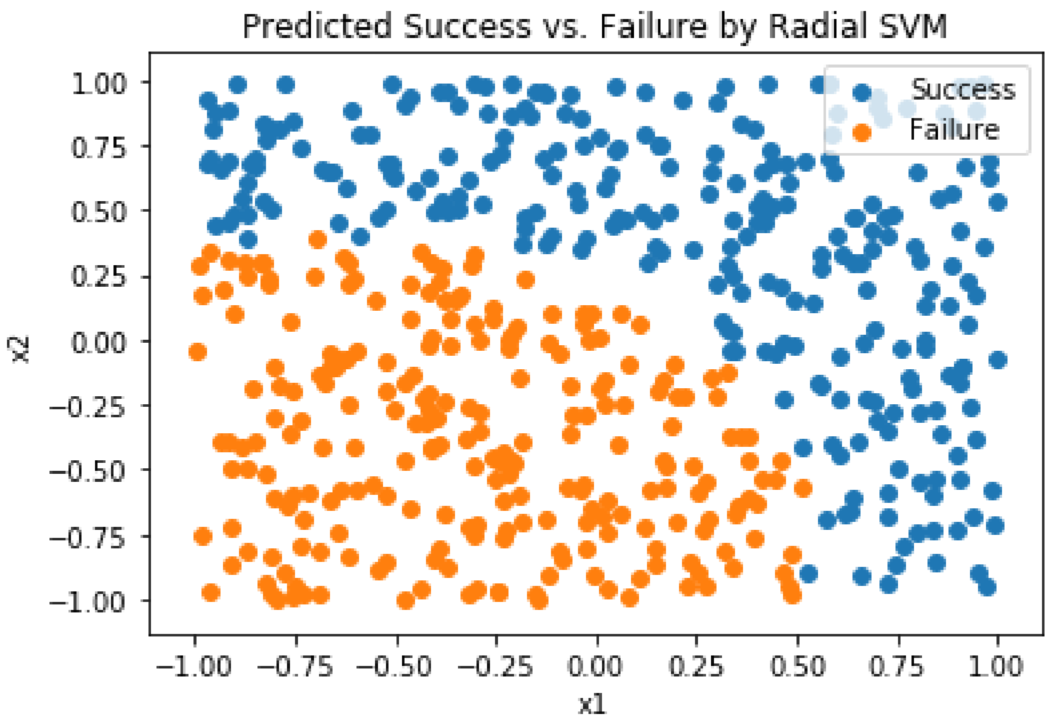
[13]:
```
<matplotlib.legend.Legend at 0x1a19607978>
```



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.

[14]:
```
svm_nonlinear = SVC().fit(X, success)
y_pred = svm_nonlinear.predict(X)
plt.scatter(x1[y_pred],x2[y_pred])
plt.scatter(x1[~y_pred],x2[~y_pred])
plt.xlabel('x1')
plt.ylabel('x2')
plt.title('Predicted Success vs. Failure by Radial SVM')
plt.legend(['Success', 'Failure'], loc = 1)
```

[14]:
```
<matplotlib.legend.Legend at 0x1a198ef080>
```



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

[15]:
```
print('Logistic Regression (Linear) Accuracy:', LR.score(X, success))
print('Logistic Regression (Non-linear) Accuracy:',
      LR_squared.score(X, success))
print('SVM (Linear) Accuracy:', svm_linear.score(X, success))
print('SVM (Non-linear) Accuracy:', svm_nonlinear.score(X, success))
```

```
Logistic Regression (Linear) Accuracy: 0.83
Logistic Regression (Non-linear) Accuracy: 0.71
SVM (Linear) Accuracy: 0.828
SVM (Non-linear) Accuracy: 0.87
```

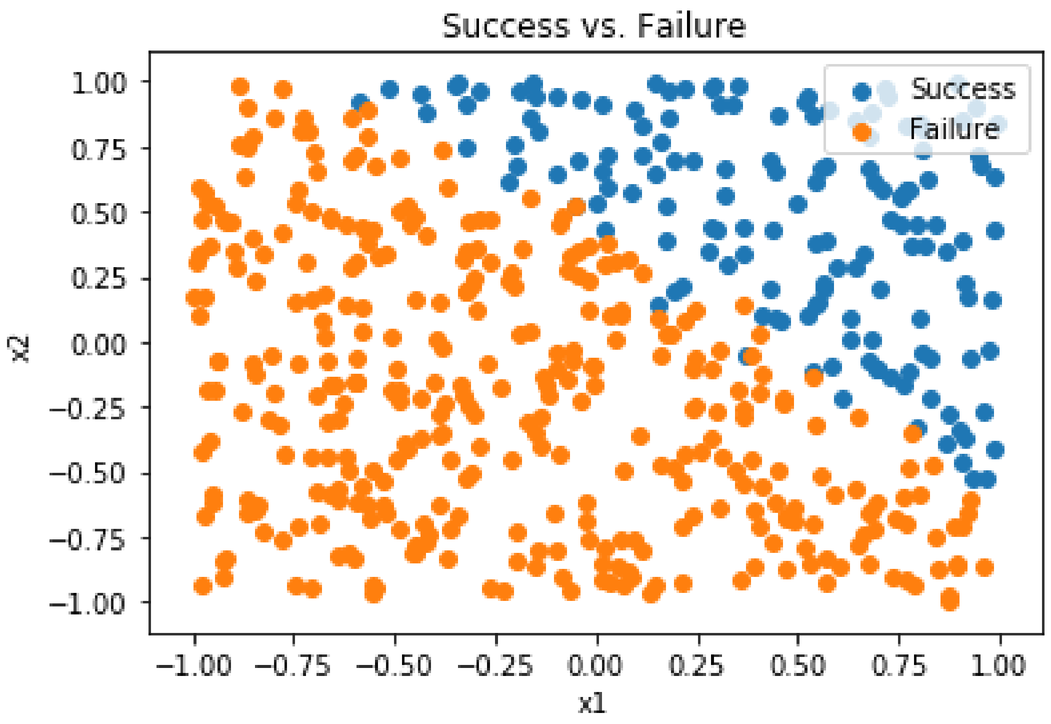
From above, we can see that the SVM with radial kernel performed better than the other three classifiers. This result is expected in the case of non-linear relationships. The prediction of non-linear logistic regression model has the lowest accuracy and we can see from the graphs, it does not capture the same shape as other classifiers. Linear Logistic Regression and the linear SVM have similar accuracy scores. Because SVM has built-in non-linear kernel, SVM seems to be a better choice when we want to estimate non-linear boundaries.

0.1.3 Tuning cost

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

[16]:
```
x1 = np.random.uniform(-1,1,500)
x2 = np.random.uniform(-1,1,500)
error = np.random.normal(0,0.1,500)
y = x1 + x2 + error
y_prob = np.exp(y) / (1+np.exp(y))
success = y_prob >= 0.6
failure = y_prob < 0.6
plt.scatter(x1[success], x2[success])
plt.scatter(x1[failure], x2[failure])
plt.xlabel('x1')
plt.ylabel('x2')
plt.title('Success vs. Failure')
plt.legend(['Success', 'Failure'], loc = 1)
```

[16]:
```
<matplotlib.legend.Legend at 0x1a199f1780>
```



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?

[17]:
```
X = np.column_stack((x1,x2))
X_train,X_test,y_train,y_test = train_test_split(X,success,test_size = 0.2)
```

[18]:
```
param = [{'C': np.linspace(0.001,10,10)}]
clf = GridSearchCV(SVC(),param, scoring = 'accuracy',cv =␣
 ↪10,return_train_score=True)
```

```
clf.fit(X_train, y_train)
```

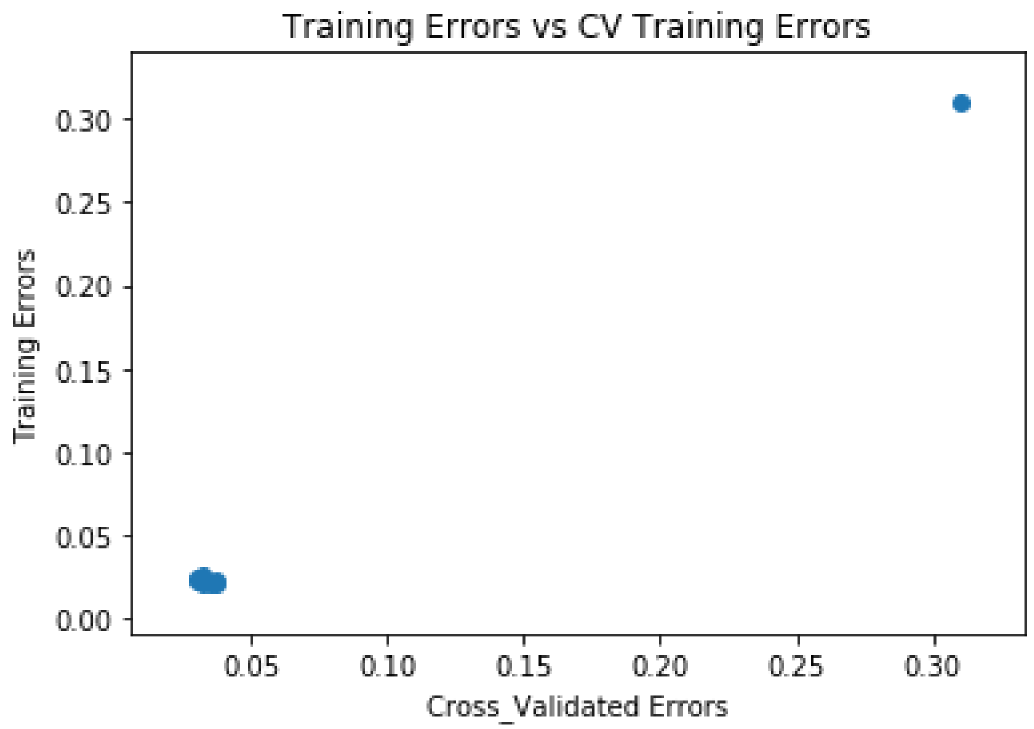
[18]:
```
GridSearchCV(cv=10, error_score='raise-deprecating',
             estimator=SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
                           decision_function_shape='ovr', degree=3,
                           gamma='auto_deprecated', kernel='rbf', max_iter=-1,
                           probability=False, random_state=None, shrinking=True,
                           tol=0.001, verbose=False),
             iid='warn', n_jobs=None,
             param_grid=[{'C': array([1.000e-03, 1.112e+00, 2.223e+00,
3.334e+00, 4.445e+00, 5.556e+00,
       6.667e+00, 7.778e+00, 8.889e+00, 1.000e+01])}],
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring='accuracy', verbose=0)
```

[19]:
```
cv_error = 1 - clf.cv_results_['mean_test_score']
training_error = 1 - clf.cv_results_['mean_train_score']
df_error = pd.DataFrame({'Cost': np.linspace(0.001,10,10), "CV errors":
                      cv_error, "Training errors": training_error})
df_error
```

[19]:
```
     Cost  CV errors  Training errors
0   0.001     0.3100         0.309999
1   1.112     0.0325         0.025835
2   2.223     0.0300         0.024442
3   3.334     0.0325         0.023611
4   4.445     0.0300         0.023333
5   5.556     0.0350         0.020834
6   6.667     0.0325         0.020833
7   7.778     0.0375         0.022222
8   8.889     0.0375         0.022498
9  10.000     0.0325         0.022779
```

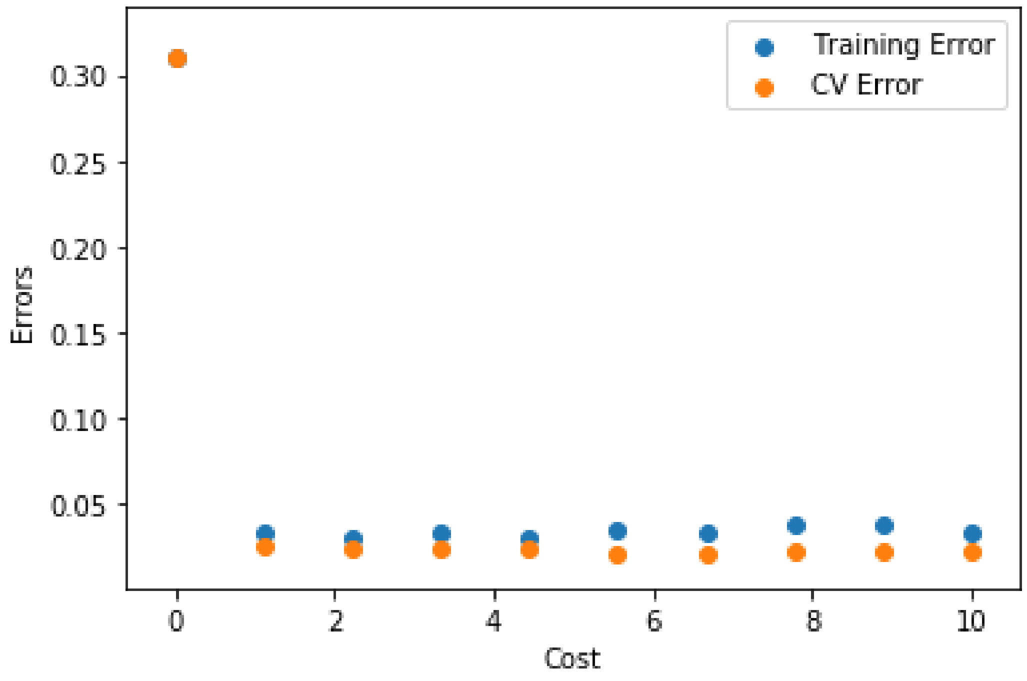
[20]:
```
plt.scatter(x = cv_error, y= training_error)
plt.ylabel('Training Errors')
plt.xlabel('Cross_Validated Errors')
plt.title('Training Errors vs CV Training Errors')
```

[20]:
```
Text(0.5, 1.0, 'Training Errors vs CV Training Errors')
```



[21]:
```
plt.scatter(np.linspace(0.001,10,10),1-clf.cv_results_['mean_test_score'])
plt.scatter(np.linspace(0.001,10,10),1-clf.cv_results_['mean_train_score'])
plt.legend({'CV Error','Training Error'})
plt.xlabel('Cost')
plt.ylabel('Errors')
```

[21]:
```
Text(0, 0.5, 'Errors')
```

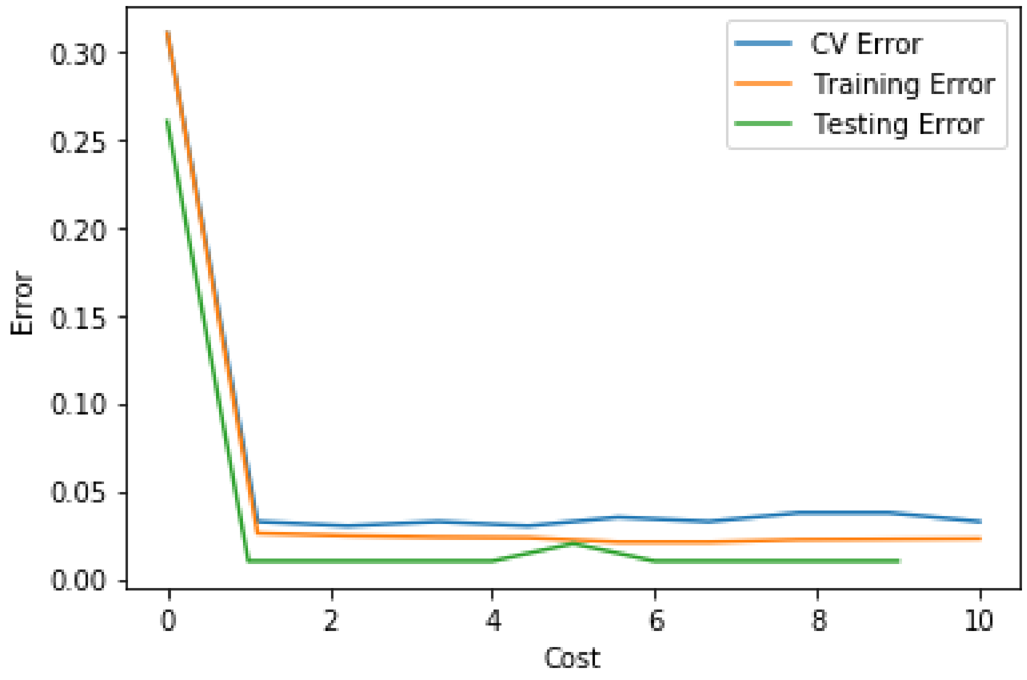


From the above two graphs, we can see that most of the cross-validation and training errors are clustering together and keep at a similar level after cost=1.

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?

[22]:
```
test_error = []
for c in np.linspace(0.001,10,10):
    model = SVC(C=c, kernel='linear').fit(X_train, y_train)
    test_error.append(1 - model.score(X_test, y_test))
plt.plot(np.linspace(0.001,10,10),1 - clf.cv_results_['mean_test_score'])
plt.plot(np.linspace(0.001,10,10),1 - clf.cv_results_['mean_train_score'])
plt.plot(test_error)
plt.legend(['CV Error','Training Error', 'Testing Error'])
plt.xlabel('Cost')
plt.ylabel('Error')
test_error
```

[22]:
```
[0.26,
 0.010000000000000009,
 0.010000000000000009,
 0.010000000000000009,
 0.010000000000000009,
 0.020000000000000018,
 0.010000000000000009,
 0.010000000000000009,
 0.010000000000000009,
 0.010000000000000009]
```



14.Discuss your results.

The test errors are minimized when cost=1. We do not need a very high cost. A SVM with small cost value is sufficient and may perform better (with the lowest error rate) than larger ones.

0.2 Application: Predicting attitudes towards racist college professors

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.

[23]:
```
train = pd.read_csv('gss_train.csv')
test = pd.read_csv('gss_test.csv')
X_train_gss = train.drop(columns="colrac")
y_train_gss = train["colrac"]
```

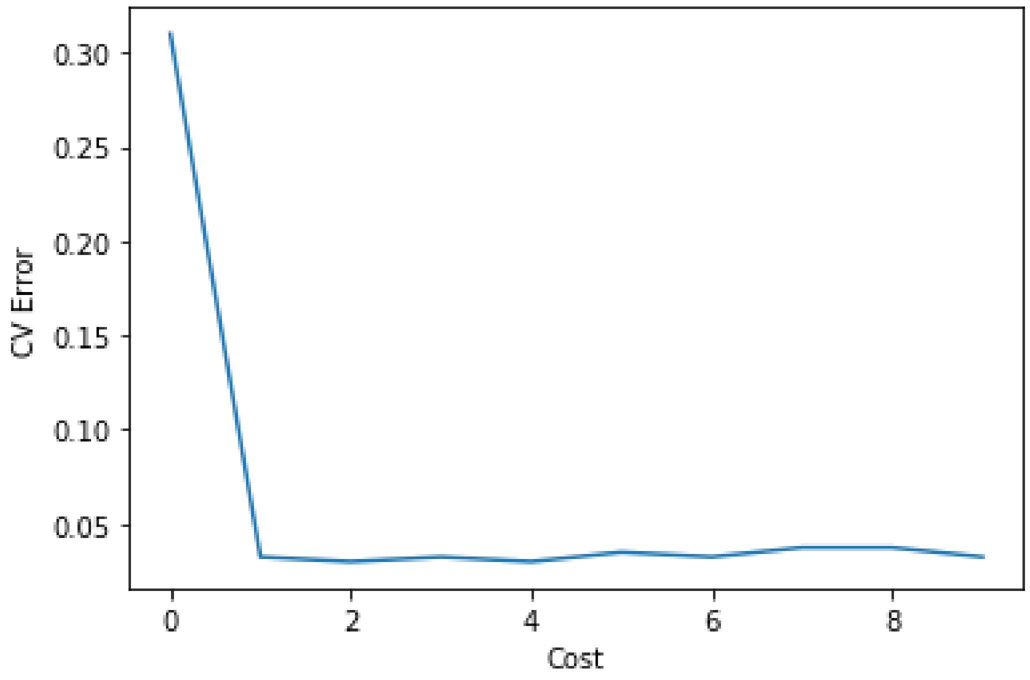
[24]:
```
param= [{'C': np.linspace(0.001,10,10)}]
clf = GridSearchCV(SVC(), param, scoring = 'accuracy',
                   cv = 10, return_train_score=True)
print(clf.fit(X_train, y_train))
print(clf.best_estimator_)
```

```
GridSearchCV(cv=10, error_score='raise-deprecating',
             estimator=SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
                                decision_function_shape='ovr', degree=3,
                                gamma='auto_deprecated', kernel='rbf', max_iter=-1,
                                probability=False, random_state=None, shrinking=True,
                                tol=0.001, verbose=False),
                 iid='warn', n_jobs=None,
                 param_grid=[{'C': array([1.000e-03, 1.112e+00, 2.223e+00,
3.334e+00, 4.445e+00, 5.556e+00,
       6.667e+00, 7.778e+00, 8.889e+00, 1.000e+01])}],
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring='accuracy', verbose=0)
SVC(C=2.223, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='auto_deprecated',
    kernel='rbf', max_iter=-1, probability=False, random_state=None,
    shrinking=True, tol=0.001, verbose=False)
```

[25]:
```python
cv_error = 1 - clf.cv_results_['mean_test_score']
training_error = 1 - clf.cv_results_['mean_train_score']
df_error = pd.DataFrame({'Cost': np.linspace(0.001,10,10), "CV errors":
                      cv_error, "Training errors": training_error})
plt.plot(cv_error)
plt.xlabel('Cost')
plt.ylabel('CV Error')
print(cv_error)
print('minimum cv error:', min(cv_error))
df_error
```

```
[0.31   0.0325 0.03   0.0325 0.03   0.035  0.0325 0.0375 0.0375 0.0325]
minimum cv error: 0.030000000000000027
```

[25]:
```
     Cost  CV errors  Training errors
0   0.001     0.3100         0.309999
1   1.112     0.0325         0.025835
2   2.223     0.0300         0.024442
3   3.334     0.0325         0.023611
4   4.445     0.0300         0.023333
5   5.556     0.0350         0.020834
6   6.667     0.0325         0.020833
7   7.778     0.0375         0.022222
8   8.889     0.0375         0.022498
9  10.000     0.0325         0.022779
```



The CV errors after 10-fold cross validation are pretty consistent across all cost values when cost >= 1. Cost of 3.34, 6.67, 8.89, 10 have the lowest cv error of 0.4

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.).

[26]:
```
param = [{'kernel':('rbf','poly','linear'),'C': [0.001,1,2,3,4,5],
          'gamma':('scale','auto'),'degree':[1,2,3,4,5]}]
clf = GridSearchCV(SVC(), param, scoring = 'accuracy',
                   cv = 10,return_train_score=True)
print(clf.fit(X_train, y_train))
print('____________________')
print('Best Estimator:')
print(clf.best_estimator_)
print('Mean cross-validated score of the best_estimator =', clf.best_score_)
```

```
GridSearchCV(cv=10, error_score='raise-deprecating',
             estimator=SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
                           decision_function_shape='ovr', degree=3,
                           gamma='auto_deprecated', kernel='rbf', max_iter=-1,
                           probability=False, random_state=None, shrinking=True,
                           tol=0.001, verbose=False),
             iid='warn', n_jobs=None,
             param_grid=[{'C': [0.001, 1, 2, 3, 4, 5],
                          'degree': [1, 2, 3, 4, 5], 'gamma': ('scale', 'auto'),
                          'kernel': ('rbf', 'poly', 'linear')}],
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring='accuracy', verbose=0)
-------------------
Best Estimator:
SVC(C=1, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=1, gamma='scale', kernel='poly',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
Mean cross-validated score of the best_estimator = 0.9725
```

I used GridSearchCV to find the best paramters for the model. The best parameters is when C=1, degree=1, and gamma='scale' with polynomial kernel. It has a Mean cross-validated score of the best_estimator of 0.9725. The polynomial kernal performs better than the radial and linear kernel.