modelhw6

March 8, 2020

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import svm
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.model_selection import cross_val_score, KFold
from skater.model import InMemoryModel
from sklearn.metrics import accuracy_score

import warnings
warnings.filterwarnings('ignore')
```

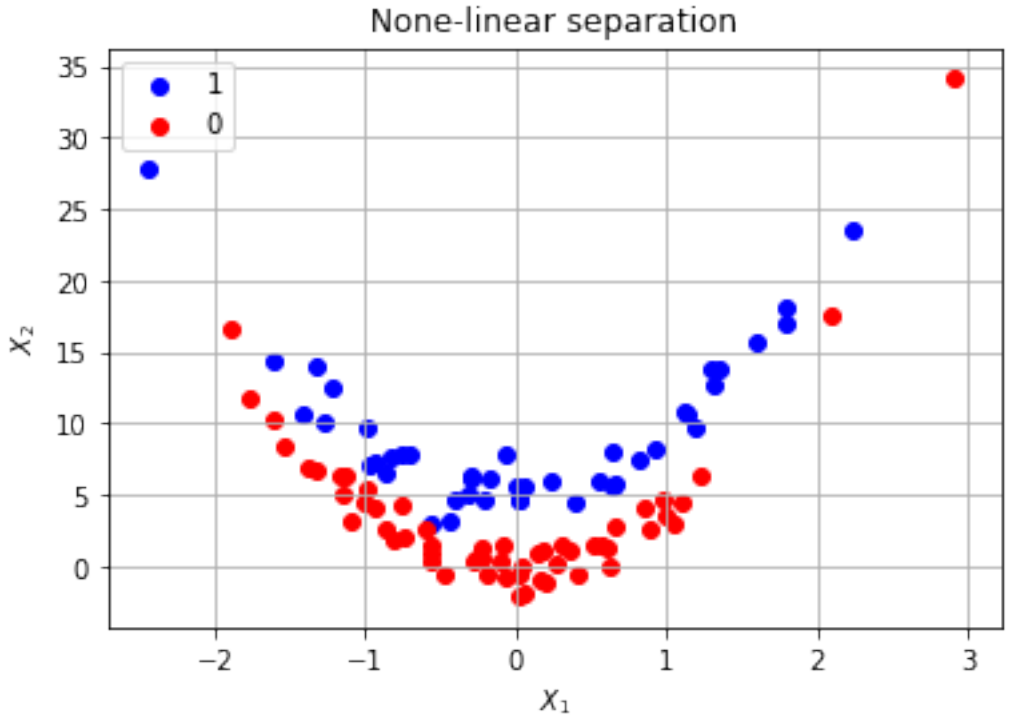
0.1 Conceptual exercises

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.

[8]:
```
np.random.seed(1988)

#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.
x1 = np.random.randn(100)
x2 = 4*x1**2 + np.random.randn(100)
label = np.random.choice(2, size=100)
x2[label==1] = x2[label==1]+5

plt.scatter(x1[label==1], x2[label==1], c='blue', label="1")
plt.scatter(x1[label==0], x2[label==0], c='red', label="0")
plt.xlabel(r'$X_1$')
plt.ylabel(r'$X_2$')
plt.legend()
```

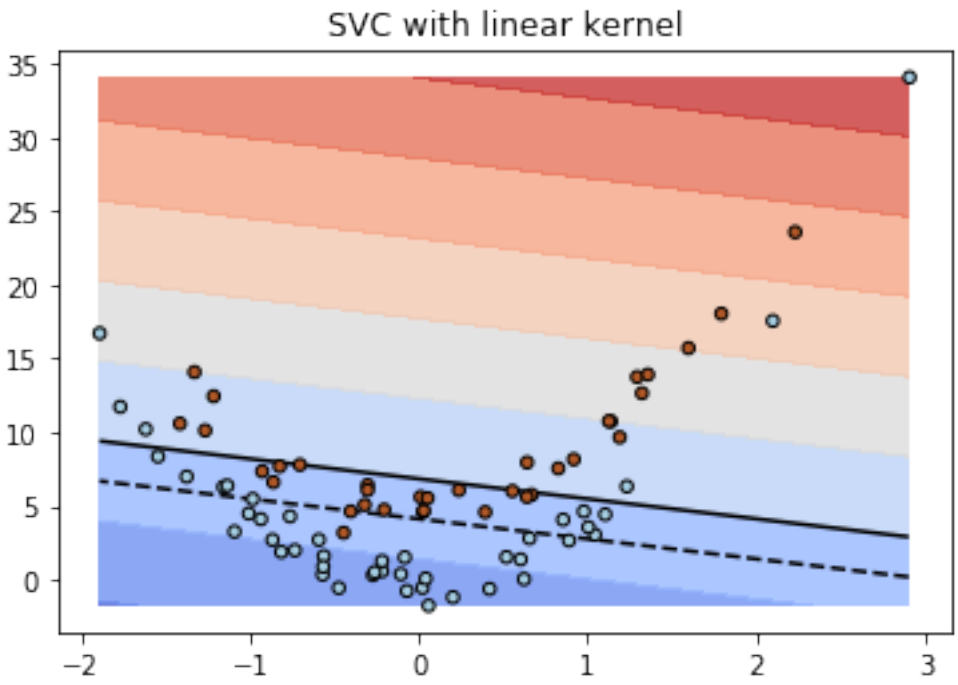
```
plt.grid()
plt.title('None-linear separation')
plt.show()
```

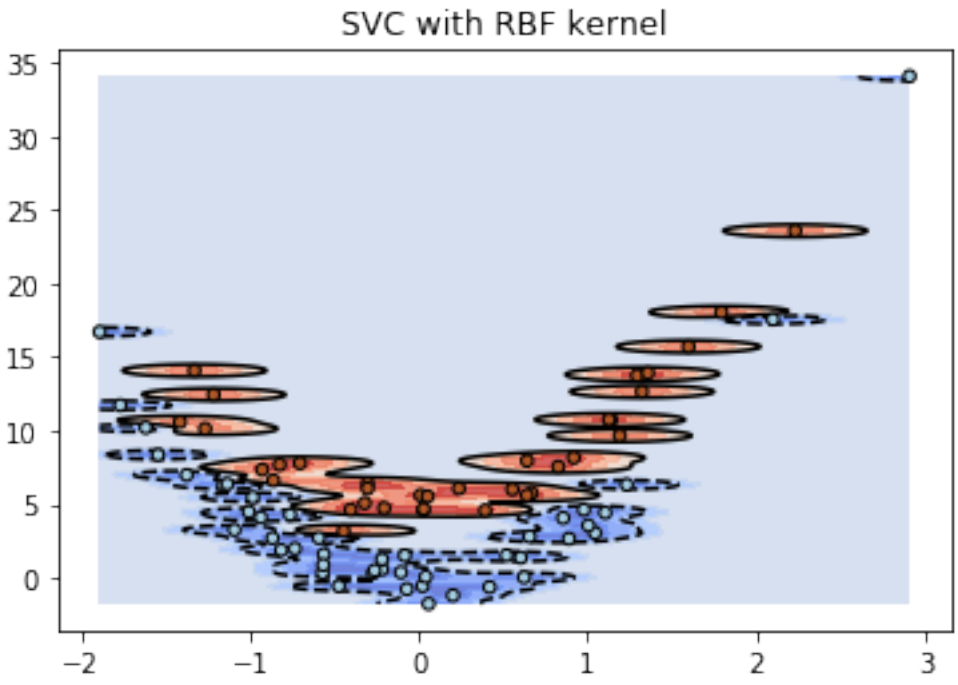


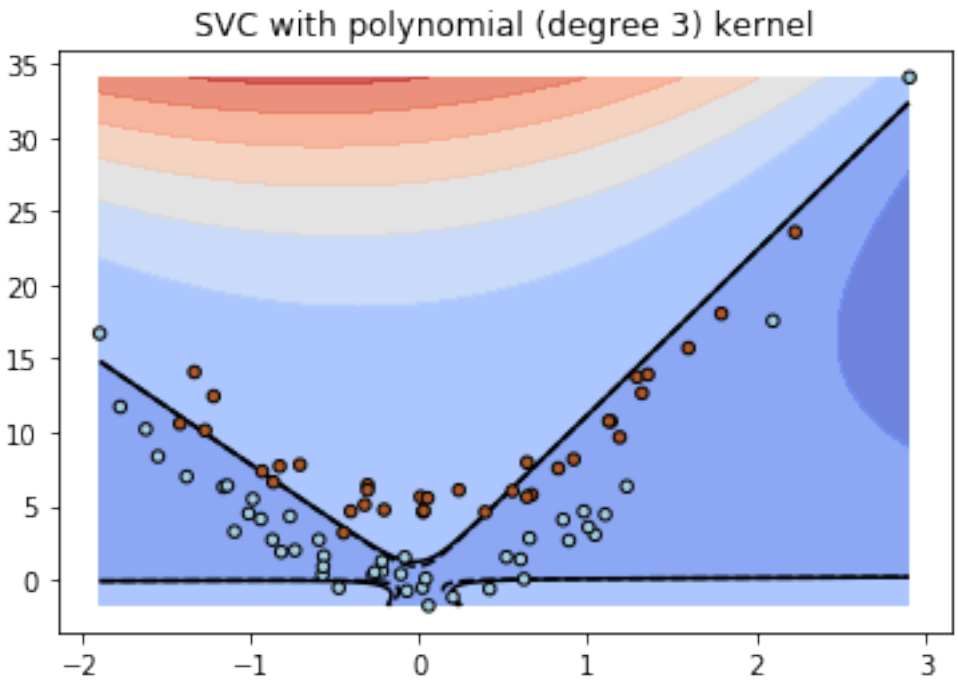
[20]:
```
#Show that in this setting,
#a support vector machine with a radial kernel will outperform a support vector␣
 ↪classifier
#(a linear kernel) on the training data.
x = np.vstack((x1, x2)).T
y = label

x_train = x[:80]
y_train = y[:80]
x_test = x[80:]
y_test = y[80:]
```

[33]:
```
# fit the model
C = 1.0  # SVM regularization parameter
models = (svm.SVC(kernel='linear', C=C),  #a linear kernel
          svm.SVC(kernel='rbf', gamma=10, C=C),#radial kernel
          svm.SVC(kernel='poly', degree=3, gamma=10, C=C))#polynomial (degree␣
 ↪3) kernel
```

```
# title for the plots
titles = ('SVC with linear kernel',
          'SVC with RBF kernel',
          'SVC with polynomial (degree 3) kernel')

# plot the models
for idx, clf in enumerate(models):
    clf.fit(x_train, y_train)

    plt.figure(idx)
    plt.clf()
    plt.scatter(x_train[:,0], x_train[:,1], c=y_train, zorder=10,
                cmap=plt.cm.Paired, edgecolor='k', s=20)
    plt.axis('tight')
    x_min = x_train[:, 0].min()
    x_max = x_train[:, 0].max()
    y_min = x_train[:, 1].min()
    y_max = x_train[:, 1].max()

    xx, yy = np.mgrid[x_min:x_max:0.01, y_min:y_max:0.01]
    Z = clf.decision_function(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    plt.contourf(xx, yy, Z, cmap=plt.cm.coolwarm, alpha=0.8)
    plt.contour(xx, yy, Z, colors=['k', 'k'],
                linestyles=['--', '-'], levels=[-.5, 0])
    plt.title(titles[idx])

plt.show()
```


SVC with linear kernel
35
30
25
20
15
10
5
0
−2
−1
0
1
2
3


SVC with RBF kernel
35
30
25
20
15
10
5
0
−2
−1
0
1
2
3



[35]:
```
# error on the training data
model_lst = [clf.fit(x_train, y_train) for clf in models]
print('The training error rate of SVC with linear is {:.2%}'.format(1-␣
 ↪model_lst[0].score(x_train, y_train)))
print('The training error rate of SVC with radial is {:.2%}'.format(1-␣
 ↪model_lst[1].score(x_train, y_train)))
print('The training error rate of SVC with polynomial is {:.2%}'.format(1-␣
 ↪model_lst[2].score(x_train, y_train)))
```

```
The training error rate of SVC with linear is 31.25%
The training error rate of SVC with radial is 0.00%
The training error rate of SVC with polynomial is 27.50%
```

[36]:
```
# error on the testing data
preds = [clf.predict(x_test) for clf in model_lst]
print('The test error rate of SVC with linear is {:.2%}'.
 ↪format(1-accuracy_score(y_test, preds[0])))
print('The test error rate of SVC with radial is {:.2%}'.
 ↪format(1-accuracy_score(y_test, preds[1])))
print('The test error rate of SVC with polynomial is {:.2%}'.
 ↪format(1-accuracy_score(y_test, preds[2])))
```

```
The test error rate of SVC with linear is 20.00%
The test error rate of SVC with radial is 20.00%
The test error rate of SVC with polynomial is 15.00%
```

According to the plots and the error rates, polynomial (degree 3) kernel performs the best. radial kernel performs well on trainings set but much worse than polynomial kernel on the test set, owing to its overfitting on training data. linear kernel performs the worst.

0.1.2 SVM vs. logistic regression

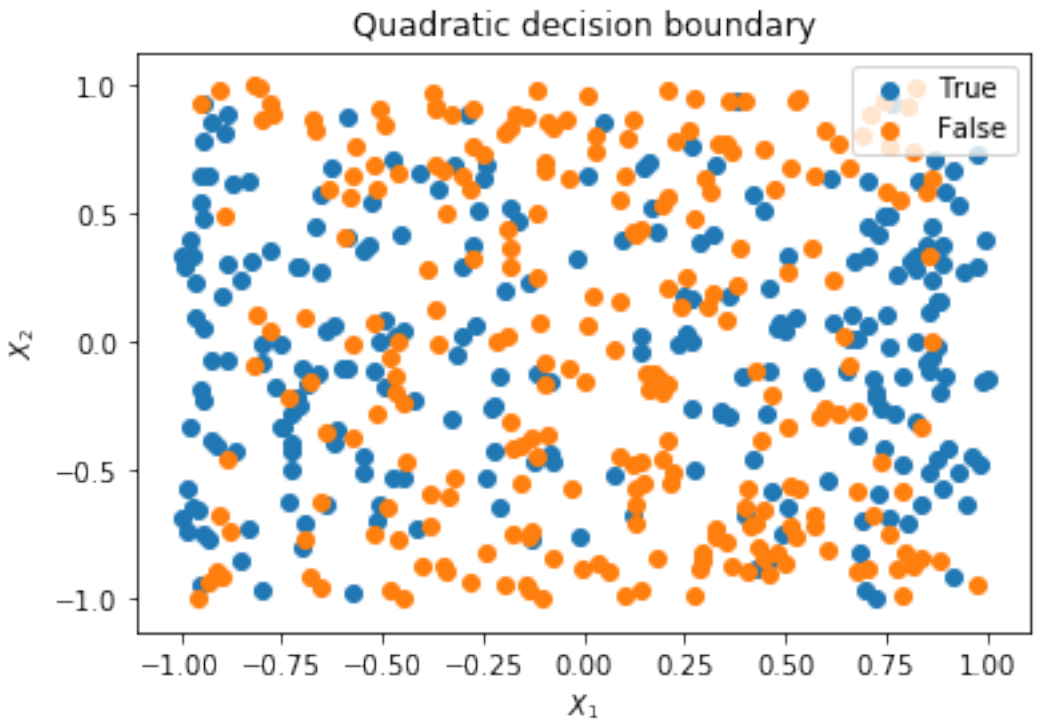
We have seen that we can fit an SVM with a non-linear kernel in order to perform classification using a non-linear decision boundary. We will now see that we can also obtain a non-linear decision boundary by performing logistic regression using non-linear transformations of the features. Your goal here 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.

[44]:
```
x1 = np.random.uniform(-1, 1, 500)
x2 = np.random.uniform(-1, 1, 500)
y = x1 ** 2 - x2 ** 2 + np.random.uniform(-1,1,500) > 0
```

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.

[45]:
```
plt.scatter(x1[y.astype(bool)], x2[y.astype(bool)], label='True')
plt.scatter(x1[~y.astype(bool)], x2[~y.astype(bool)], label='False')
plt.xlabel(r'$X_1$')
plt.ylabel(r'$X_2$')
plt.title('Quadratic decision boundary')
plt.legend()
plt.show()
```

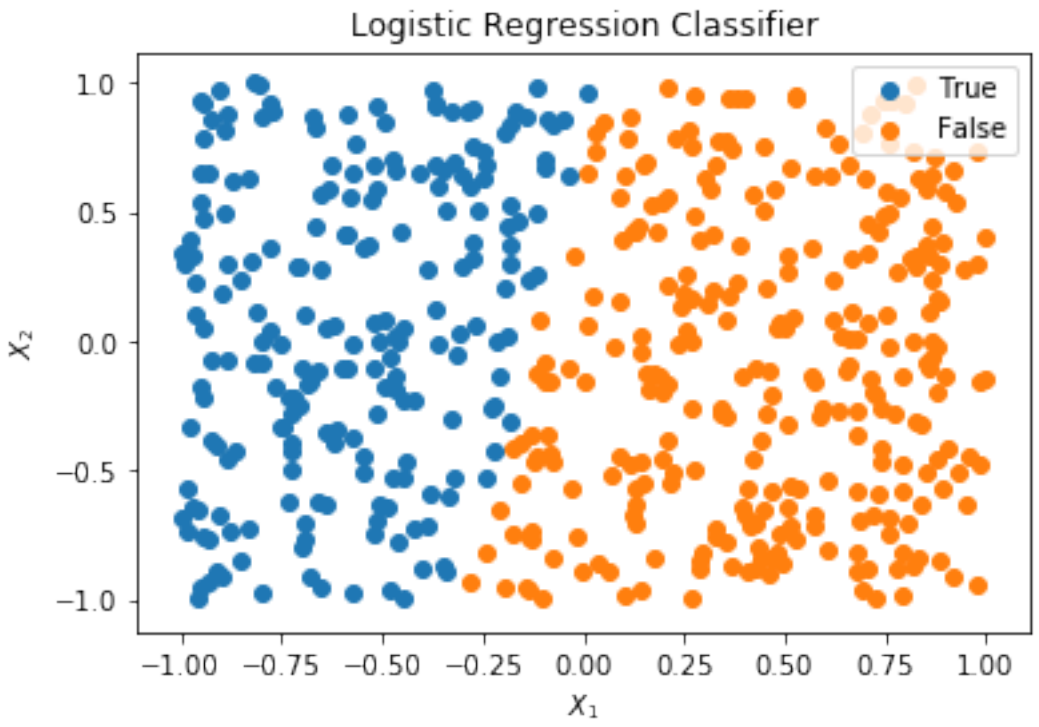


4. Fit a logistic regression model to the data, using X_1 and X_2 as predictors.

[51]:
```
# fit logistic regression
X = np.vstack((x1, x2)).T
logit = LogisticRegression().fit(X, y)
```

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).

[53]:
```
logit_pred = logit.predict(X)
plt.scatter(x1[logit_pred.astype(bool)], x2[logit_pred.astype(bool)],␣
 ↪label='True')
plt.scatter(x1[~logit_pred.astype(bool)], x2[~logit_pred.astype(bool)],␣
 ↪label='False')
plt.xlabel(r'$X_1$')
plt.ylabel(r'$X_2$')
plt.title('Logistic Regression Classifier')
plt.legend()
plt.show()
```



Obviously the decision boundary is linear.

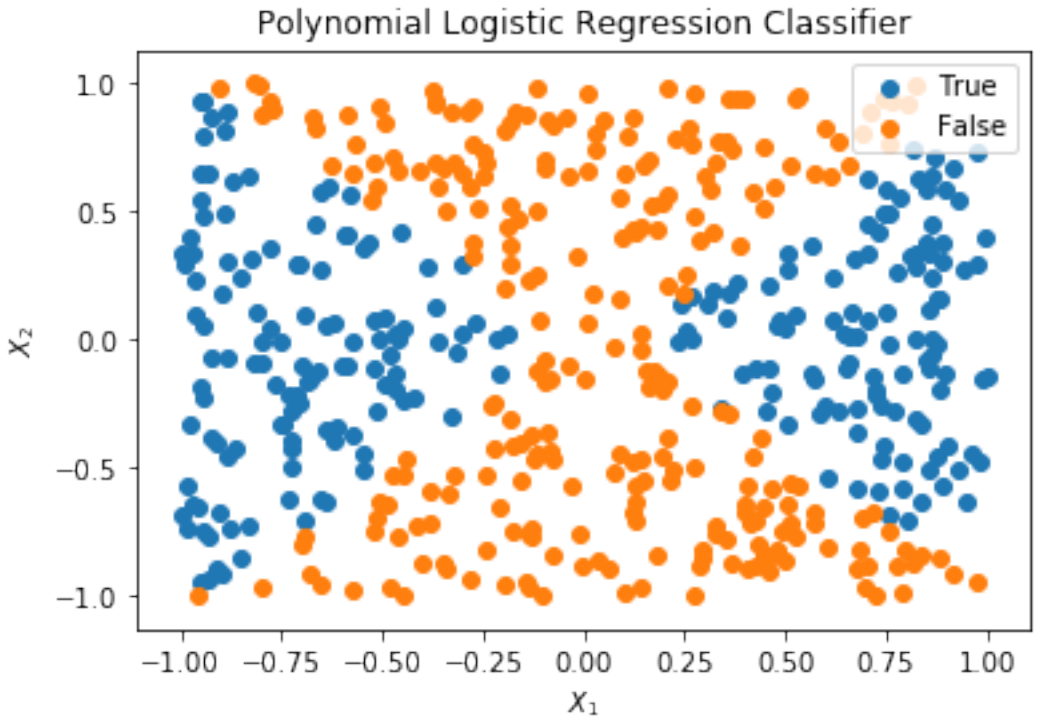
6. Now fit a logistic regression model to the data, but this time using some non-linear function of both X_1 and X_2 as predictors (e.g. $X_1^2, X_1 \times X_2, \log(X_2)$, and so on).

[54]:
```
# fit polynomial logistic regression
tf_X = PolynomialFeatures(2).fit_transform(X)
logit_poly = LogisticRegression().fit(tf_X, y)
```

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.

[55]:
```
poly_pred = logit_poly.predict(tf_X)
plt.scatter(x1[poly_pred.astype(bool)], x2[poly_pred.astype(bool)],␣
 ↪label='True')
plt.scatter(x1[~poly_pred.astype(bool)], x2[~poly_pred.astype(bool)],␣
 ↪label='False')
plt.title('Polynomial Logistic Regression Classifier')
plt.xlabel(r'$X_1$')
plt.ylabel(r'$X_2$')
```

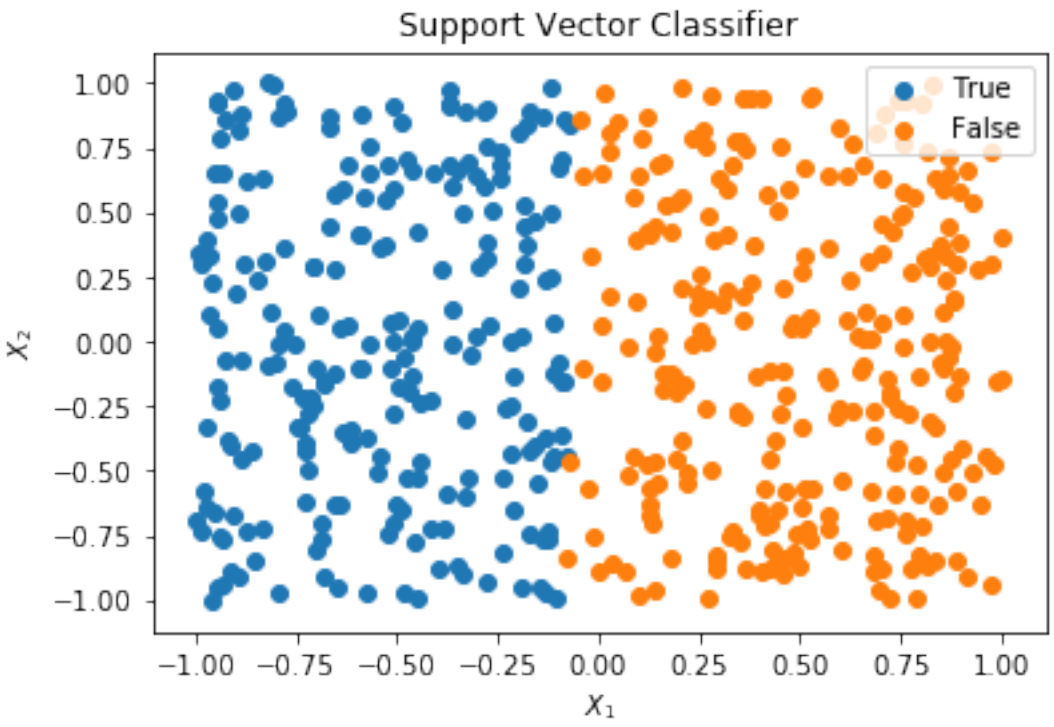
```
plt.legend()
plt.show()
```



Obviously the decision boundary is nonlinear

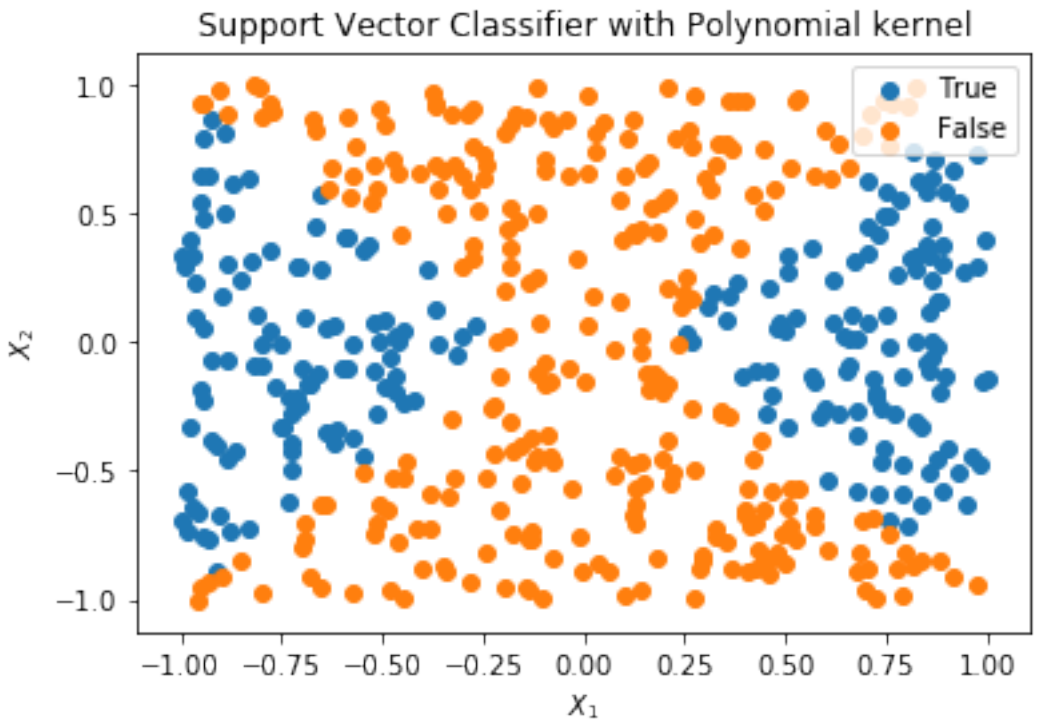
8. Now, fit a support vector classifier (linear kernel) to the data with original X_1 and X_2 as predictors. Obtain a class prediction for each observation. Plot the observations, colored according to the predicted class labels.

[56]:
```
# fit linear SVC
linear_svm = svm.SVC(kernel='linear').fit(X, y)
svm_pred = linear_svm.predict(X)

#plot
plt.scatter(x1[svm_pred.astype(bool)], x2[svm_pred.astype(bool)], label='True')
plt.scatter(x1[~svm_pred.astype(bool)], x2[~svm_pred.astype(bool)],␣
 ↪label='False')
plt.title('Support Vector Classifier')
plt.xlabel(r'$X_1$')
plt.ylabel(r'$X_2$')
plt.legend()
plt.show()
```



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.

[57]:
```
# fit SVC with polynomial kernel
poly_svm = svm.SVC(kernel='poly', degree=2).fit(X, y)
poly_svm_pred = poly_svm.predict(X)

plt.scatter(x1[poly_svm_pred.astype(bool)], x2[poly_svm_pred.astype(bool)],␣
↪label='True')
plt.scatter(x1[~poly_svm_pred.astype(bool)], x2[~poly_svm_pred.astype(bool)],␣
↪label='False')
plt.title('Support Vector Classifier with Polynomial kernel')
plt.xlabel(r'$X_1$')
plt.ylabel(r'$X_2$')
plt.legend()
plt.show()
```



10. Discuss your results and specifically the tradeoffs between estimating non-linear decision boundaries using these two different approaches. According to the results above, with the real decision boundary as quadratic, the logistic model and SVM model with linear kernel would fail to detect the boundary, while non-linear logistic regression and SVM with non-linear kernel perform much better. However, adding extra features to the logistic regression is computationally intensive and would not be a good idea when the feature space is very high-dimensional. Therefore, we should prefer the SVM model, which has only very few parameters to tune up and works more efficiently.

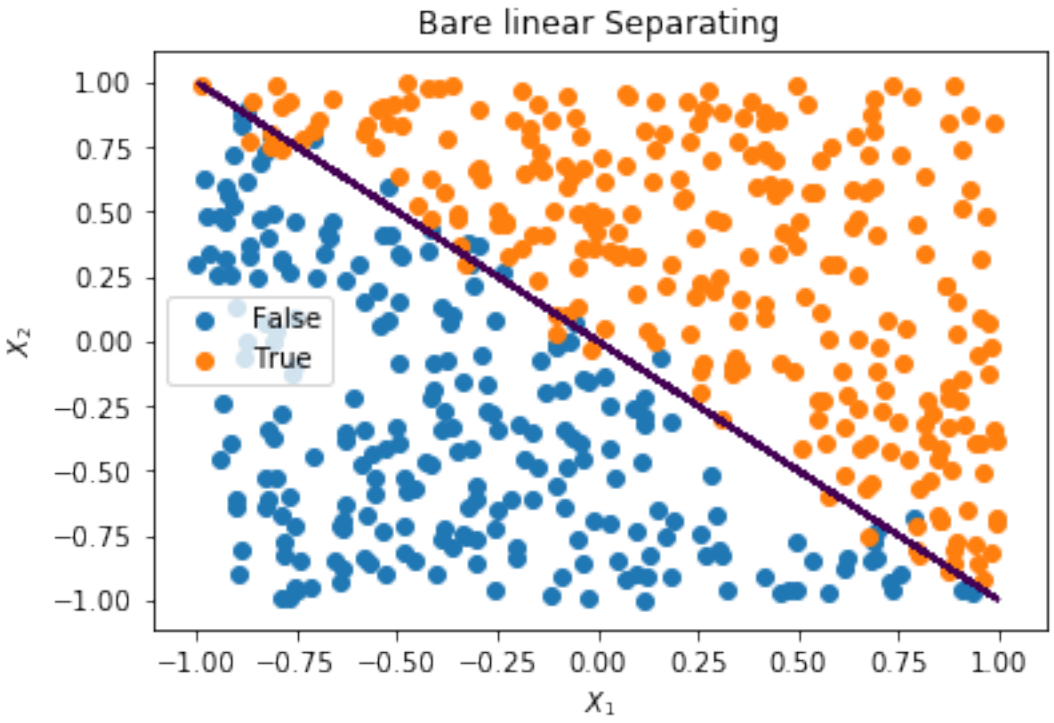
0.1.3 Tuning cost

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.

[58]:
```
x1 = np.random.uniform(-1, 1, 500)
x2 = np.random.uniform(-1, 1, 500)
y = (x1 + x2 + np.random.normal(0, 0.1, 500)) > 0
X = np.array([x1, x2]).T
```

```
fig, ax = plt.subplots()
for group in np.unique(y):
    i = np.where(y == group)
    ax.scatter(x1[i], x2[i], label=group)

ax.legend()
XX, YY =np.meshgrid(x1, x2)
plt.contour(XX, YY, XX+YY, [0])
plt.title('Bare linear Separating')
plt.xlabel(r'$X_1$')
plt.ylabel(r'$X_2$')
plt.show()
```



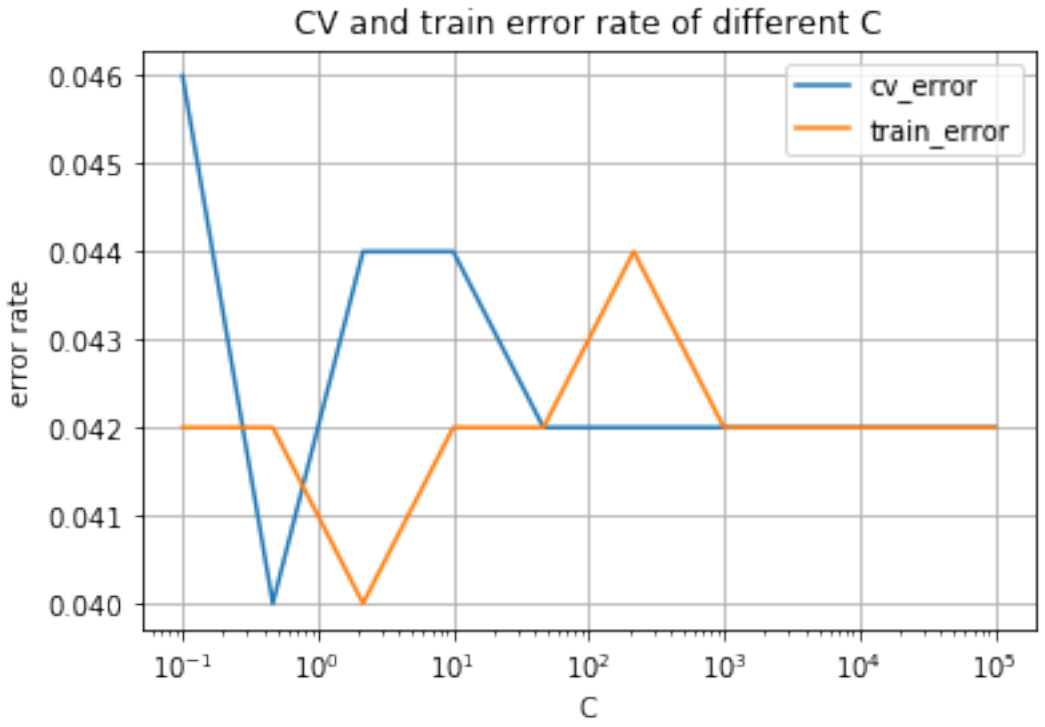
Obviously the classes are linearly separable

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?

[64]:
```
Cs = np.logspace(-1, 5, 10)  # a range of cost values
svm_lst = [svm.SVC(kernel='linear', C=c) for c in Cs]
#cross-validation error rates
svm_cv_errs = [1 - np.mean(cross_val_score(model, X, y, cv=KFold(10),
                                scoring='accuracy')) for model in svm_lst]
```

```
#training errors
svm_train_errs = [1 - model.fit(X,y).score(X, y) for model in svm_lst]
```

[65]:
```
plt.plot(Cs, svm_cv_errs, label="cv_error")
plt.plot(Cs, svm_train_errs,label='train_error')
plt.xscale('log')
plt.title('CV and train error rate of different C')
plt.xlabel('C')
plt.ylabel('error rate')
plt.legend()
plt.grid()
plt.show()
print('CV error rate reaches its minimum {:.3f} with C={:.1f} '.format(np.
 ↪min(svm_cv_errs), Cs[np.argmin(svm_cv_errs)]))
print('Train error rate reaches its minimum {:.3f} with C={:.1f} '.format(np.
 ↪min(svm_train_errs), Cs[np.argmin(svm_train_errs)]))
```



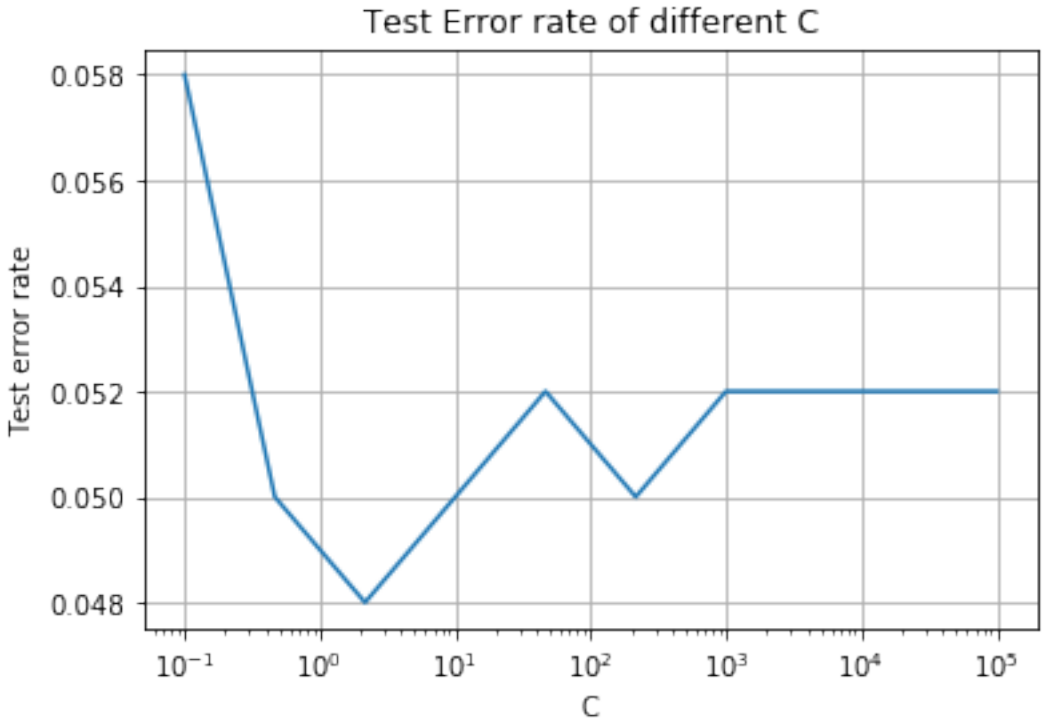

```
CV error rate reaches its minimum 0.040 with C=0.5
Train error rate reaches its minimum 0.040 with C=2.2
```

The smallest cv error rate and training error rate are the same–0.040, but cv error reaches its minimum with a smaller cost value of 0.5, while training error hits its minimum with c=2.2. They both increase and decrease drastically after hitting the smallest error rate and remain stable at the same error rate(0.042) after passing a certain cost value.

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?

[66]:
```
# test data set
test_x1 = np.random.uniform(-1, 1, 500)
test_x2 = np.random.uniform(-1, 1, 500)
test_X = np.vstack((test_x1, test_x2)).T
test_y = (test_x1 + test_x2) + np.random.normal(0, 0.1, 500) > 0
```

[68]:
```
# test error rate for SVMs
svm_test_errs =[1 - model.fit(X,y).score(test_X, test_y) for model in svm_lst]
plt.plot(Cs, svm_test_errs)
plt.xscale('log')
plt.xlabel('C')
plt.ylabel('Test error rate')
plt.title('Test Error rate of different C')
plt.grid()
plt.show()
print('Test error rate reaches its minimum {:.3f} with C={:.1f} '.format(np.
 ↪min(svm_test_errs), Cs[np.argmin(svm_test_errs)]))
```




```
Test error rate reaches its minimum 0.048 with C=2.2
```

14. Discuss your results The test errors plot shares similar changing trend with train error rate. They reach minimum error rate with the same C=2.2, larger than the cv error one(C=0.5), and after the minimum, both test error and train error pick up, decrease and finally remain stable. The noticeable difference is that training error and test error change in different direction in the C range of 10-1000. When train error reaches local maximum, test error hits local minimum. This could be explained by the overfitting problem when training error is low, so when train error increases, the model actually fits the test set better and improves its performance. And a larger cost value leading to the decline of train error, on the other hand, makes test error larger owing to the problem of overfitting.

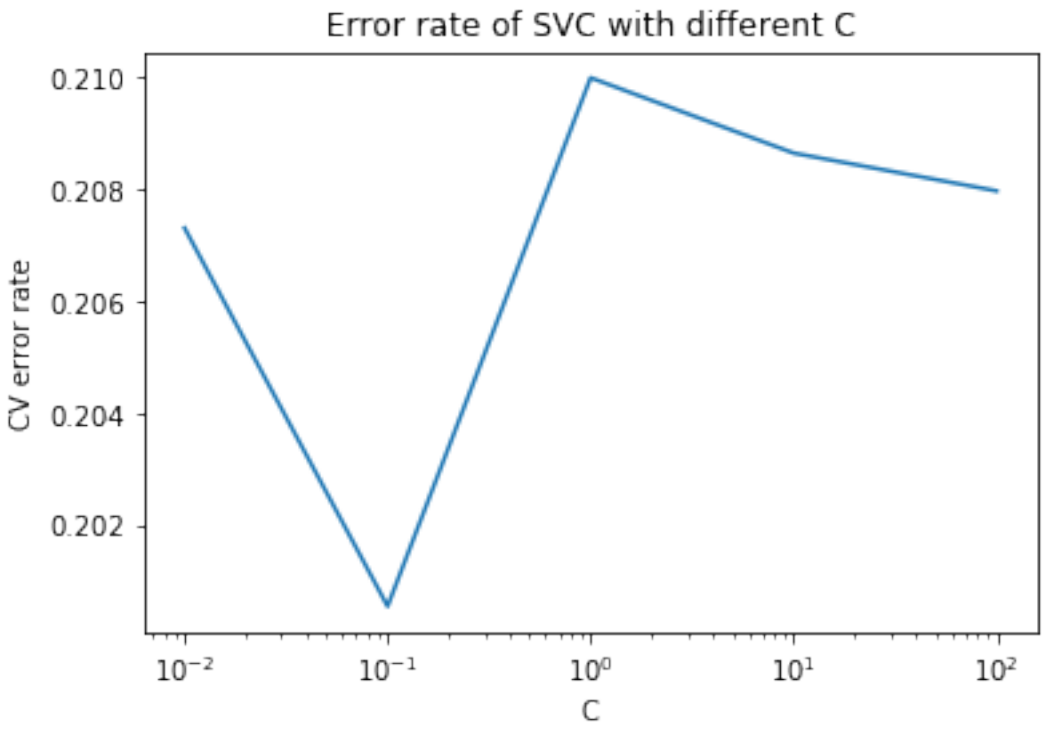
0.2 Predicting attitudes towards racist college professors

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.

[2]:
```
# load data
gss_train = pd.read_csv('gss_train.csv')
gss_test = pd.read_csv('gss_test.csv')

# training/testing dataset
x_train = gss_train.drop(['colrac'], axis=1).values
y_train = gss_train['colrac'].values
x_test = gss_test.drop(['colrac'], axis=1).values
y_test = gss_test['colrac'].values
```

[3]:
```
# calculate cv error rates based on different cs
Cs = np.logspace(-2, 2, 5)
svm_lst = [svm.SVC(kernel='linear', C=c) for c in Cs]
svm_cv_errs = [1 -np.mean(cross_val_score(model, x_train, y_train,␣
 ↪cv=KFold(10), scoring='accuracy')) for model in svm_lst]
```

[6]:
```
#plot error rates
plt.plot(Cs, svm_cv_errs)
plt.xscale('log')
plt.title('Error rate of SVC with different C')
plt.xlabel('C')
plt.ylabel('CV error rate')
plt.show()
print('Error rate of optimal C: {:.5f}'.format(min(svm_cv_errs)))
```



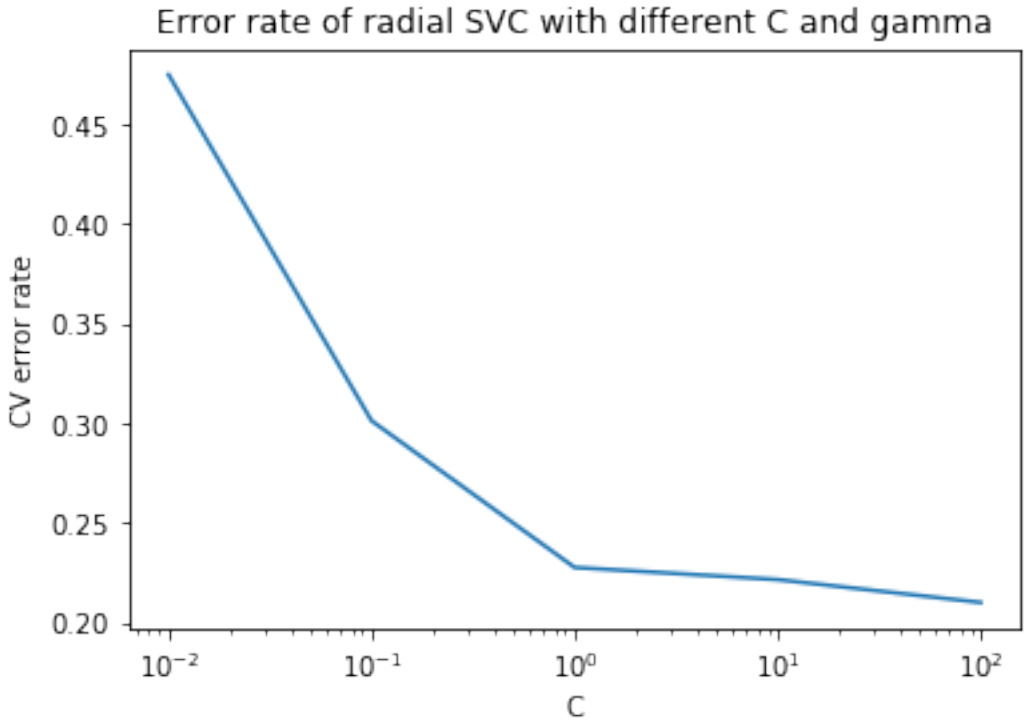

```
Error rate of optimal C: 0.20055
```

From the linear kernel, we can find that the cv error hits the minimum at C=0.2, and then gets larger as C increases

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.).

[7]:
```
# radial kernel with different Cs and gammas
Cs = np.logspace(-2, 2, 5)
gammas = np.logspace(-5, 0, 5)
cv_error = []
opt_gamma = []
for c in Cs:
    svm_lst = [svm.SVC(kernel='rbf', C=c, gamma = gamma) for gamma in gammas]
    errors = [1-np.mean(cross_val_score(svm, x_train, y_train, cv=KFold(10),␣
 ↪scoring='accuracy')) for svm in svm_lst]
    idx = np.argmin(errors)
    cv_error.append(errors[idx])
    opt_gamma.append(gammas[idx])
```

[8]:
```
plt.plot(Cs, cv_error)
plt.xscale('log')
plt.title('Error rate of radial SVC with different C and gamma')
plt.xlabel("C")
plt.ylabel('CV error rate')
plt.show()
opt_idx = np.argmin(cv_error)
print("The cv error rate reaches its minimum of {} when gamma={} and c={}"
      .format(cv_error[opt_idx], opt_gamma[opt_idx], Cs[opt_idx]))
```

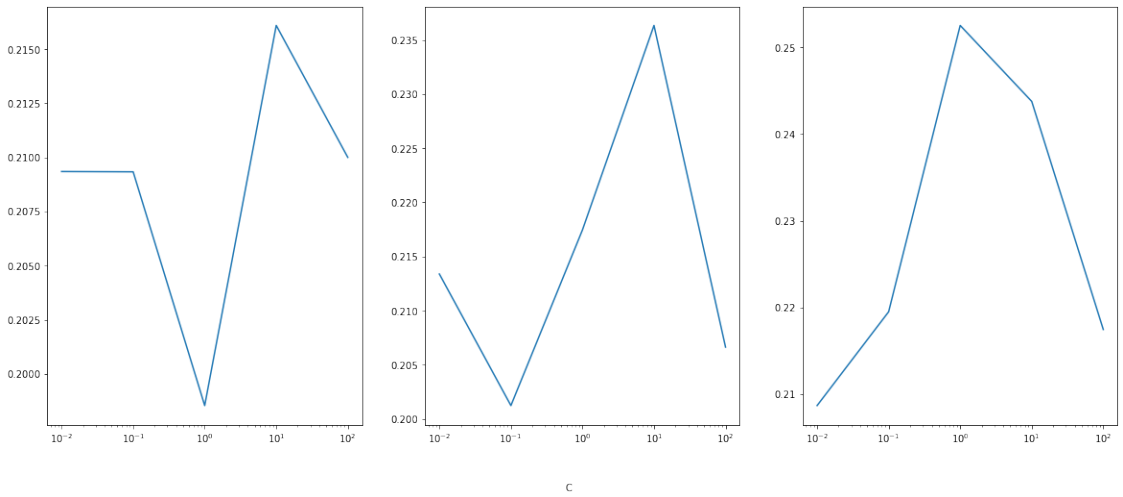



```
The cv error rate reaches its minimum of 0.21001269726101945 when
gamma=0.00017782794100389227 and c=100.0
```

[9]:
```
# polynomial SVC with different gammas, degrees, and Cs
degrees = np.arange(2, 5, 1)  #different degrees
cv_error = []
opt_gamma = []
for degree in degrees:
    for c in Cs:
        svm_lst = [svm.SVC(kernel='poly', C=c, degree = degree, gamma = gamma)␣
↪for gamma in gammas]
        errors = [1-np.mean(cross_val_score(svm, x_train, y_train,␣
↪cv=KFold(10), scoring='accuracy')) for svm in svm_lst]
```

```
        idx = np.argmin(errors)
        cv_error.append(errors[idx])
        opt_gamma.append(gammas[idx])
```

[10]:
```
fig, axes = plt.subplots(1,3)
fig.set_size_inches((20, 8))
for i in range(3):
    x_ax = Cs
    y_ax = cv_error[i*5:i*5+5]
    ax = axes[i]
    ax.plot(x_ax, y_ax)
    ax.set_xscale('log')
fig.text(0.5, 0.01, 'C', va='center')
plt.show()
opt_idx = np.argmin(cv_error)
print("The cv error rate reaches its minimum of {}, when gamma={}, degree={}␣
 ↪and cost={}"
      .format(cv_error[opt_idx], opt_gamma[opt_idx], degrees[opt_idx//5],␣
 ↪Cs[opt_idx%5]))
```




```
The cv error rate reaches its minimum of 0.19853528024668976, when
gamma=0.0031622776601683794, degree=2 and cost=1.0
```

For the radial kernel, I tuned SVC with different Cs and gammas. The training cv error constantly decreases as the cost value gets larger. And when C hits its maximum 100, the error yields its minimum of 0.21, with the parameter gamma being 0.000178. For the polynomial kernel, I tried with different polynomial degrees, gammas and cost values. The cv error in all models decline first, then pick up, and decrease again when the cost value gets very large. Of all the models I have created, the model with best performance is the polynomial kernel SVC where c=1.0, degree=2, and gamma=0.003, which has the smallest error rate of 0.199

[ ]:

[ ]: