In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import math

import warnings
warnings.filterwarnings('ignore')

import patsy
from sklearn import preprocessing
from sklearn import svm
from sklearn import metrics
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
from sklearn.datasets import make_moons, make_classification
from sklearn.linear_model import LogisticRegression
from sklearn.datasets.samples_generator import make_blobs
from sklearn.model_selection import train_test_split
```

Conceptual Exercises

Non-linear separation

1.

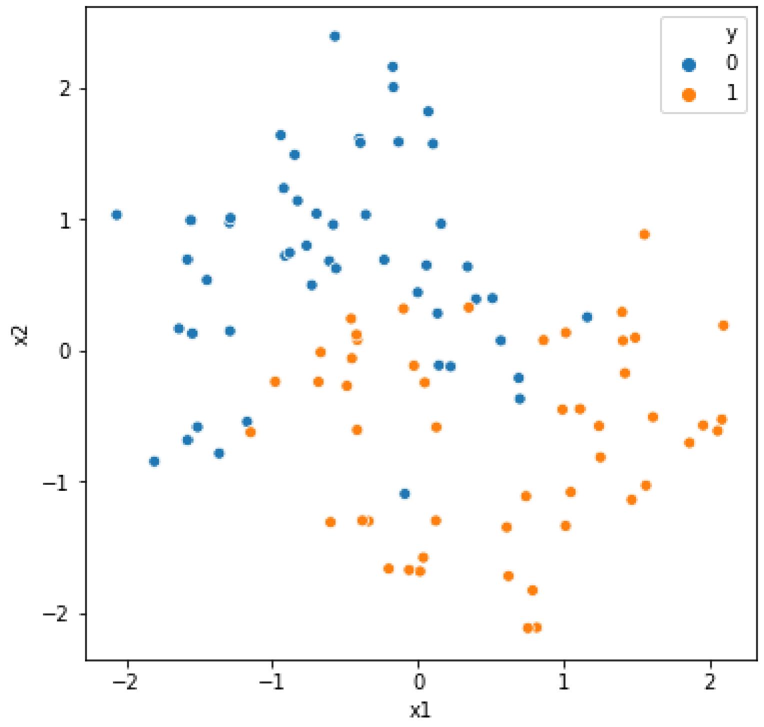
Generate a simulated two-class data set with 100 observations and two features in which there is a visible (clear) but still non-linear separation between the two classes.

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.

In [163]:
```
# Generate the data set
n_samples = 100
noise     = .3
X_train, y_train = make_moons(n_samples=n_samples, noise=noise, random_s
tate=1314)
X_test, y_test = make_moons(n_samples=n_samples, noise=noise, random_sta
te=7071)

# Scale data
X_train = preprocessing.scale(X_train)
X_test = preprocessing.scale(X_test)
```

In [47]:
```
# Plot data
df = pd.concat([pd.DataFrame(data=X_train, columns=['x1', 'x2']), pd.Ser
ies(y_train, name='y')], axis=1)
plt.figure(figsize=(6, 6))
sns.scatterplot(x='x1', y='x2', hue='y', data=df);
```



In [15]:
```
df.head()
```

Out[15]:

	x1	x2	y
0	-0.696735	1.040153	0
1	1.016821	0.132107	1
2	-1.450183	0.533869	0
3	-0.827022	1.138286	0
4	0.126864	-0.588210	1

In [38]:
```
acc = {'model':[],
       'training accuracy': [],
       'test accuracy': []}
```

In [76]:
```python
def plot_clf(model, df, grid_range, hue):
    # Decision boundary plot

    # Get grid of values in given range
    x1 = grid_range
    x2 = grid_range
    xx1, xx2 = np.meshgrid(x1, x2, sparse=False)
    Xgrid = np.stack((xx1.flatten(), xx2.flatten())).T

    # Get decision boundary values for plot grid
    decision_boundary      = model.predict(Xgrid)
    decision_boundary_grid = decision_boundary.reshape(len(x2), len(x1))

    # Get decision function values for plot grid
    decision_function      = model.decision_function(Xgrid)
    decision_function_grid = decision_function.reshape(len(x2), len(x1))

    fig = plt.figure(figsize=(10, 10))
    plt.contourf(x1, x2, decision_function_grid);
    plt.contour(x1, x2, decision_boundary_grid);

    sns.scatterplot(x='x1', y='x2', hue=hue, data=df)
    plt.show();
```

SVC: Linear Kernel

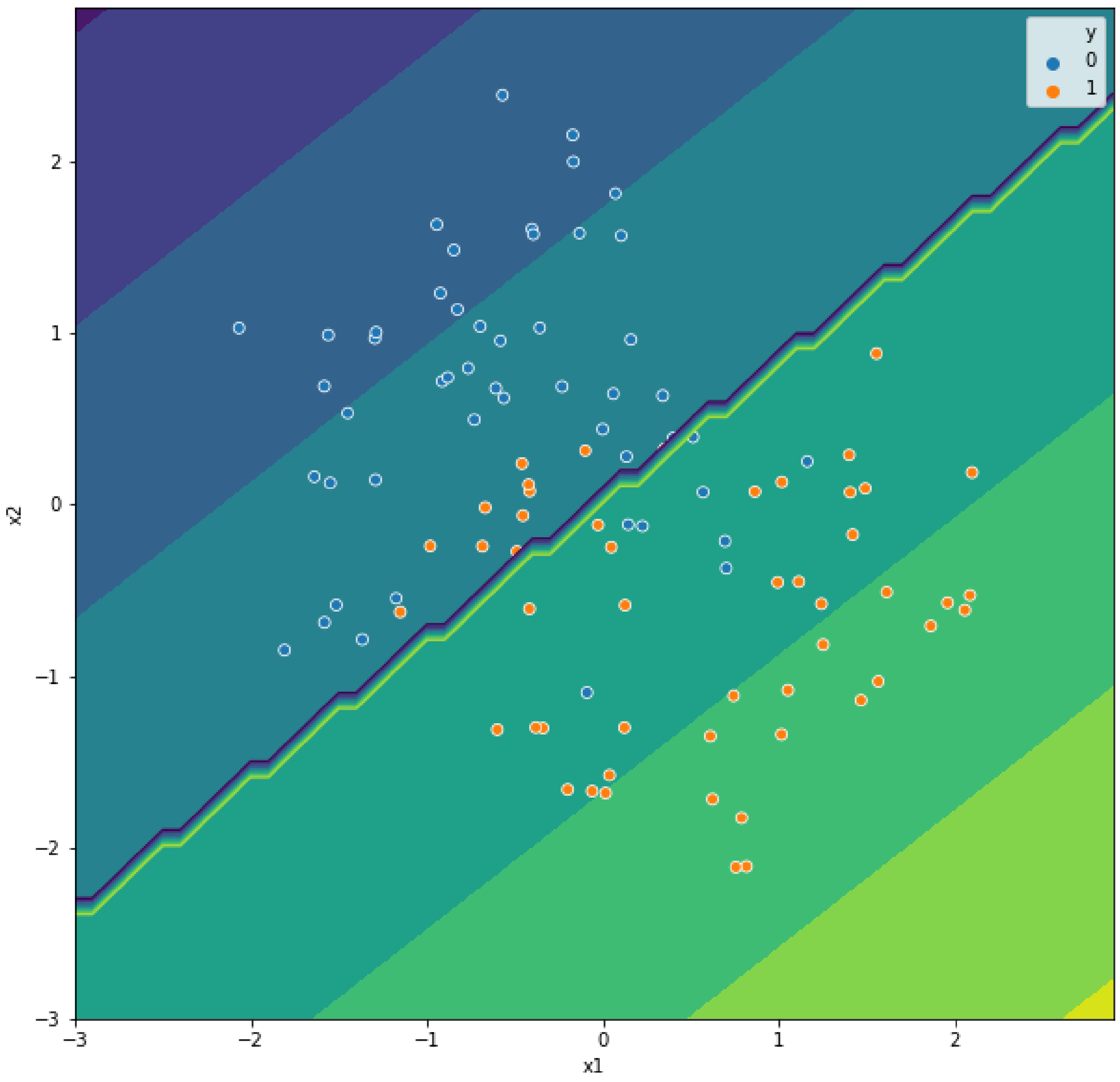
In [29]:
```python
print(svm.SVC().get_params().keys())
```

```
dict_keys(['C', 'break_ties', 'cache_size', 'class_weight', 'coef0', 'd
ecision_function_shape', 'degree', 'gamma', 'kernel', 'max_iter', 'prob
ability', 'random_state', 'shrinking', 'tol', 'verbose'])
```

In [36]:
```python
para_svc = {'C': np.arange(1, 30)}
clf_svc = GridSearchCV(svm.SVC(kernel='linear'), para_dt,
                       scoring='accuracy', cv=10)
clf_svc.fit(X_train, y_train)
clf_svc.best_estimator_
```

Out[36]:
```
SVC(C=1, break_ties=False, cache_size=200, class_weight=None, coef0=0.
0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='lin
ear',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

In [39]:
```
svc = clf_svc.best_estimator_.fit(X_train, y_train)

plot_clf(svc, df, np.arange(-3, 3, .1), 'y')

acc['model'].append('Linear Kernel')
acc['training accuracy'].append(svc.score(X_train, y_train))
acc['test accuracy'].append(svc.score(X_test, y_test))
```

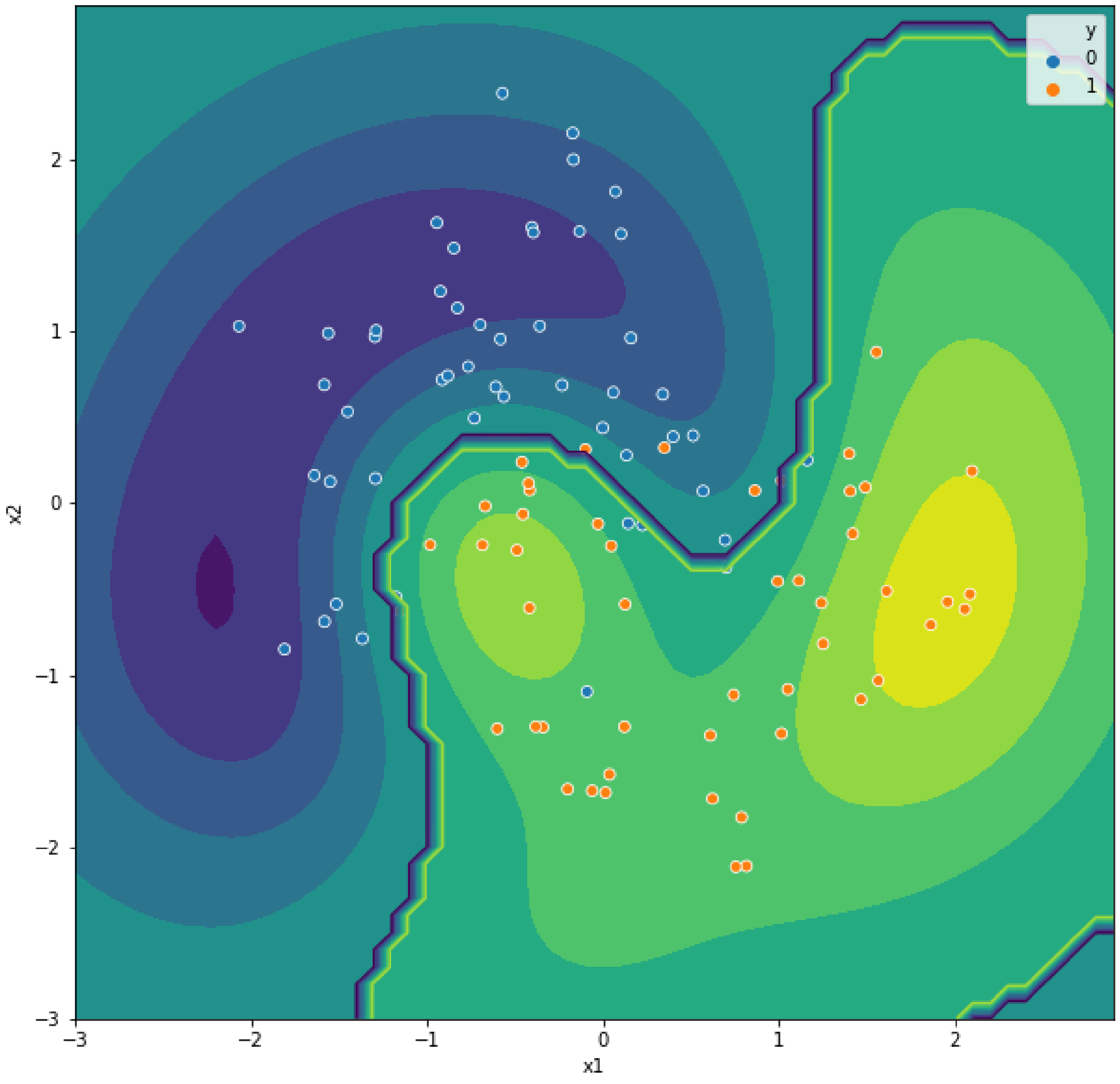


SVM: Radial Kernel

```
In [40]: para_svm = {'C': np.arange(1, 30)}
         clf_svm = GridSearchCV(svm.SVC(kernel='rbf'), para_dt,
                                scoring='accuracy', cv=10)
         clf_svm.fit(X_train, y_train)
         clf_svm.best_estimator_
```

```
Out[40]: SVC(C=12, break_ties=False, cache_size=200, class_weight=None, coef0=0.
         0,
             decision_function_shape='ovr', degree=3, gamma='scale', kernel='rb
         f',
             max_iter=-1, probability=False, random_state=None, shrinking=True,
             tol=0.001, verbose=False)
```

```
In [41]: svm = clf_svm.best_estimator_.fit(X_train, y_train)

         plot_clf(svm, df, np.arange(-3, 3, .1), 'y')

         acc['model'].append('Radial Kernel')
         acc['training accuracy'].append(svm.score(X_train, y_train))
         acc['test accuracy'].append(svm.score(X_test, y_test))
```



```
In [42]: pd.DataFrame(acc)
```

Out[42]:

	model	training accuracy	test accuracy
0	Linear Kernel	0.81	0.88
1	Radial Kernel	0.90	0.94

As revealed by the plots and data, a support vector machine with a radial kernal outperforms a support vector classifier on the training data.

This also applies to the test data.

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.

```
In [90]: np.random.seed(1917)
         x1 = np.random.uniform(-0.5, 0.5, 500)
         x2 = np.random.uniform(-0.5, 0.5, 500)
         y = 1*(x1**2 - x2**2 > 0)
         df2 = pd.DataFrame({'x1': x1, 'x2': x2, 'y': y})
         df2.head()
```

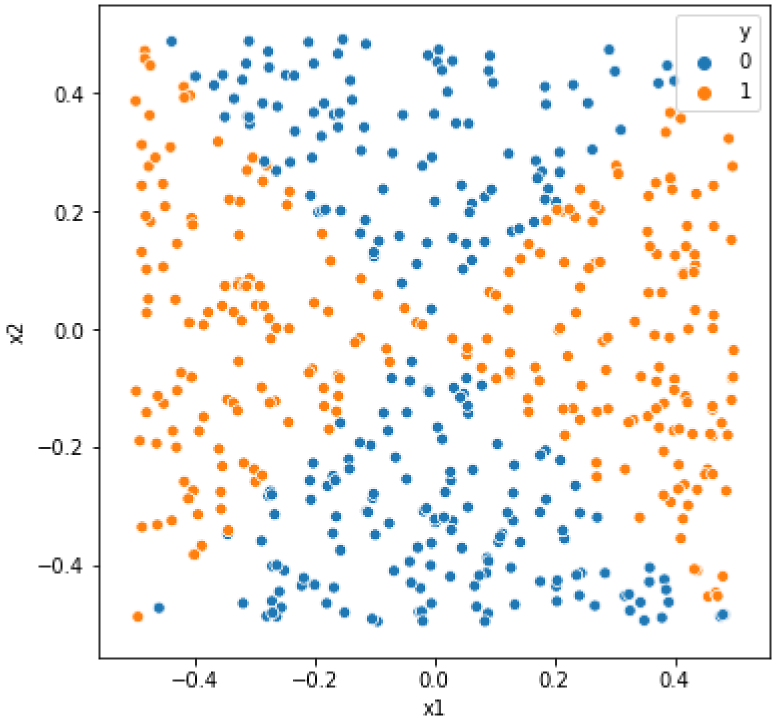
Out[90]:

	x1	x2	y
0	-0.345482	-0.348260	0
1	-0.292105	0.073305	1
2	-0.497280	-0.105342	1
3	-0.327037	0.079644	1
4	0.355556	-0.147961	1

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.

In [52]:
```
plt.figure(figsize=(6, 6))
sns.scatterplot(x='x1', y='x2', hue='y', data=df2);
```



4.

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

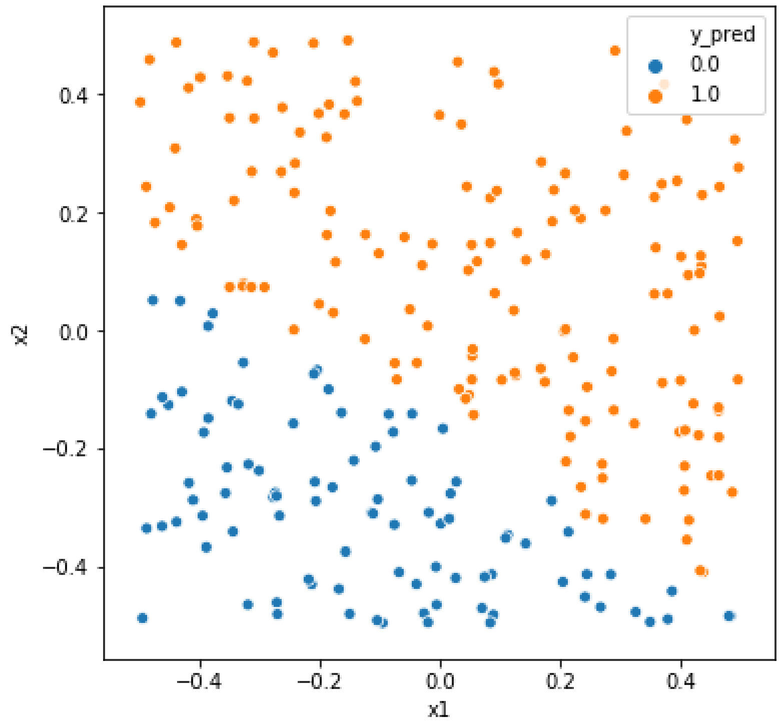
In [63]:
```
# Pre-process data with only linear features
f = 'y ~ x1 + x2'
y, X = patsy.dmatrices(f, df)
y = np.ravel(y)

train = np.random.random(len(y)) > 0.5
```

In [64]:
```
clf_lr = LogisticRegression().fit(X[train], y[train])
```

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).

```
In [65]: pred = clf_lr.predict(X[train])

         plot_df = pd.DataFrame({'x1':X[train][:,1], 'x2':X[train][:,2], 'y_pred'
         :pred})
         plt.figure(figsize=(6, 6))
         sns.scatterplot(x='x1', y='x2', hue='y_pred', data=plot_df)
         plt.show();
```



6.

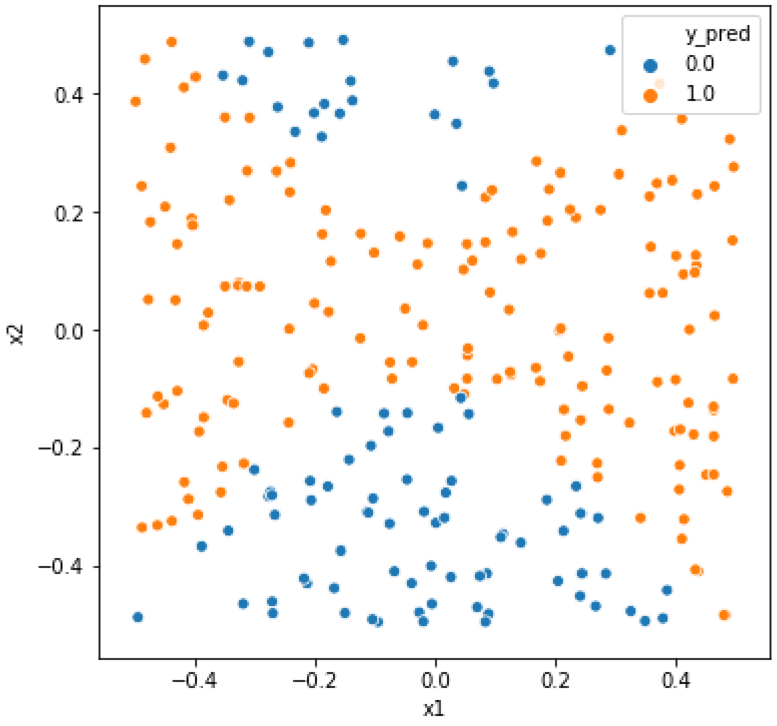
Now fit a logistic regression model to the data, but this time using some non-linear function of both X1 and X2 as predictors (e.g. X12, X1 × X2, log(X2), and so on).

```
In [85]: f = 'y ~ x1 + x2 + np.power(x1, 2) + np.power(x2, 2)'
         y, X = patsy.dmatrices(f, df)
         y = np.ravel(y)

         clf_lr_q = LogisticRegression().fit(X[train], y[train])
```

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.

```
In [86]: pred2 = clf_lr_q.predict(X[train])

         plot_df2 = pd.DataFrame({'x1':X[train][:,1], 'x2':X[train][:,2], 'y_pre
         d':pred2})
         plt.figure(figsize=(6, 6))
         sns.scatterplot(x='x1', y='x2', hue='y_pred', data=plot_df2)
         plt.show();
```

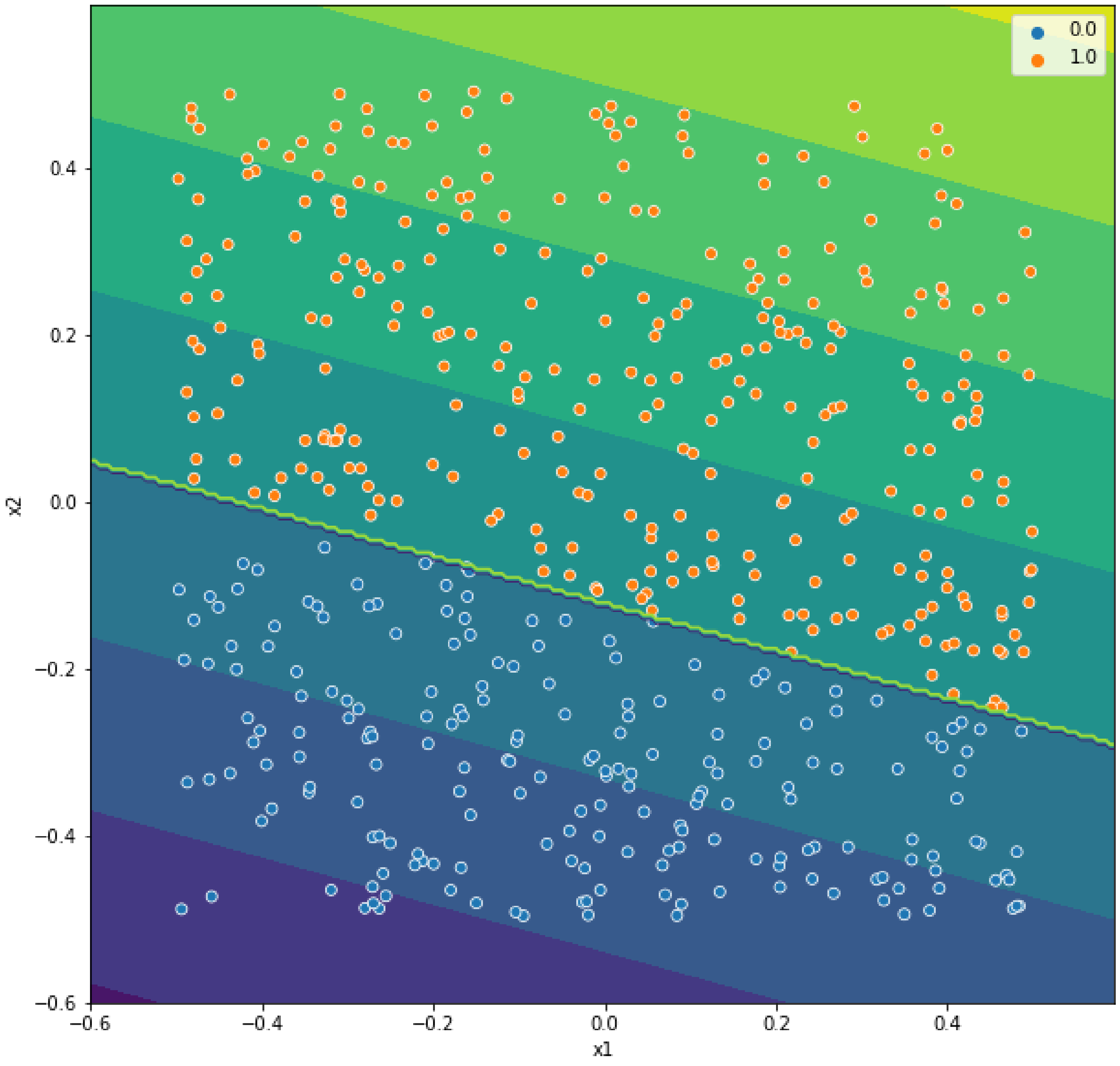


```
In [87]: print(f'Training accuracy: {clf_lr_q.score(X[train], y[train])}')
         print(f'Test accuracy     : {clf_lr_q.score(X[~train], y[~train])}')

         Training accuracy: 0.8477366255144033
         Test accuracy     : 0.8521400778210116
```

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.

```
In [88]: f = 'y ~ x1 + x2 - 1'
         y, X = patsy.dmatrices(f, df)
         y = np.ravel(y)

         svc = svm.SVC(kernel='linear', gamma=1, C=1, random_state=0,
                       probability=True).fit(X[train], y[train])

         plot_clf(svc, df, np.arange(-.6, .6, .005),
                  hue=svc.predict(X))
```



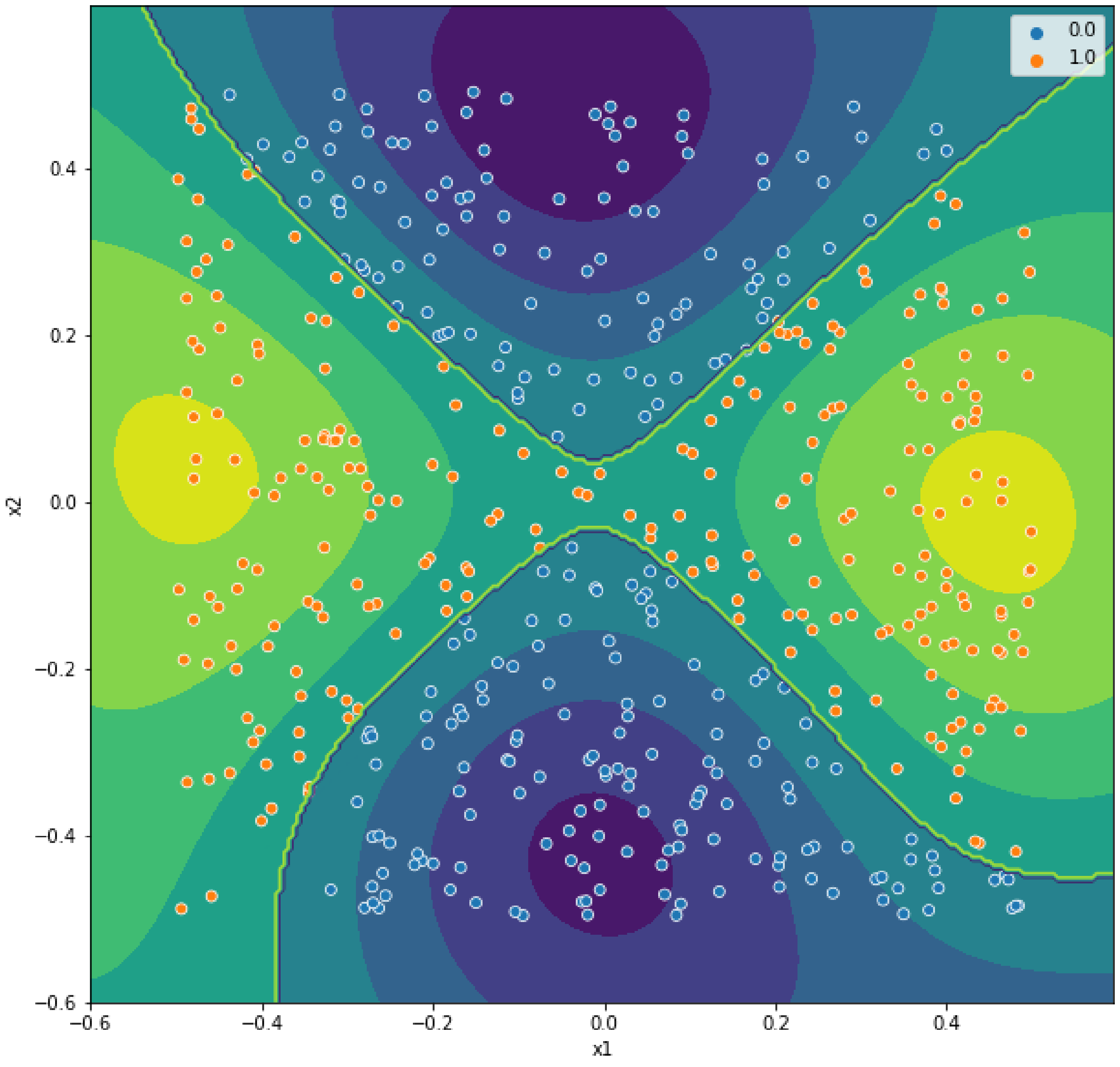
9.

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.

```
In [80]: para_svm2 = {'C': np.arange(1, 30),
                      'kernel': ['rbf', 'poly']}
         clf_svm2 = GridSearchCV(svm.SVC(), para_svm2,
                                 scoring='accuracy', cv=10)
         clf_svm2.fit(X[train], y[train])
         clf_svm2.best_estimator_
```

```
Out[80]: SVC(C=18, break_ties=False, cache_size=200, class_weight=None, coef0=0.
         0,
             decision_function_shape='ovr', degree=3, gamma='scale', kernel='rb
         f',
             max_iter=-1, probability=False, random_state=None, shrinking=True,
             tol=0.001, verbose=False)
```

```
In [82]: svm2 = clf_svm2.best_estimator_.fit(X[train], y[train])

         plot_clf(svm2, df, np.arange(-.6, .6, .005), hue=svm2.predict(X))
```



```
In [89]: print(f'Training accuracy: {svm2.score(X[train], y[train])}')
         print(f'Test accuracy    : {svm2.score(X[~train], y[~train])}')
```

```
Training accuracy: 0.9876543209876543
Test accuracy    : 0.9610894941634242
```

10.

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

As depicted above, both linear logistic regression and SVC with linear kernel have poor performance on classifying the observations. It is reasonable since the original seperation between classes is far from linear.

Both SVM with radial kernel and polynominal logistic regression achieve better performance and have similar non-linear decision boundaries. The SVM outperforms logistic regression on both training (98.77% vs. 84.77%) and test (96.11% vs. 85.21%) accuracy.

0.8477366255144033 Test accuracy : 0.8521400778210

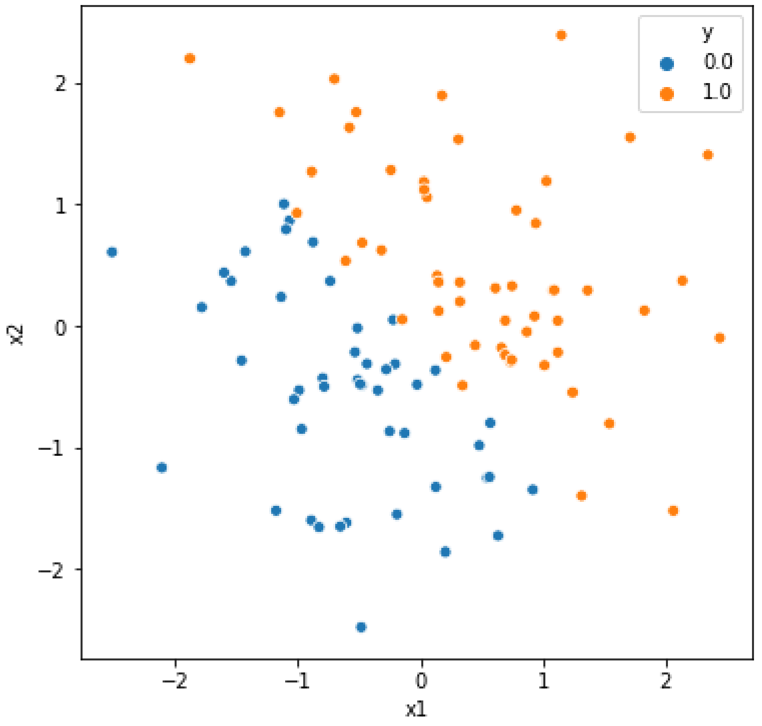
Tuning cost

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.

In [93]:
```python
X, _ = make_blobs(n_samples=200, centers=1, n_features=2, random_state=0
)
y = np.zeros(len(X))
y[np.sum(X, axis=1) > 5] = 1

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5,
random_state=1)

# Scale data
X_train = preprocessing.scale(X_train)
X_test = preprocessing.scale(X_test)

# Plot data
df = pd.DataFrame({'x1':X_train[:,0], 'x2':X_train[:,1], 'y':y_train})
plt.figure(figsize=(6, 6))
sns.scatterplot(x='x1', y='x2', hue='y', data=df)
plt.show();
```



1. 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?
2. 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 [124]:
```python
m = svm.SVC(kernel='linear', C=2, random_state=1917)
cross_val_score(m, X_train, y_train, cv=5).mean()
```

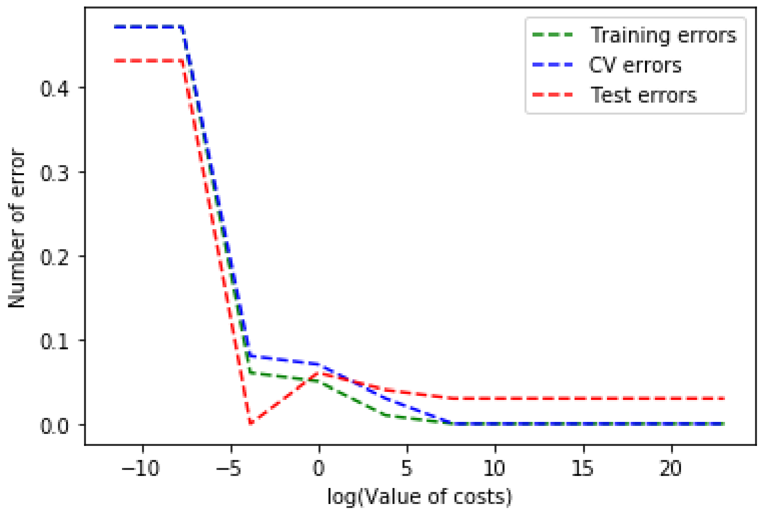
Out[124]:
```
0.93
```

In [149]:
```
costs = np.logspace(-5, 10, 10)
scores = []
for i in costs:
    model = svm.SVC(kernel='linear', C=i, random_state=1917)
    cv_er = 1 - cross_val_score(model, X_train, y_train, cv=10).mean()
    model.fit(X_train, y_train)
    train_error = 1 - model.score(X_train, y_train)
    test_error = 1 - model.score(X_test, y_test)
    scores.append([i, cv_er, train_error, test_error])

columns = ['Value of cost', 'CV errors', 'Training errors', 'Test error
s']
cv_df = pd.DataFrame(data=np.asarray(scores), columns=columns)

cv_df
```

Out[149]:

	Value of cost	CV errors	Training errors	Test errors
0	1.000000e-05	0.47	0.47	0.43
1	4.641589e-04	0.47	0.47	0.43
2	2.154435e-02	0.08	0.06	0.00
3	1.000000e+00	0.07	0.05	0.06
4	4.641589e+01	0.03	0.01	0.04
5	2.154435e+03	0.00	0.00	0.03
6	1.000000e+05	0.00	0.00	0.03
7	4.641589e+06	0.00	0.00	0.03
8	2.154435e+08	0.00	0.00	0.03
9	1.000000e+10	0.00	0.00	0.03

In [153]:
```
p1=plt.plot(np.log(costs), cv_df['Training errors'], 'g--', label='Train
ing errors')
p2=plt.plot(np.log(costs), cv_df['CV errors'], 'b--', label='CV errors')
p3=plt.plot(np.log(costs), cv_df['Test errors'], 'r--', label='Test erro
rs')

plt.xlabel('log(Value of costs)')
plt.ylabel('Number of error');
plt.legend()
plt.show();
```



In [168]:
```
from math import exp
exp(-4)
```

Out[168]:
```
0.01831563888873418
```

1. Discuss your results.

```
The plot above depicts the cv error rates, training error rates and test
 error rates at a range of cost values.
Training error has similar trend with cv error at small values and become
s lower as the value of costs increases.
The lowest test error rate, however, is achieved at a lower cost value
 (0.018). This suggests that for classes that are just barely linearly se
parable, a large value of costs would lead to overfit while a small one m
ay achive better accuracy.
```

Application: Predicting attitudes towards racist college professors

In this problem set, you are going to return to the GSS question from last week and predict attitudes towards racist college professors. Recall, each respondent was asked “Should a person who believes that Blacks are genetically inferior be allowed to teach in a college or university?” Given the kerfuffle over Richard J. Herrnstein and Charles Murray’s The Bell Curve and the ostracization of Nobel laureate James Watson over his controversial views on race and intelligence, this analysis will provide further insight into the public debate over this issue.

The outcome of interest `colrac` is a binary variable coded as either ALLOWED or NOT ALLOWED, where 1 = the racist professor should be allowed to teach, and 0 = the racist professor should not be allowed to teach. Some data pre-processing has been done in advance for you to ease your model fitting: (1) Missing values have been imputed; (2) Categorical variables with low-frequency classes had those classes collapsed into an “other” category; (3) Nominal variables with more than two classes have been converted to dummy variables; and (4) Remaining categorical variables have been converted to integer values, stripping their original labels. This week, building on last week’s problem set, you will approach this classification problem from an SVM-based framework.

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.

In [2]:
```
train = pd.read_csv('gss_train.csv')
test = pd.read_csv('gss_test.csv')

X_train, y_train = train.drop('colrac', axis=1), train['colrac']
X_test, y_test = test.drop('colrac', axis=1), test['colrac']
```

In [183]:
```
costs = np.logspace(-5, 1, 18)
scores = []
for i in costs:
    model = svm.SVC(kernel='linear', C=i, random_state=0)
    score = 1 - cross_val_score(model, X_train, y_train, cv=10).mean()
    scores.append([i, score])

columns=['Cost', 'CV error rate']
results_df = pd.DataFrame(data=np.asarray(scores), columns=columns)

display(results_df)
```

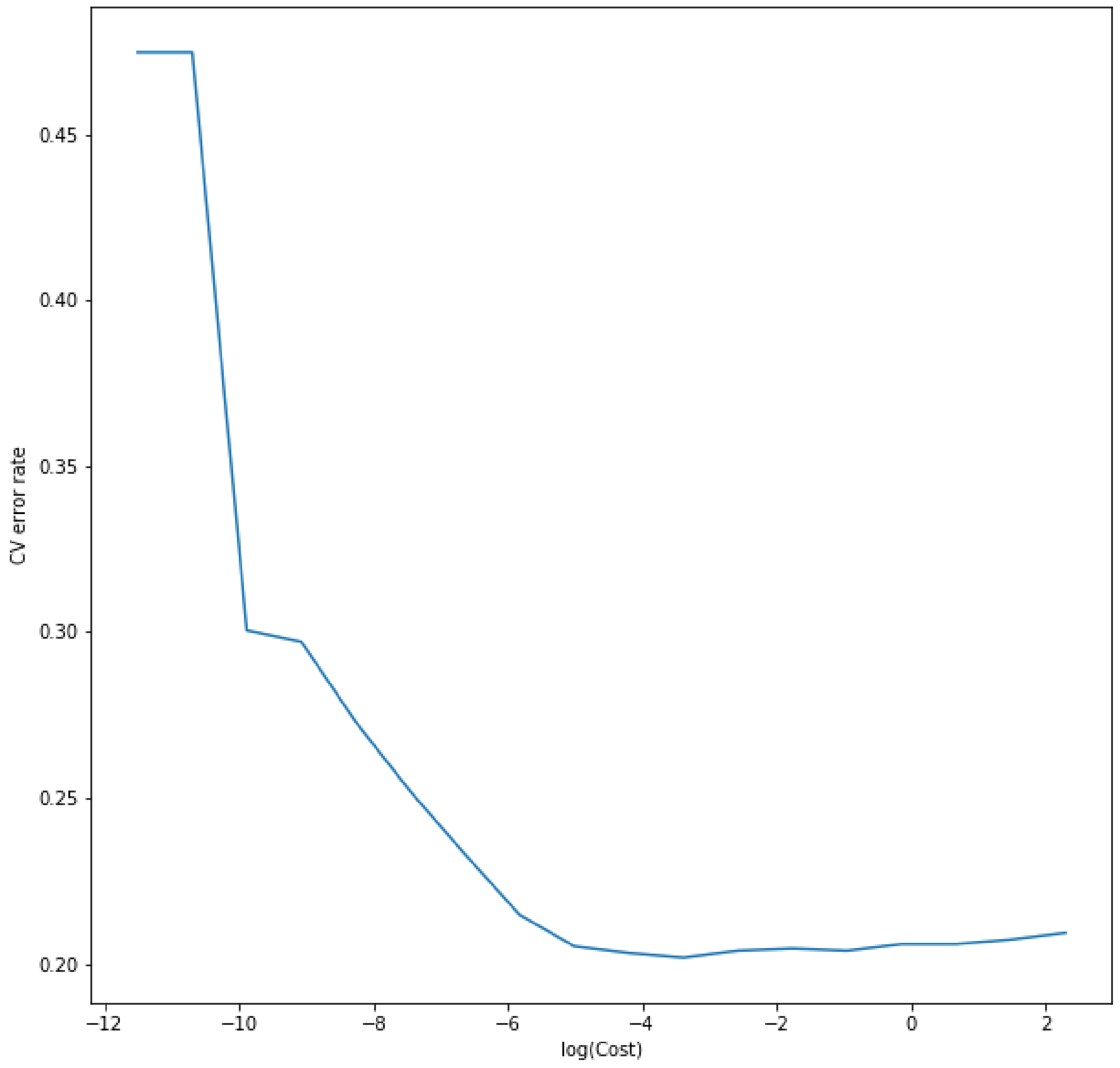
	Cost	CV error rate
0	0.000010	0.474678
1	0.000023	0.474678
2	0.000051	0.300440
3	0.000115	0.297061
4	0.000258	0.272773
5	0.000582	0.251859
6	0.001311	0.232949
7	0.002955	0.214724
8	0.006661	0.205265
9	0.015013	0.203242
10	0.033839	0.201891
11	0.076270	0.203918
12	0.171907	0.204585
13	0.387468	0.203909
14	0.873326	0.205931
15	1.968419	0.205931
16	4.436687	0.207283
17	10.000000	0.209310

In [187]:
```
results_df[results_df['CV error rate'] == results_df['CV error rate'].mi
n()]
```

Out[187]:

	Cost	CV error rate	log(Cost)
10	0.033839	0.201891	-3.386155

In [185]:
```
results_df['log(Cost)'] = np.log(results_df['Cost'])
plt.figure(figsize=(10,10))
sns.lineplot(x='log(Cost)', y='CV error rate', data=results_df);
```



In [ ]:
```

```

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.).

In [3]:
```
print(svm.SVC().get_params().keys())
```

```
dict_keys(['C', 'break_ties', 'cache_size', 'class_weight', 'coef0', 'd
ecision_function_shape', 'degree', 'gamma', 'kernel', 'max_iter', 'prob
ability', 'random_state', 'shrinking', 'tol', 'verbose'])
```

In [5]:
```python
para_svm = {'kernel': ['rdf', 'poly'],
            'C': np.logspace(-5, 0, 10),
            'gamma': ['scale', 'auto'],
            'degree': [3, 5, 7]
           }
grid_svm = GridSearchCV(svm.SVC(kernel='linear'), para_svm,
                        scoring='accuracy', cv=10,
                        return_train_score=True)
grid_svm.fit(X_train, y_train)
```

Out[5]:
```
GridSearchCV(cv=10, error_score=nan,
             estimator=SVC(C=1.0, break_ties=False, cache_size=200,
                           class_weight=None, coef0=0.0,
                           decision_function_shape='ovr', degree=3,
                           gamma='scale', kernel='linear', max_iter=-1,
                           probability=False, random_state=None, shrink
ing=True,
                           tol=0.001, verbose=False),
             iid='deprecated', n_jobs=None,
             param_grid={'C': array([1.00000000e-05, 3.59381366e-05, 1.
29154967e-04, 4.64158883e-04,
       1.66810054e-03, 5.99484250e-03, 2.15443469e-02, 7.74263683e-02,
       2.78255940e-01, 1.00000000e+00]),
                         'degree': [3, 5, 7], 'gamma': ['scale', 'aut
o'],
                         'kernel': ['rdf', 'poly']},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=Tr
ue,
             scoring='accuracy', verbose=0)
```

In [7]:
```
pd.DataFrame(grid_svm.cv_results_).sort_values('rank_test_score',
                                               ascending=True)
```

Out[7]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_C	param_degree	pa
27	0.096328	0.005596	0.005577	0.000151	0.000129155	3	
39	0.159773	0.014572	0.005206	0.000106	0.000464159	3	
51	0.436124	0.043629	0.004964	0.000139	0.0016681	3	
15	0.073116	0.001735	0.006181	0.000187	3.59381e-05	3	
63	1.405925	0.138902	0.004819	0.000120	0.00599484	3	
...	...	...	...	...	...	...	
14	0.001658	0.000185	0.000000	0.000000	3.59381e-05	3	
84	0.001833	0.000101	0.000000	0.000000	0.0774264	3	
42	0.001634	0.000132	0.000000	0.000000	0.000464159	5	
70	0.001603	0.000123	0.000000	0.000000	0.00599484	7	
0	0.003426	0.001177	0.000000	0.000000	1e-05	3	

120 rows × 34 columns

In [14]:
```
pd.DataFrame(grid_svm.cv_results_).sort_values('rank_test_score',
                                    ascending=True)['params'].iloc[0]
```

Out[14]:
```
{'C': 0.00012915496650148841, 'degree': 3, 'gamma': 'auto', 'kernel': 'p
oly'}
```

In [15]:
```
pd.DataFrame(grid_svm.cv_results_).sort_values('rank_test_score',
                                    ascending=True)['mean_test_score'].iloc[0]
```

Out[15]:
```
0.7940458915291131
```

The table above lists all tested SVMs models with different kernels, values for gamma, degree and cost. The models were tested through grid search and ranked by cross-validation test scores.

The model with the best performance has a polynomial kernal and a extremely small C (0.00013). The test score is 0.794, which is similar to the accuracy achieved by the SVC model.

```
In [ ]:
```