Yehong_Deng_HW6

March 8, 2020

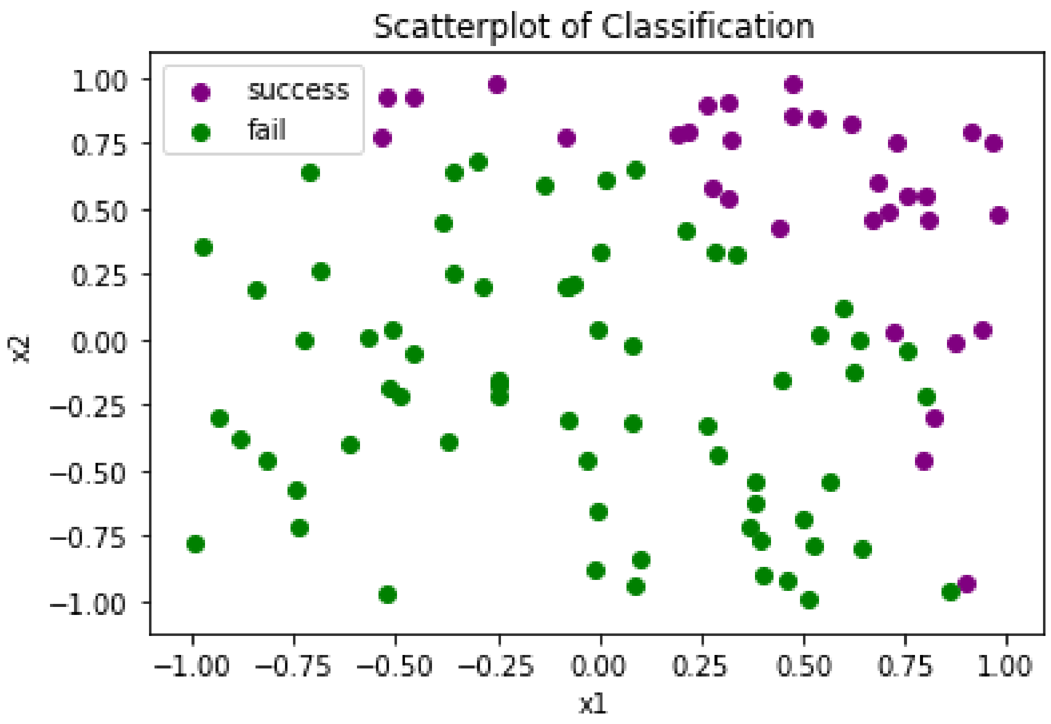
Non-linear separation 1. (15 points) Generate a simulated two-class data set with 100 observations and two features in which there is a visible (clear) but still non-linear separation between the two classes. Show that in this setting, a support vector machine with a radial kernel will outperform a support 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.

```
[17]: import numpy as np
      import pandas as pd
      import matplotlib.pyplot as plt
      from sklearn.svm import SVC
      from sklearn.linear_model import LogisticRegression
      from sklearn.model_selection import train_test_split, cross_val_score,␣
       ↪GridSearchCV
```

```
[18]: np.random.seed(234)
      x1 = np.random.uniform(-1, 1, 100)
      x2 = np.random.uniform(-1, 1, 100)
      err = np.random.uniform(0, 1, 100)
      y = x1 + x1**2 + x2 + x2**2 + err
      y_class = (y - np.min(y)) / (np.max(y) - np.min(y))
      success = y_class > 0.5
      fail = y_class <= 0.5
```

```
[19]: plt.scatter(x1[success], x2[success], color = 'purple')
      plt.scatter(x1[fail], x2[fail], color = 'green')
      plt.xlabel('x1')
      plt.ylabel('x2')
      plt.legend(['success', 'fail'])
      plt.title("Scatterplot of Classification")
```

```
[19]: Text(0.5, 1.0, 'Scatterplot of Classification')
```



[20]:
```
x_df = pd.DataFrame({'x1':x1, 'x2': x2})
x_train, x_test, y_train, y_test = train_test_split(x_df, success, test_size =␣
 ↪0.3, random_state = 234)
```

[21]:
```
#svc referenced from https://scikit-learn.org/stable/modules/generated/sklearn.
 ↪svm.SVC.html#sklearn.svm.SVC
radial_svm = SVC(kernel = 'rbf', gamma = 'scale').fit(x_train, y_train)
linear_svm = SVC(kernel = 'linear').fit(x_train, y_train)
```

[22]:
```
#training Error
train_err_radial = 1 - radial_svm.score(x_train, y_train)
train_err_linear = 1 - linear_svm.score(x_train, y_train)
print("training error for radial kernel SVM:", train_err_radial)
print("training error for linear kernel SVM:", train_err_linear)
```

```
training error for radial kernel SVM: 0.05714285714285716
training error for linear kernel SVM: 0.1428571428571429
```

[23]:
```
#test error
test_err_radial = 1 - radial_svm.score(x_test, y_test)
test_err_linear = 1 - linear_svm.score(x_test, y_test)
print("test error for radial kernel SVM:", test_err_radial)
print("test error for linear kernel SVM:", test_err_linear)
```

```
test error for radial kernel SVM: 0.033333333333333326
test error for linear kernel SVM: 0.13333333333333333
```

Based on the calculated result for error rate, radial SVM outperform the linear kernel SVM both on training error and test error.

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.

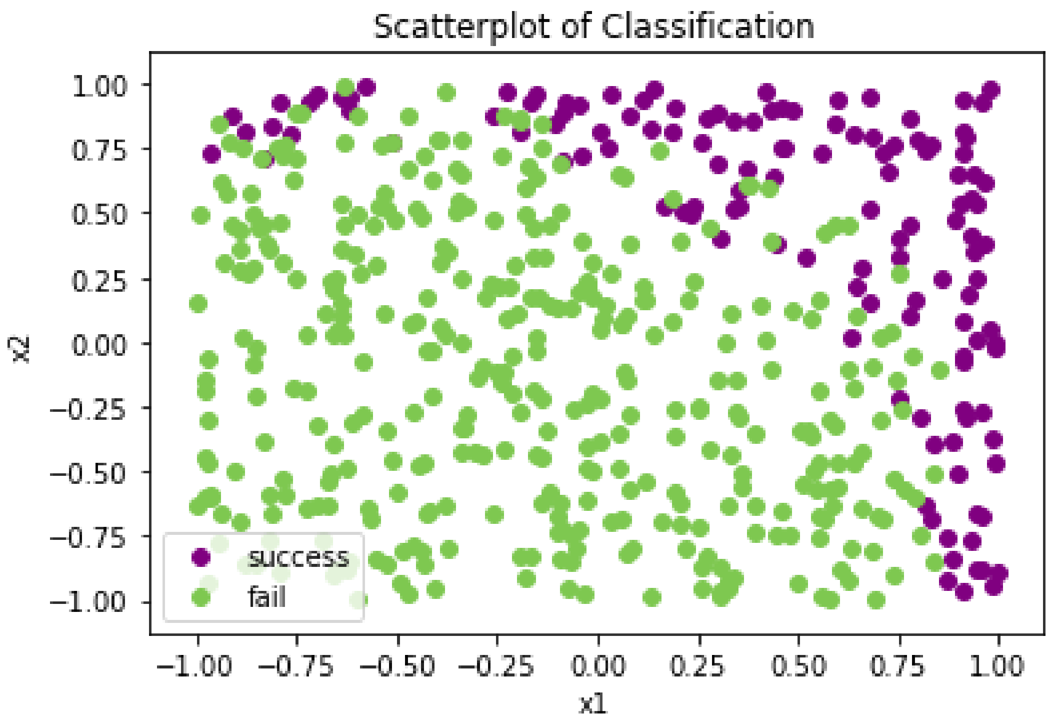
2. (5 points) Generate a data set with $n = 500$ and $p = 2$, such that the observations belong to two classes with some overlapping, non-linear boundary between them.

[24]:
```
x3 = np.random.uniform(-1, 1, 500)
x4 = np.random.uniform(-1, 1, 500)
err = np.random.uniform(0, 1, 500)
y2 = x3 + x3**2 + x4 + x4**2 + err
y2_class = (y2 - np.min(y2)) / (np.max(y2) - np.min(y2))
success2 = y2_class > 0.5
fail2 = y2_class <= 0.5
```

3. (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.

[25]:
```
plt.scatter(x3[success2], x4[success2], color = 'purple')
plt.scatter(x3[fail2], x4[fail2], color = '#7EC850')
plt.xlabel('x1')
plt.ylabel('x2')
plt.legend(['success', 'fail'])
plt.title("Scatterplot of Classification")
```

[25]:
```
Text(0.5, 1.0, 'Scatterplot of Classification')
```



4. (5 points) Fit a logistic regression model to the data, using X_1 and X_2 as predictors.

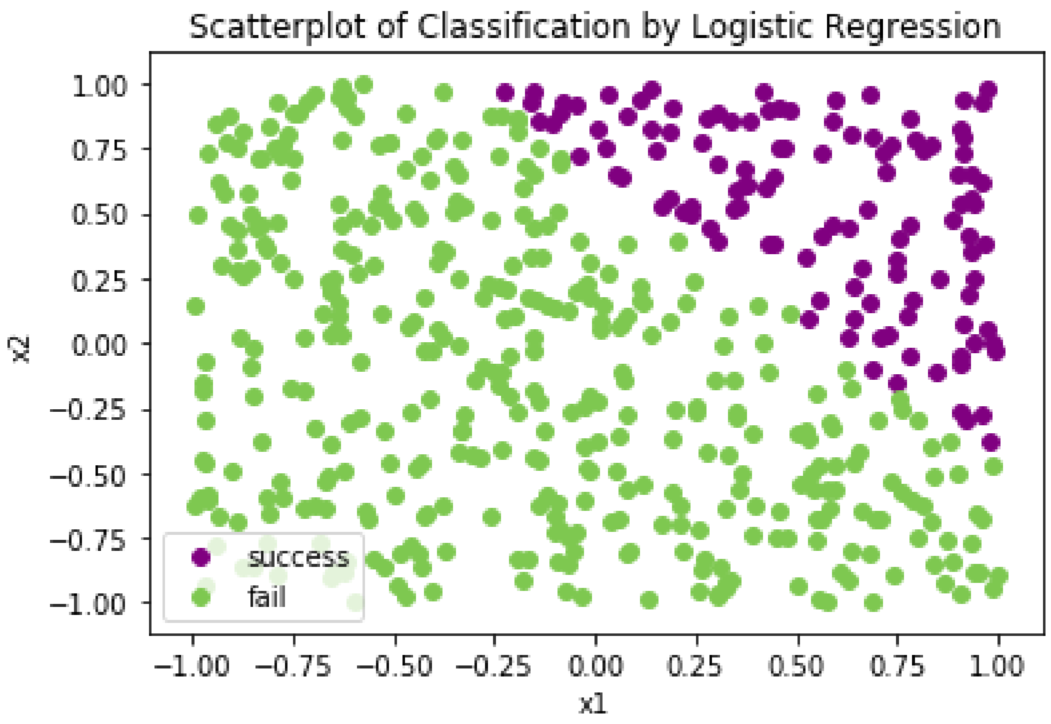
[26]:
```python
import warnings
warnings.filterwarnings('ignore')
x_df = pd.DataFrame({'x1':x3, 'x2': x4})
logit_reg = LogisticRegression().fit(x_df, success2)
```

5. (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).

[27]:
```python
logit_pred = logit_reg.predict(x_df)

plt.scatter(x3[logit_pred], x4[logit_pred], color = 'purple')
plt.scatter(x3[~logit_pred], x4[~logit_pred], color = '#7EC850')
plt.xlabel('x1')
plt.ylabel('x2')
plt.legend(['success', 'fail'])
plt.title("Scatterplot of Classification by Logistic Regression")
```

[27]:
```
Text(0.5, 1.0, 'Scatterplot of Classification by Logistic Regression')
```



6. (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).

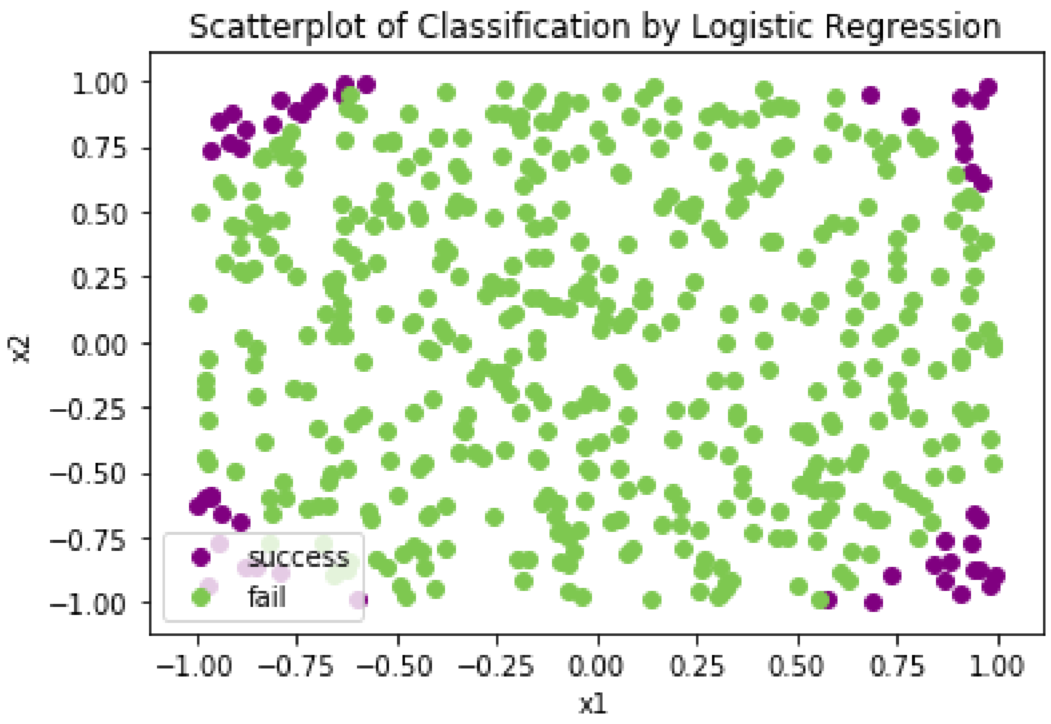
[30]:
```
x_df_non_linear = pd.DataFrame({'x1':x3**2, 'x2': x4**2})
logit_non_linear = LogisticRegression().fit(x_df_non_linear, success2)
```

7. (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.

[31]:
```
logit_pred2 = logit_non_linear.predict(x_df_non_linear)

plt.scatter(x3[logit_pred2], x4[logit_pred2], color = 'purple')
plt.scatter(x3[~logit_pred2], x4[~logit_pred2], color = '#7EC850')
plt.xlabel('x1')
plt.ylabel('x2')
plt.legend(['success', 'fail'])
plt.title("Scatterplot of Classification by Logistic Regression")
```

[31]:
```
Text(0.5, 1.0, 'Scatterplot of Classification by Logistic Regression')
```

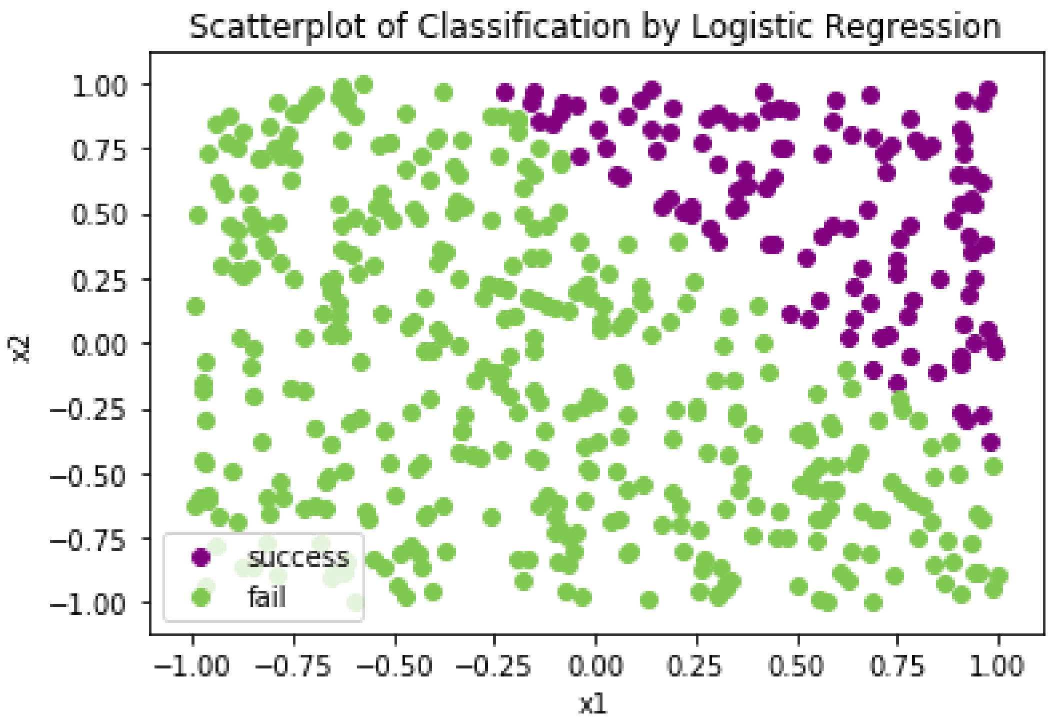


8. (5 points) Now, fit a support vector classifier (linear kernel) to the data with original X_1 and X_2 as predictors. Obtain a class prediction for each observation. Plot the observations, colored according to the predicted class labels.

[33]:
```
x_df = pd.DataFrame({'x1':x3, 'x2': x4})
linear_svm = SVC(kernel = 'linear').fit(x_df, success2)
linear_pred = linear_svm.predict(x_df)

plt.scatter(x3[linear_pred], x4[linear_pred], color = 'purple')
plt.scatter(x3[~linear_pred], x4[~linear_pred], color = '#7EC850')
plt.xlabel('x1')
plt.ylabel('x2')
plt.legend(['success', 'fail'])
plt.title("Scatterplot of Classification by Logistic Regression")
```

[33]:
```
Text(0.5, 1.0, 'Scatterplot of Classification by Logistic Regression')
```

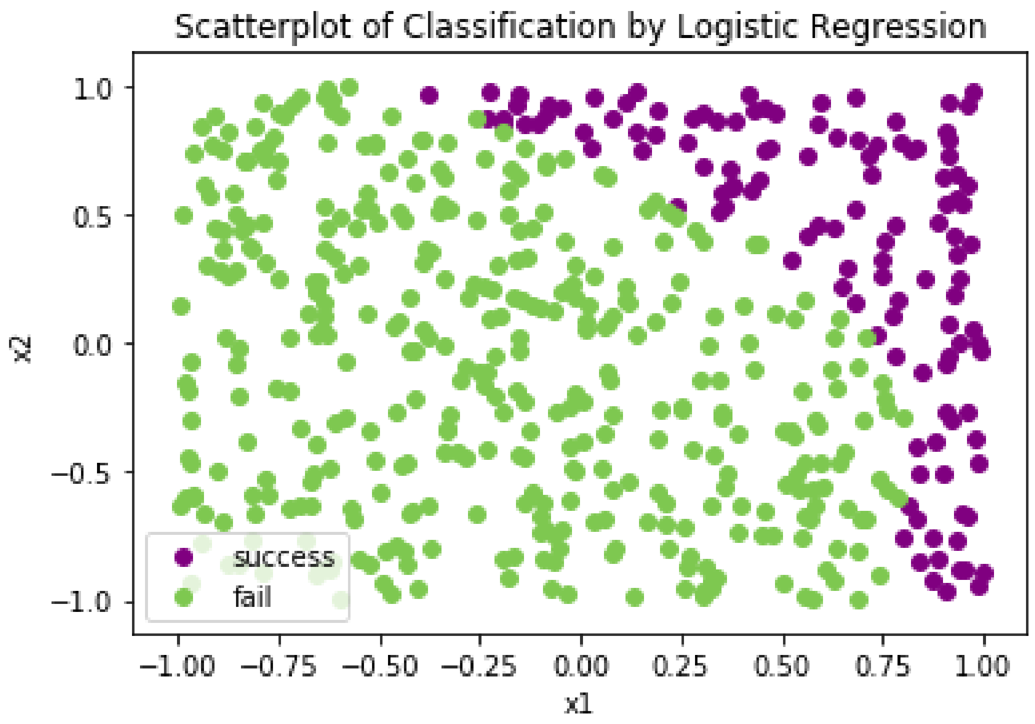


9. (5 points) Fit a SVM using a non-linear kernel to the data. Obtain a class prediction for each observation. Plot the observations, colored according to the predicted class labels.

[34]:
```
non_linear_svm = SVC(kernel = 'rbf', gamma = 'scale').fit(x_df, success2)
non_linear_pred = non_linear_svm.predict(x_df)

plt.scatter(x3[non_linear_pred], x4[non_linear_pred], color = 'purple')
plt.scatter(x3[~non_linear_pred], x4[~non_linear_pred], color = '#7EC850')
plt.xlabel('x1')
plt.ylabel('x2')
plt.legend(['success', 'fail'])
plt.title("Scatterplot of Classification by Logistic Regression")
```

[34]:
```
Text(0.5, 1.0, 'Scatterplot of Classification by Logistic Regression')
```



10. (5 points) Discuss your results and specifically the tradeoffs between estimating non-linear decision boundaries using these two different approaches.

[37]:
```
#Accuracy
print('linear logistic regression:', logit_reg.score(x_df, success2))
print('non_linear logistic regression:',logit_non_linear.score(x_df_non_linear,␣
↪success2))
print('linear SVM:',linear_svm.score(x_df,success2))
print('non_linear SVM:',non_linear_svm.score(x_df,success2))
```

```
linear logistic regression: 0.882
non_linear logistic regression: 0.756
linear SVM: 0.88
non_linear SVM: 0.91
```

As we can see in the result accuracy score, linear logistic regression and linear SVM have very similar results. Nonetheless, the non-linear SVM has a much better performance than the non_linear logistic regression. Among all the methods, the non_linear SVM has the highest accuracy rate. When comparing the two non_linear methods, we have to note that the non_linear logistic regression can specify the non_linear relations among the predictors while the the SVM only has the built_in black box method. Therefore, the tradeoffs is that whethre we should sacrifice the accuracy of estimation to use a computational cheaper method and get clearer interpretations of the relations among predictors in using logistic regression. In other hand, should we just let the built_in function to get a higher accurate estimation.

Tuning cost

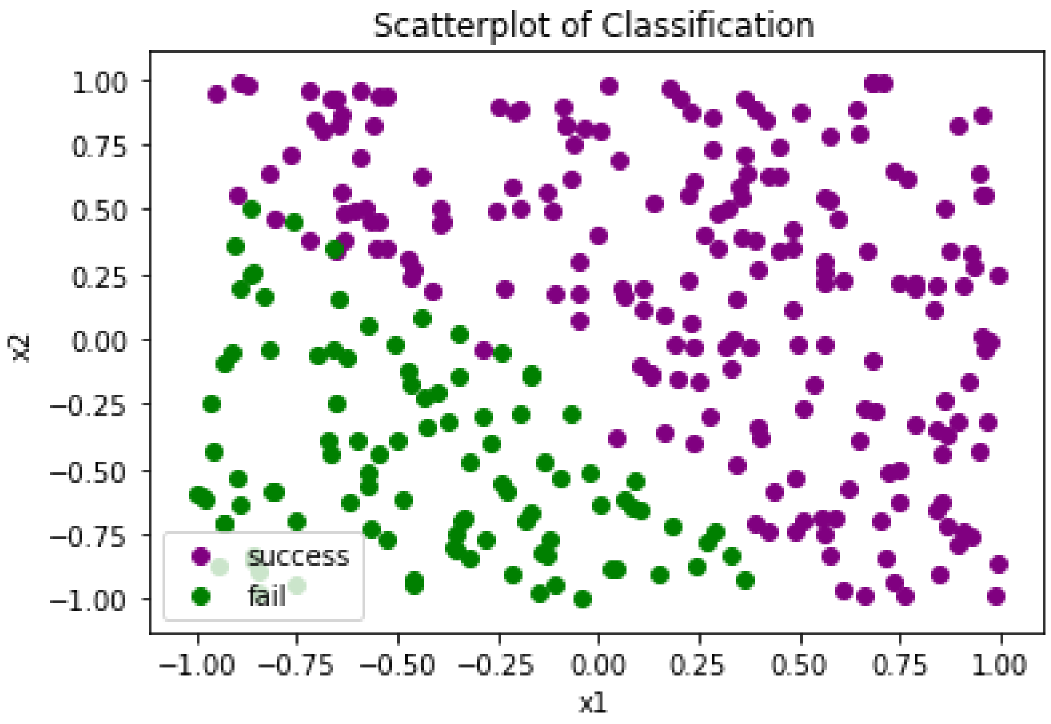
In class we learned that in the case of data that is just barely linearly separable, a support vector classifier with a small value of cost that misclassifies a couple of training observations may perform better on test data than one with a huge value of cost that does not misclassify any training observations.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

11. (5 points) Generate two-class data with $p = 2$ in such a way that the classes are just barely linearly separable.

[41]:
```
x1 = np.random.uniform(-1, 1, 300)
x2 = np.random.uniform(-1, 1, 300)
err = np.random.uniform(0, 0.2, 300)
y = x1 + x2 + err
y_class = (y - np.min(y)) / (np.max(y) - np.min(y))
success = y_class > 0.4
fail = y_class <= 0.4
```

[42]:
```
plt.scatter(x1[success], x2[success], color = 'purple')
plt.scatter(x1[fail], x2[fail], color = 'green')
plt.xlabel('x1')
plt.ylabel('x2')
plt.legend(['success', 'fail'])
plt.title("Scatterplot of Classification")
```

[42]:
```
Text(0.5, 1.0, 'Scatterplot of Classification')
```



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?

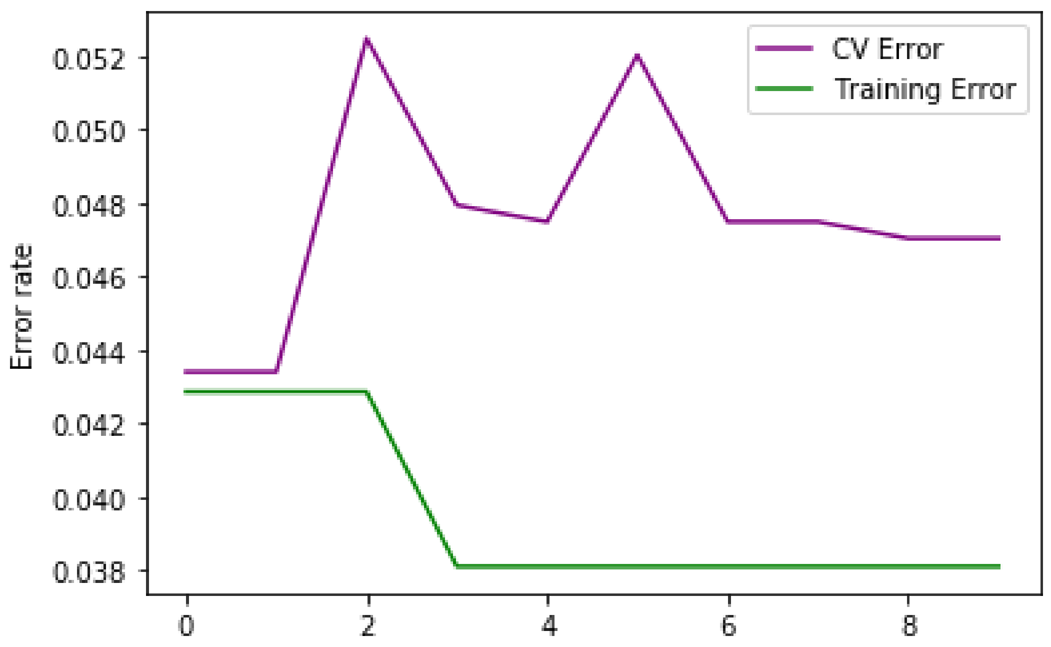
[43]:
```
x_df = pd.DataFrame({'x1':x1, 'x2': x2})
x_train, x_test, y_train, y_test = train_test_split(x_df, success, test_size =␣
↪0.3, random_state = 234)
```

[53]:
```
cost = [i for i in range(1,11)]
cv_err = []
train_err = []
for i in cost:
    model = SVC(C = i, kernel = 'linear')
    cv = 1 - np.mean(cross_val_score(model, x_train, y_train, cv = 10, scoring␣
↪= 'accuracy'))
    train = 1 - model.fit(x_train, y_train).score(x_train, y_train)
    cv_err.append(cv)
    train_err.append(train)
err_table = pd.DataFrame({'cost': cost, 'cv error': cv_err, 'training error':␣
↪train_err})
err_table
```

[53]:
```
   cost  cv error  training error
0     1  0.043398        0.042857
1     2  0.043398        0.042857
2     3  0.052489        0.042857
3     4  0.047944        0.038095
4     5  0.047489        0.038095
5     6  0.052035        0.038095
6     7  0.047489        0.038095
7     8  0.047489        0.038095
8     9  0.047035        0.038095
9    10  0.047035        0.038095
```

[54]:
```
plt.plot(cv_err, color = 'purple')
plt.plot(train_err, color = 'green')
plt.ylabel('Error rate')
plt.legend(['CV Error','Training Error'])
```

[54]:
```
<matplotlib.legend.Legend at 0x16f76f3be08>
```



As the cost value increases, the cv error first increase and had a peak at 0.052489 when cost equal 3, and then decrease to a local minimum at 0.047489 when cost equals to 5. Then, cv error increase again to a peak of 0.052035 at cost equals to 6. Finally, it begins to be stable around at 0.048 as cost continue to increase. For the training error, the error has its maximum 0.042857 when cost range from 1 to 3, then decrease and stable around 0.038095.

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?

[55]:
```
test_err = []
for i in cost:
    model = SVC(C = i, kernel = 'linear')
    test = 1 - model.fit(x_test, y_test).score(x_test, y_test)
    test_err.append(test)
```

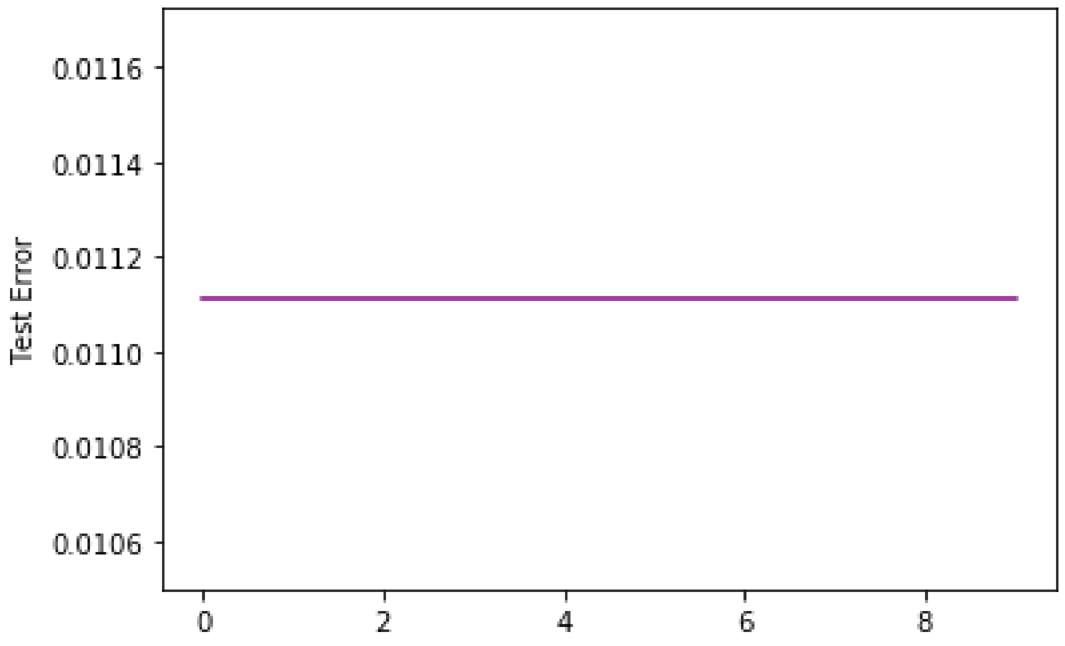
[57]:
```
err_table = pd.DataFrame({'cost': cost, 'test error': test_err})
err_table
```

[57]:
```
   cost  test error
0     1    0.011111
1     2    0.011111
2     3    0.011111
3     4    0.011111
4     5    0.011111
5     6    0.011111
```

```
6     7    0.011111
7     8    0.011111
8     9    0.011111
9    10    0.011111
```

[56]:
```
plt.plot(test_err, color = 'purple')
plt.ylabel('Test Error')
```

[56]:
```
Text(0, 0.5, 'Test Error')
```



The test error is constant at 0.011111 for all cost values. compare to the training and cv error, the test error does not change across all cost values. By comparison, the cv error has its minimum when cost value is smaller than 2, and the trainign error has its minimum when cost value is bigger than 4.

14. (5 points) Discuss your results.

This result suggest that when the data is barely linearly separable, a small cost value will be the best method. As we can see in the scatter plot, the data is already clearly separated, noises are only along the boundary. So a smaller tolerance to error will be benefitial because it needs smaller bias to seperate the data at the boundary.

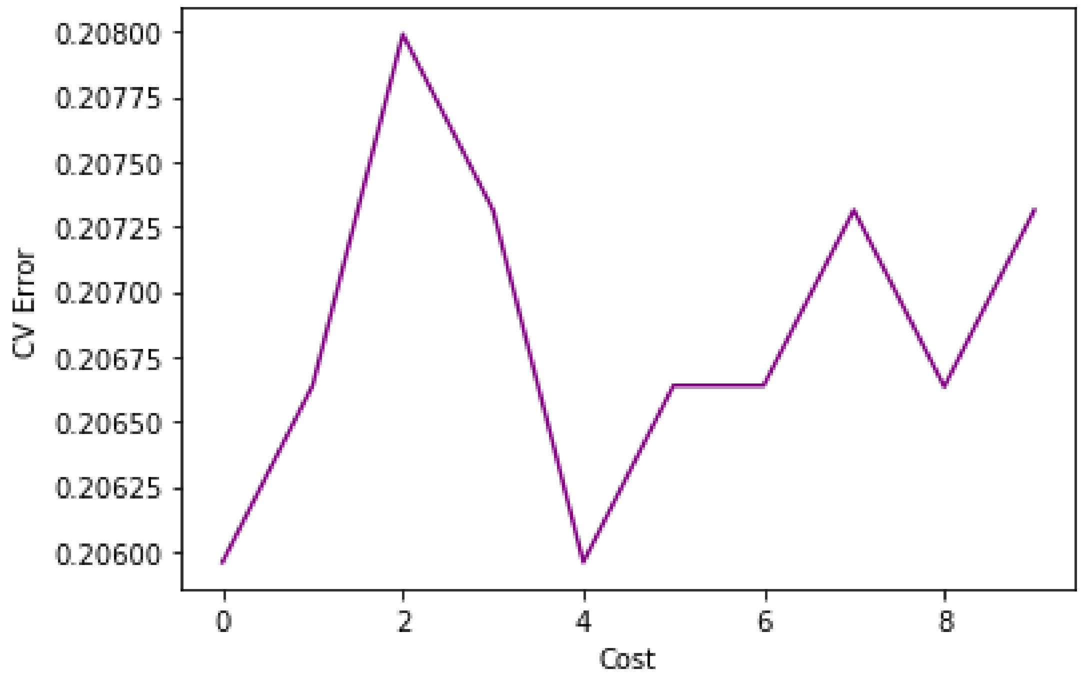
Application: Predicting attitudes towards racist college professors

15. (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.

```
[58]: gss_train = pd.read_csv('gss_train.csv')
      x_train = gss_train.drop('colrac', axis = 1)
      y_train = gss_train['colrac']
```

```
[59]: cv_err = []
      for i in cost:
          model = SVC(C = i, kernel = 'linear')
          cv = 1 - np.mean(cross_val_score(model, x_train, y_train, cv = 10, scoring␣
       ↪= 'accuracy'))
          cv_err.append(cv)
      plt.plot(cv_err, color = 'purple')
      plt.xlabel('Cost')
      plt.ylabel('CV Error')
      err_table = pd.DataFrame({'cost': cost, 'cv error': cv_err})
      err_table
```

```
[59]:    cost  cv error
      0     1  0.205958
      1     2  0.206638
      2     3  0.207990
      3     4  0.207314
      4     5  0.205958
      5     6  0.206638
      6     7  0.206638
      7     8  0.207314
      8     9  0.206634
      9    10  0.207314
```



According to the table as well as the graph, the CV error has its minimum at cost equals to 1 at first, and then increase to its maximum at cost equals to 3. Then, it decreases and reaches its minimum again when cost equals to 5. Although the graph seems to show a great fluctaution, it actually is quite table around 0.206.

16. (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.).

[65]:
```
models = {
    SVC(kernel = 'rbf'): {'C': [i for i in range(1,11)], 'degree': [2,3,4,5],␣
 ↪'gamma': ['scale', 'auto']},
    SVC(kernel = 'poly'): {'C': [i for i in range(1,11)], 'degree': [2,3,4,5],␣
 ↪'gamma': ['scale', 'auto']}
}
```

[66]:
```
for model, param in models.items():
    grid_search = GridSearchCV(model, param, cv = 10, refit = True)
    grid_search.fit(x_train, y_train)
    print(grid_search.best_estimator_)
    print(grid_search.best_score_)
    print('----------------------')
```

```
SVC(C=10, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=2, gamma='scale', kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
0.7825793382849426
----------------------
SVC(C=9, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=2, gamma='scale', kernel='poly',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
0.7886563133018231
----------------------
```

For the radial basis kernel, the best model's C is equal to 10, the degree equals to 2, and the gamma is equal to scale. the mean accuracy is 0.7825793382849426. For the polynomial basis kernel, the best C is 9, the best degree is 2, and the best gamma is scale. Then mean accuracy for the polynomial basis kernel is 0.7886563133018231. Compared this two method, the polynomial basis kernel perform better.