In [10]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import sklearn
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
import math
```

Conceptual exercises

Non-linear separation

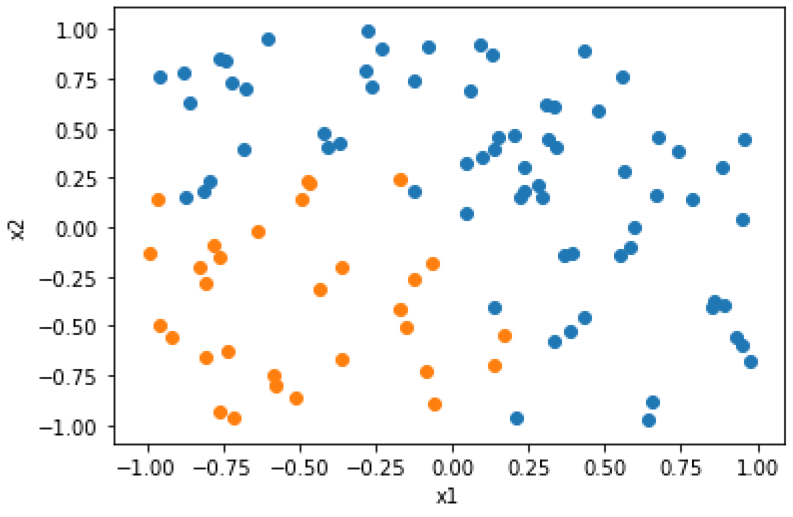
1. Generate a simulated two-class data set with 100 observations and two features in which there is a visible (clear) but still non-linear separation between the two classes. Show that in this setting, a support vector machine with a radial kernel will outperform a support vector classifier (a linear 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 [19]:

```
np.random.seed(0)
x1 = np.random.uniform(-1,1,100)
x2 = np.random.uniform(-1,1,100)
e = np.random.normal(0,0.1,100)
y = x1+x1*x1 + x2+x2*x2+e
y_prob = np.exp(y)/(1+np.exp(y))
plt.scatter(x1[y_prob>=0.5],x2[y_prob>=0.5])
plt.scatter(x1[y_prob<0.5],x2[y_prob<0.5])
plt.xlabel('x1')
plt.ylabel('x2')
```

Out[19]:

```
Text(0, 0.5, 'x2')
```



In [20]:

```
X = np.column_stack((x1,x2))
y = (y_prob>=0.5)

X_train,X_test,y_train,y_test = train_test_split(X,y,test_size = 0.2)
```

In [21]:

```
from sklearn import svm
svm_nonlinear = svm.SVC(gamma = 'auto').fit(X_train,y_train)
print('radical train error tate:',1-svm_nonlinear.score(X_train,y_train))
print('radical test error rate:',1-svm_nonlinear.score(X_test,y_test))
```

```
radical train error tate: 0.037499999999999998
radical test error rate: 0.19999999999999996
```

In [22]:

```
svm_linear = svm.SVC(gamma = 'auto',kernel = 'linear').fit(X_train,y_train)
print('linear train error tate:',1-svm_linear.score(X_train,y_train))
print('linear test error rate:',1-svm_linear.score(X_test,y_test))
```

```
linear train error tate: 0.07499999999999996
linear test error rate: 0.25
```

We can see that radical kernel SVM outperforms linear kernel SVM because the error rates are lower in both training and test sets in radical kernel SVM than linear kernel SVM

SVM v.s. logistic regression

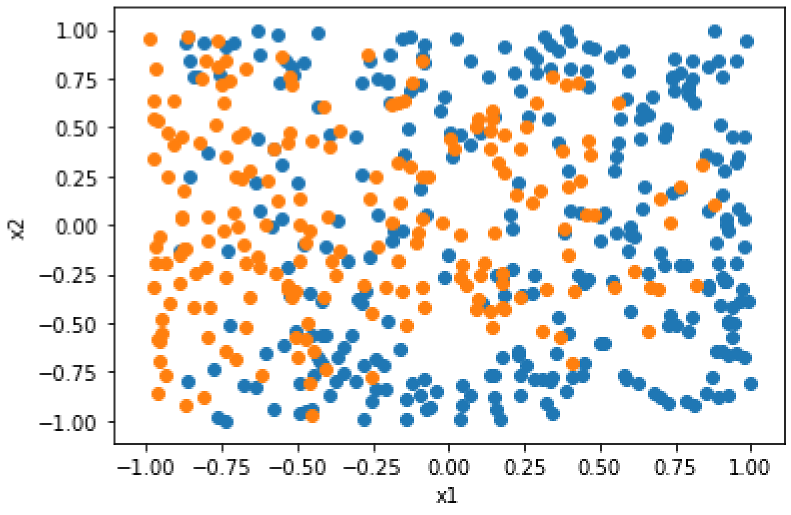
1. (5 points) Generate a data set with n = 500 and p = 2, such that the observations belong to two classes with some overlapping, non-linear boundary between them.
2. (5 points) Plot the observations with colors according to their class labels (y). Your plot should display X1 on the x-axis and X2 on the y-axis.

In [102]:

```
np.random.seed(0)
x1 = np.random.uniform(-1,1,500)
x2 = np.random.uniform(-1,1,500)
e = np.random.normal(0,0.5,500)
y = x1*x1*x1+x2*x2*x2*x2+e
y_prob = np.exp(y)/(1+np.exp(y))
plt.scatter(x1[y_prob>=0.5],x2[y_prob>=0.5])
plt.scatter(x1[y_prob<0.5],x2[y_prob<0.5])
plt.xlabel('x1')
plt.ylabel('x2')
```

Out[102]:

```
Text(0, 0.5, 'x2')
```



1. (5 points) Fit a logistic regression model to the data, using X1 and X2 as predictors.

In [103]:

```
from sklearn.linear_model import LogisticRegression
X = np.column_stack((x1,x2))
y = (y_prob>=0.5)
logi = LogisticRegression(solver = 'lbfgs').fit(X,y)
```

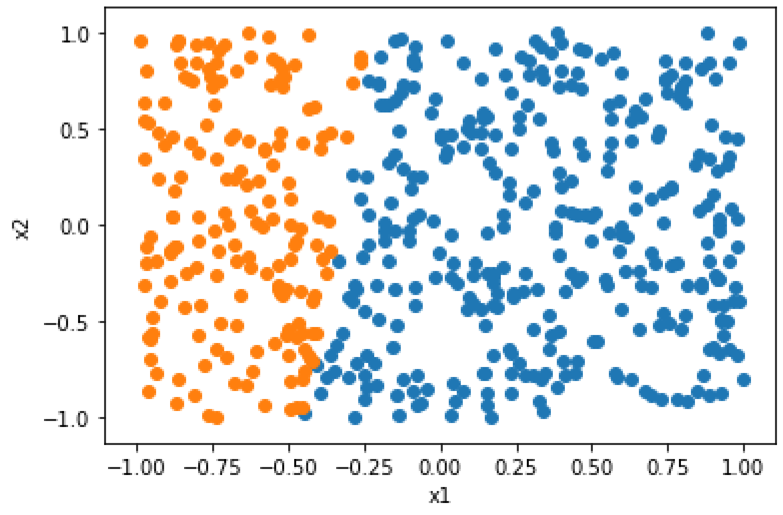
1. (5 points) Obtain a predicted class label for each observation based on the logistic model previously fit. Plot the observations, colored according to the predicted class labels (the predicted decision boundary should look linear).

In [104]:

```
y_pred = logi.predict(X)
plt.scatter(x1[y_pred],x2[y_pred])
plt.scatter(x1[~y_pred],x2[~y_pred])
plt.xlabel('x1')
plt.ylabel('x2')
```

Out[104]:

```
Text(0, 0.5, 'x2')
```



1. (5 points) 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 [105]:

```
logi2 = LogisticRegression(solver = 'lbfgs').fit((X**2),y)
```

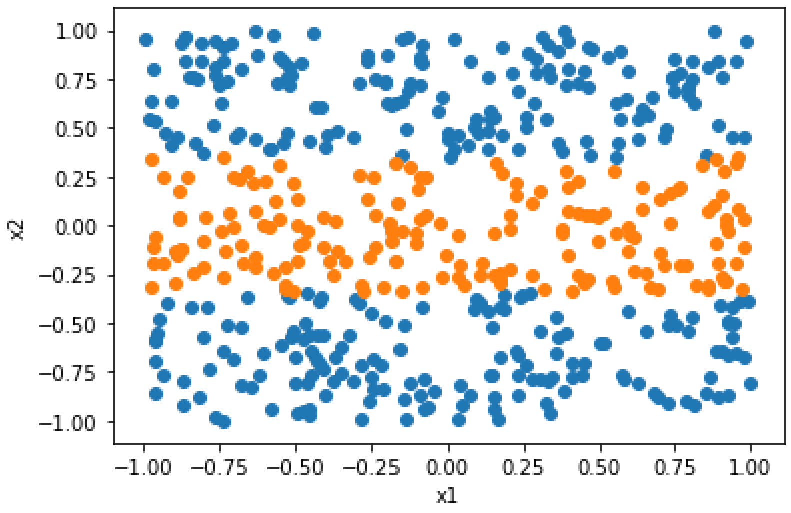
1. (5 points) Now, obtain a predicted class label for each observation based on the fitted model with non-linear transformations of the X features in the previous question. Plot the observations, colored according to the new predicted class labels from the non-linear model (the decision boundary should 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 [106]:

```
y_pred = logi2.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')
```

Out[106]:

```
Text(0, 0.5, 'x2')
```



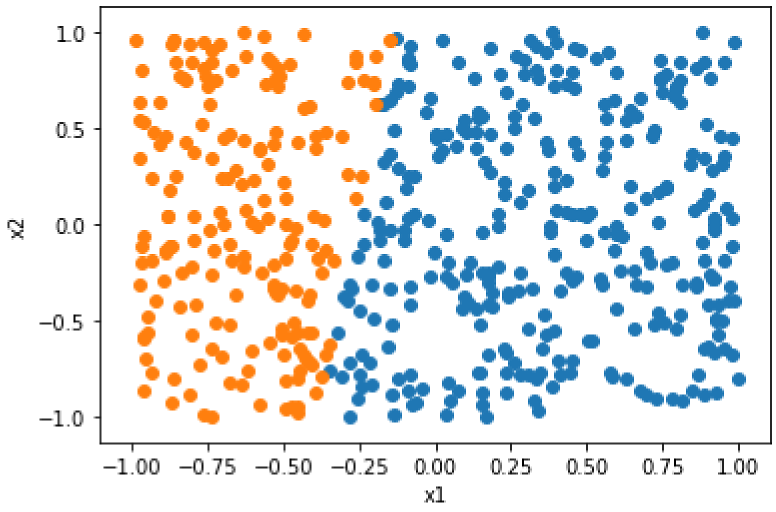
1. (5 points) Now, fit a support vector classifier (linear kernel) to the data with original X1 and X2 as predictors. Obtain a class prediction for each observation. Plot the observations, colored according to the predicted class labels.

In [107]:

```
from sklearn import svm
svm_linear = svm.SVC(gamma = 'auto',kernel = 'linear').fit(X,y)
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')
```

Out[107]:

```
Text(0, 0.5, 'x2')
```



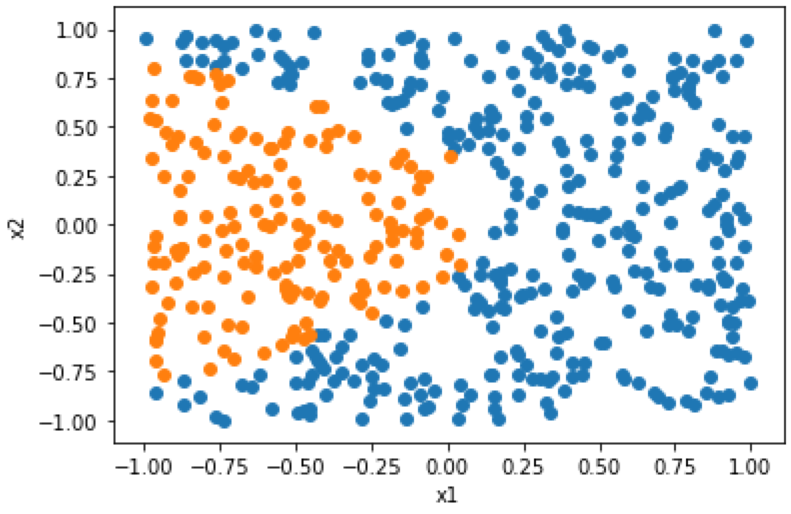
1. (5 points) Fit a SVM using a non-linear kernel to the data. Obtain a class prediction for each observation. Plot the observations, colored according to the predicted class labels.

In [108]:

```
svm_nonlinear = svm.SVC(gamma = 'auto').fit(X,y)
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')
```

Out[108]:

```
Text(0, 0.5, 'x2')
```



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

In [110]:

```
print('logistic regression(linear predictors) accurace',logi.score(X,y))
print('logistic regression(nonlinear predictors) accurace',logi2.score(X**2,y))
print('SVM(linear kernel) accurace',svm_linear.score(X,y))
print('SVM(nonlinear kernel) accurace',svm_nonlinear.score(X,y))
```

```
logistic regression(linear predictors) accurace 0.69
logistic regression(nonlinear predictors) accurace 0.64
SVM(linear kernel) accurace 0.672
SVM(nonlinear kernel) accurace 0.73
```

Radical kernel SVM outperforms the other three models. It is not surprising that SVM with radical kernel performs the best when it comes to nonlinear relationship. logistic regression with nonlinear predictor is worse than losgistic regression with linear predictors because X**2 is far from the true relationship that is quartric

Tuning cost

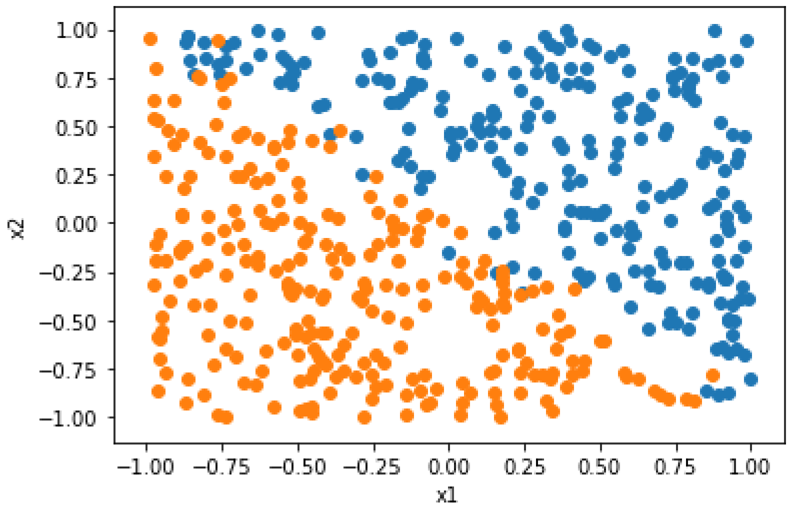
1. (5 points) Generate two-class data with p = 2 in such a way that the classes are just barely linearly separable.

In [112]:

```
np.random.seed(0)
x1 = np.random.uniform(-1,1,500)
x2 = np.random.uniform(-1,1,500)
e = np.random.normal(0,0.1,500)
y = x1+x2+e
y_prob = np.exp(y)/(1+np.exp(y))
plt.scatter(x1[y_prob>=0.5],x2[y_prob>=0.5])
plt.scatter(x1[y_prob<0.5],x2[y_prob<0.5])
plt.xlabel('x1')
plt.ylabel('x2')
```

Out[112]:

```
Text(0, 0.5, 'x2')
```



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 [113]:

```
X = np.column_stack((x1,x2))
y = (y_prob>=0.5)
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size = 0.2)
```

In [176]:

```
import warnings
warnings.filterwarnings("ignore", category=DeprecationWarning)
warnings.filterwarnings("ignore", category=FutureWarning)
```

In [186]:

```
param_grid = [{'C': np.linspace(0.001,10,10)}]
clf = GridSearchCV(svm.SVC(gamma = 'auto'),
                   param_grid, scoring = 'accuracy',cv = 10,return_train_score=T
rue)
clf.fit(X_train, y_train)
```

Out[186]:

```
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', kernel='rbf', max_iter=-1,
                           probability=False, random_state=None, shr
inking=True,
                           tol=0.001, verbose=False),
             iid='warn', n_jobs=None,
             param_grid=[{'C': array([1.000e-03, 1.112e+00, 2.223e+0
0, 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)
```

In [187]:

```
print('The models accuracy on train set is',clf.cv_results_['mean_train_score'])
print('the models accuracy on cross-validation set is',clf.cv_results_['mean_tes
t_score'])
```

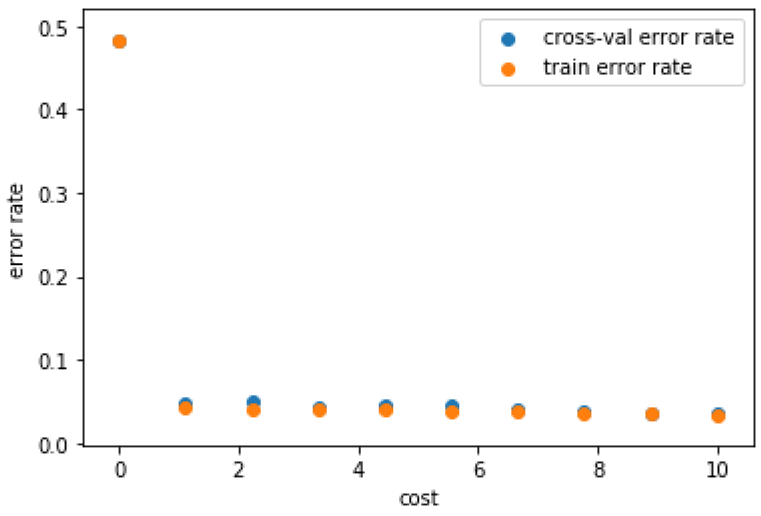
```
The models accuracy on train set is [0.51750008 0.95722049 0.9608316
2 0.96083084 0.96055152 0.96194196
 0.96333008 0.96471897 0.96443965 0.96582932]
the models accuracy on cross-validation set is [0.5175 0.9525 0.95
0.9575 0.955  0.955  0.96   0.9625 0.965  0.965 ]
```

In [193]:

```
plt.scatter(np.linspace(0,10,10),1-clf.cv_results_['mean_test_score'])
plt.scatter(np.linspace(0,10,10),1-clf.cv_results_['mean_train_score'])
plt.legend({'cross-val error rate','train error rate'})
plt.xlabel('cost')
plt.ylabel('error rate')
```

Out[193]:

```
Text(0, 0.5, 'error rate')
```

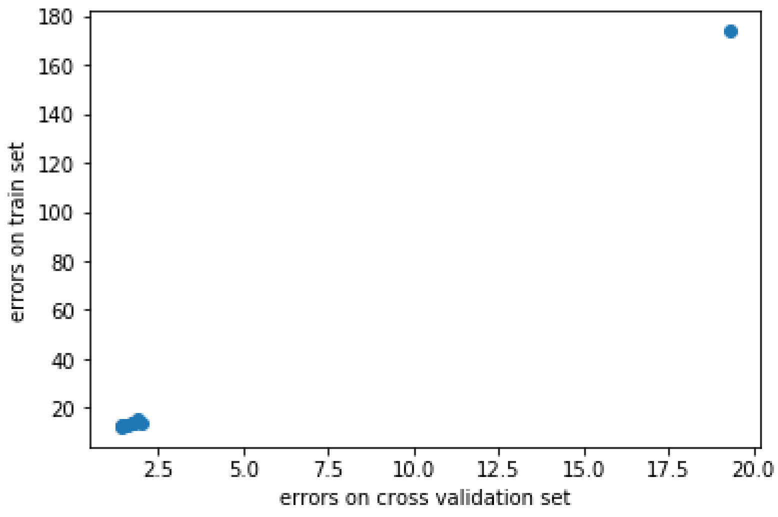


In [189]:

```
plt.scatter(500*0.8*0.1*(1-clf.cv_results_['mean_test_score']),500*0.8*0.9*(1-cl
f.cv_results_['mean_train_score']))
plt.xlabel('errors on cross validation set')
plt.ylabel('errors on train set')
```

Out[189]:

```
Text(0, 0.5, 'errors on train set')
```



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 [194]:

```
test_error = []
test_score = []
for item in clf.cv_results_['param_C']:
    model = svm.SVC(C = item).fit(X_train,y_train)
    test_score.append(model.score(X_test,y_test))
    test_error.append(500*0.2*(1-model.score(X_test,y_test)))
print(test_score)
print('The test accuracy is 1 when the cost is around 1')
```

```
[0.5, 1.0, 0.99, 0.99, 0.99, 0.99, 0.99, 0.99, 0.99, 0.99]
The test accuracy is 1 when the cost is around 1
```

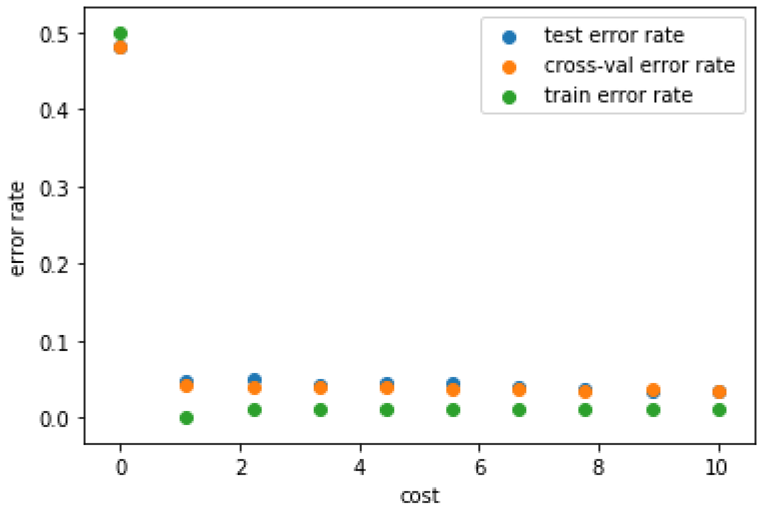
1. (5 points) Discuss your results.

In [192]:

```
plt.scatter(np.linspace(0,10,10),1-clf.cv_results_['mean_test_score'])
plt.scatter(np.linspace(0,10,10),1-clf.cv_results_['mean_train_score'])
plt.scatter(np.linspace(0,10,10),1-np.asarray(test_score))
plt.legend({'cross-val error rate','train error rate','test error rate'})
plt.xlabel('cost')
plt.ylabel('error rate')
```

Out[192]:

```
Text(0, 0.5, 'error rate')
```



we can see from the above graph, the test error is minimized when cost is around 1. This suggests that a support vector classifier with a small value of cost may perform better on test test than one with huge value of cost that is around 10.

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 [195]:

```
df_train = pd.read_csv('/Users/lijiaxuan/Downloads/problem-set-6-master/data/gss
_train.csv')
df_test = pd.read_csv('/Users/lijiaxuan/Downloads/problem-set-6-master/data/gss_
test.csv')
```

In [198]:

```
X_train, y_train = df_train.loc[:, df_train.columns != 'colrac'],df_train['colra
c']
X_test, y_test = df_test.loc[:, df_test.columns != 'colrac'],df_test['colrac']
```

In [199]:

```
param_grid = [{'C': np.linspace(0.001,10,10)}]
clf = GridSearchCV(svm.SVC(gamma = 'auto'),
                   param_grid, scoring = 'accuracy',cv = 10,return_train_score=T
rue)
clf.fit(X_train, y_train)
```

Out[199]:

```
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', kernel='rbf', max_iter=-1,
                           probability=False, random_state=None, shr
inking=True,
                           tol=0.001, verbose=False),
             iid='warn', n_jobs=None,
             param_grid=[{'C': array([1.000e-03, 1.112e+00, 2.223e+0
0, 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)
```

In [200]:

```
clf.best_estimator_
```

Out[200]:

```
SVC(C=2.223, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='auto', kernel='r
bf',
    max_iter=-1, probability=False, random_state=None, shrinking=Tru
e,
    tol=0.001, verbose=False)
```

In [201]:

```
print('the models error rate on cross-validation set is',1-clf.cv_results_['mean
_test_score'])
```

```
the models error rate on cross-validation set is [0.47467927 0.26806
212 0.26333558 0.26468602 0.2640108  0.2640108
 0.2640108  0.2640108  0.2640108  0.2640108 ]
```

The model performs the best when cost is around 2.2. The CV errors drops when cost increases from 0.001 to around 1 and remains steady from 1 to 10.

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 [225]:

```
param_grid = [{'kernel':('rbf','poly','linear'),'C': np.linspace(0.1,5,5),
              'gamma':('scale','auto'),'degree':[1,3,5]}]
clf = GridSearchCV(svm.SVC(),
                   param_grid, scoring = 'accuracy',cv = 3,return_train_score=Tr
ue)
clf.fit(X_train, y_train)
```

In [239]:

```
clf.best_estimator_
```

Out[239]:

```
SVC(C=2.5500000000000003, cache_size=200, class_weight=None, coef0=
0.0,
    decision_function_shape='ovr', degree=3, gamma='auto', kernel='l
inear',
    max_iter=-1, probability=False, random_state=None, shrinking=Tru
e,
    tol=0.001, verbose=False)
```

In [243]:

```
best_svm = svm.SVC(gamma = 'auto',kernel = 'linear',degree = 3,C = 2.55).fit(X_t
rain,y_train)
```

In [244]:

```
best_svm.score(X_test,y_test)
```

Out[244]:

```
0.7809330628803245
```

the model performs the best when degree equals to 3, gamma is auto, use radical kernel and the cost is 10. the model accuracy is 0.78. Generally, linear kernel outperforms radical and polynomial kernels, and the model's cost is best at around 1.1.