Homework 3

LATE

Ross B. Alexander (rbalexan@stanford.edu)

Problem 1 (C6E3)

(a) iv. Steadily decrease.

This is because at s=0 we are essentially forcing the beta values to be zero (over-regularizing). This will lead to a very high training RSS since the model will not fit the data. As s is increased, the training RSS will steadily decrease as the beta values are allowed to approach their values achieved in least-squares linear regression.

(b) ii. Decrease initially, and then eventually start increasing in a U shape.

This is because at s=0 we are essentially forcing the beta values to be zero (over-regularizing). This will lead to a very high test RSS since the model (betas) will not fit the training data and therefore won't be a good predictor for the test data. As s is increased, the test RSS will decrease until the model begins to overfit on the training data. At this point, the test RSS will increase, leading to an overall U shape.

(c) iii. Steadily increase.

This is because at s=0 we are essentially forcing the beta values to be zero (over-regularizing). At s=0, there is only one possible model, so there is no variance across datasets. As s is increased, there is additional flexibility in the model, so the model will begin to fit the training data, leading to increased model variance. Even as s approaches large values and the model overfits to the training data, the variance still increases since the model is highly dependent on the dataset.

(d) iv. Steadily decrease.

This is because at s=0 we are essentially forcing the beta values to be zero (over-regularizing). Since the model (betas) will not fit the training data, the model will have a large (squared) bias. As s is increased, the model (betas) will begin to fit the training data better and therefore the (squared) bias of the model will decrease.

(e) v. Remain constant.

Irreducible error is an intrinsic property of the variance of the dataset and it does not change based on the kind of model that is fit to the dataset.

Problem 2 (C6E4)

(a) iii. Steadily increase.

When lambda=0, we are not performing any regularization and we achieve the betas from least-squares linear regression, which necessarily lead to the lowest possible training RSS. As lambda increases, we are doing additional regularization and increasingly forcing the betas toward zero, which means that the model will not fit the data and thus, the training RSS will increase.

(b) ii. Decrease initially, and then eventually start increasing in a U shape.

When lambda=0, we are not performing any regularization and we achieve the betas from least-squares linear regression, which are likely overfit to the training dataset. As such, we will have non-minimum test RSS. As lambda increases, we regularize our betas to prevent overfitting and eventually achieve the best model for the test dataset, leading to minimum test RSS. As lambda increases beyond this value, we are over-regularizing and increasingly forcing the betas toward zero, which means that the model will not fit the data and this, our test RSS will be quite large. Overall, this behavior implies a U shape.

(c) iv. Steadily decrease.

When lambda=0, we are not performing any regularization and we achieve the model (betas) from least-squares linear regression, which is strongly dependent on the training dataset. As such, we will have relatively large model variance. As lambda increases, we are increasingly forcing the betas toward zero and limiting the flexibility of our model, which means the model variance will decrease. Eventually, when lambda is sufficiently large (infinity), the betas will be all 0 for any dataset, leading to zero variance.

(d) iii. Steadily increase.

When lambda=0, we are not performing any regularization and we achieve the model (betas) from least-squares linear regression, which is fit (or overfit) on the training dataset. As such, we will have relatively small model (squared) bias since the model accurately captures the underlying trend. As lambda increases, we are increasingly forcing the betas toward zero and limiting the flexibility of our model, which means the model will not fit the underlying trend and thus, has increased (squared) bias. Eventually, when lambda is sufficiently large (infinity), the betas will be all 0 for any dataset, which is likely a poor fit for our dataset, leading to maximal (squared) bias.

(e) v. Remain constant.

Irreducible error is an intrinsic property of the variance of the dataset and it does not change based on the kind of model that is fit to the dataset.

Problem 3 (C6E9)

(a) - (d) $\boxed{\text{See attached code.}}$

Problem 4 (C7E1)

(a) Find a cubic polynomial

$$f_1(x) = a_1 + b_1 x + c_1 x^2 + d_1 x^3$$

such that

$$f(x) = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \beta_4 (x-\xi)_+^3 = f_1(x)$$

for all $x \le \xi$.
Express a_1, b_1, c_1, d_1 in terms of $\beta_0, \beta_1, \beta_2, \beta_3, \beta_4$.

- if $x \le \xi$, then

$$f(x) = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \beta_4 \overset{\to 0}{\cancel{(x-\xi)_+^3}}$$

$$f(x) = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3$$

- comparing this against $f_1(x)$, we have direct correspondence of the coefficients

$$a_1 = \beta_0$$
$$b_1 = \beta_1$$
$$c_1 = \beta_2$$
$$d_1 = \beta_3$$

(b) Repeat (a) for $x > \xi$ and $f_2(x) = a_2 + b_2 x + c_2 x^2 + d_2 x^3$.

- if $x > \xi$, we have

$$\begin{aligned} f(x) &= \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \beta_4 (x-\xi)_+^3 \\ &= \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \beta_4 (x-\xi)(x^2 - 2\xi x + \xi^2) \\ &= \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \beta_4 (x^3 - 2\xi x^2 + \xi^2 x - \xi x^2 + 2\xi^2 x - \xi^3) \\ &= \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \beta_4 (x^3 - 3\xi x^2 + 3\xi^2 x - \xi^3) \\ &= (\beta_0 - \beta_4 \xi^3) + (\beta_1 + 3\beta_4 \xi) x + (\beta_2 - 3\beta_4 \xi) x^2 + (\beta_3 + \beta_4) x^3 \end{aligned}$$

- comparing this against $f_2(x)$, we have the following correspondance of the coefficients

$$a_2 = \beta_0 - \beta_4 \xi^3$$
$$b_2 = \beta_1 + 3\beta_4 \xi^2$$
$$c_2 = \beta_2 - 3\beta_4 \xi$$
$$d_2 = \beta_3 + \beta_4$$

(c)

- at ξ, we have for f_1

$$f_1(\xi) = \beta_0 + \beta_1 \xi + \beta_2 \xi^2 + \beta_3 \xi^3$$

- at ξ, we have for f_2

$$\begin{aligned} f_2(\xi) &= (\beta_0 - \beta_4 \xi^3) + (\beta_1 + 3\beta_4 \xi^2)\xi + (\beta_2 - 3\beta_4 \xi)\xi^2 + (\beta_3 + \beta_4)\xi^3 \\ &= \beta_0 - \cancel{\beta_4 \xi^3} + \beta_1 \xi + \cancel{3\beta_4 \xi^3} + \beta_2 \xi^2 - \cancel{3\beta_4 \xi^3} + \beta_3 \xi^3 + \cancel{\beta_4 \xi^3} \\ &= \beta_0 + \beta_1 \xi + \beta_2 \xi^2 + \beta_3 \xi^3 \end{aligned}$$

- therefore,

$$f_1(\xi) = f_2(\xi)$$

(d)

- at ξ, we have f_1'

$$f_1'(x) = \beta_1 + 2\beta_2 x + 3\beta_3 x^2$$
$$f_1'(\xi) = \beta_1 + 2\beta_2 \xi + 3\beta_3 \xi^2$$

- at ξ, we have f_2'

$$f_2'(x) = (\beta_1 + 3\beta_4 \xi^2) + 2(\beta_2 - 3\beta_4 \xi)x + 3(\beta_3 + \beta_4)x^2$$

$$\begin{aligned} f_2'(\xi) &= (\beta_1 + 3\beta_4 \xi^2) + 2(\beta_2 - 3\beta_4 \xi)\xi + 3(\beta_3 + \beta_4)\xi^2 \\ &= \beta_1 + \cancel{3\beta_4 \xi^2} + 2\beta_2 \xi - \cancel{6\beta_4 \xi^2} + 3\beta_3 \xi^2 + \cancel{3\beta_4 \xi^2} \\ &= \beta_1 + 2\beta_2 \xi + 3\beta_3 \xi^2 \end{aligned}$$

- therefore,

$$f_1'(\xi) = f_2'(\xi)$$

(e)

- at ξ, we have f_1''

$$f_1''(x) = 2\beta_2 + 6\beta_3 x$$
$$f_1''(\xi) = 2\beta_2 + 6\beta_3 \xi$$

- at ξ, we have f_2''

$$f_2''(x) = 2(\beta_2 - 3\beta_4 \xi) + 6(\beta_3 + \beta_4)x$$

$$\begin{aligned} f_2''(\xi) &= 2(\beta_2 - 3\beta_4 \xi) + 6(\beta_3 + \beta_4)\xi \\ &= 2\beta_2 - \cancel{6\beta_4 \xi} + 6\beta_3 \xi + \cancel{6\beta_4 \xi} \\ &= 2\beta_2 + 6\beta_3 \xi \end{aligned}$$

- therefore,

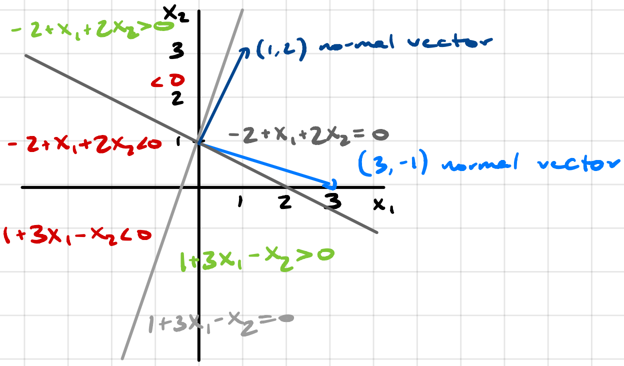
$$f_1''(\xi) = f_2''(\xi)$$

Problem 5 (C7E8)

(a) $\boxed{\text{See attached code.}}$

Problem 6 (C9E1)

(a) Sketch $1+3x_1-x_2=0$.



(b) Sketch $-2+x_1+2x_2=0$ (on the same plot).

- see above plot

Problem 7 (C9E8)

(a) - (h) $\boxed{\text{See attached code.}}$

Problem 3 (Chapter 6, Exercise 9, excluding parts (e), (f), and (g))

In [1]:
```
import pandas as pd

college = pd.read_csv("data/College.csv")

college = college.drop(columns=['Unnamed: 0'])
college['Private'] = college['Private'].astype('category')
college['Private'] = college['Private'].cat.codes
```

Problem 3(a)

In [2]:
```
from sklearn.model_selection import train_test_split

X = college.drop(columns=['Apps'])
y = college['Apps']

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

Problem 3(b)

In [3]:
```
import statsmodels.api as sm
import numpy as np

X_train = sm.add_constant(X_train)
X_test  = sm.add_constant(X_test)

linear = sm.regression.linear_model.OLS(y_train, X_train).fit()

y_pred = linear.predict(X_test)
test_mse = np.mean((y_pred - y_test)**2)

print(test_mse)
```

```
1261087.1761283777
```

Problem 3(c)

In [4]:

```python
from sklearn import linear_model
from sklearn.model_selection import cross_validate

lambdas           = np.logspace(-6, 6, 13)
avg_test_mse_hist = []

for lam in lambdas:

    ridge = linear_model.Ridge(alpha=lam)

    cv_results = cross_validate(ridge, X, y, cv=10, scoring='neg_mean_squared_

    avg_test_mse = np.mean(-1*cv_results['test_score'])
    avg_test_mse_hist.append(avg_test_mse)

    # print(lam, avg_test_mse)

min_idx = np.argmin(avg_test_mse_hist)
print("best avg test mse:    ", avg_test_mse_hist[min_idx])
print("corresponding lambda:", lambdas[min_idx])
```

```
best avg test mse:    1287421.609196762
corresponding lambda: 10.0
```

Problem 3(d)

In [5]:

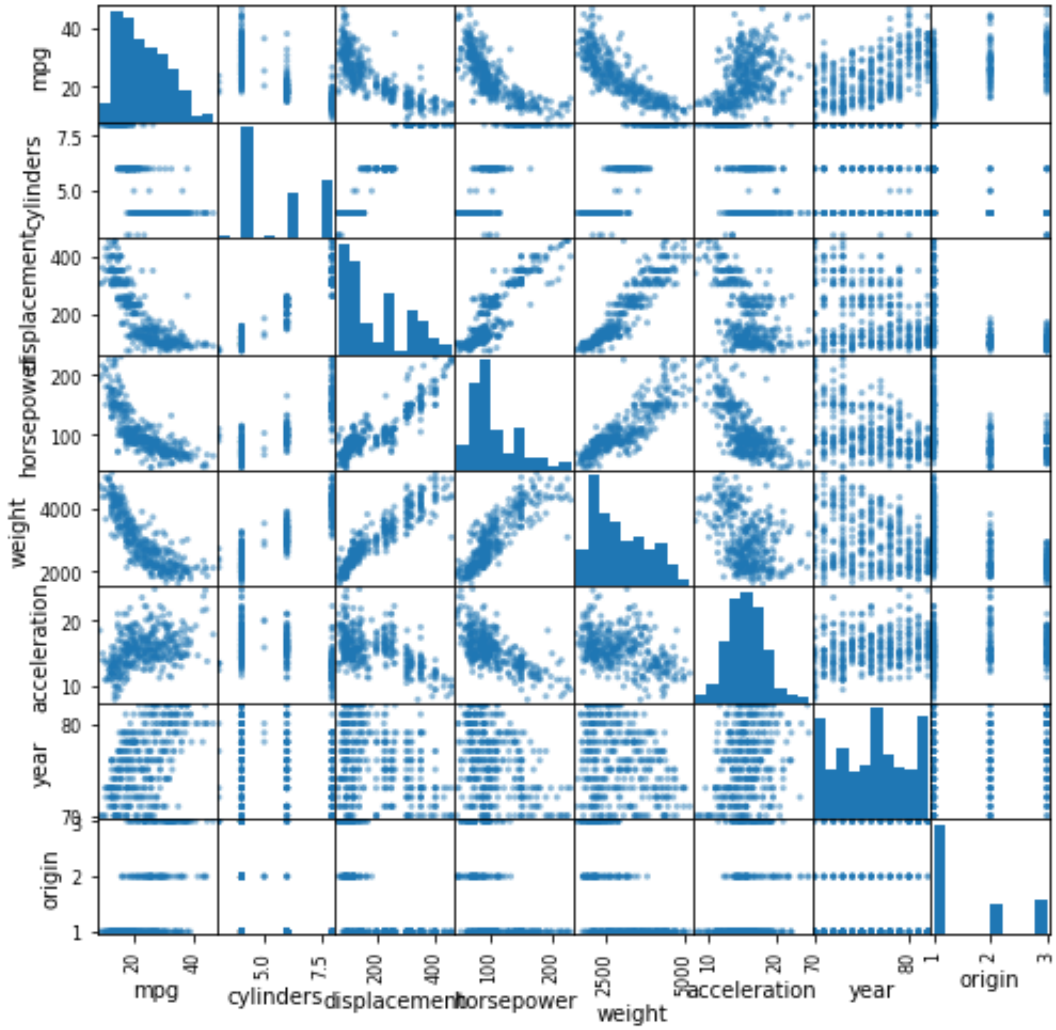
```python
lambdas           = np.logspace(-6, 6, 13)
avg_test_mse_hist = []

for lam in lambdas:

    lasso = linear_model.Lasso(alpha=lam)

    cv_results = cross_validate(lasso, X, y, cv=10, scoring='neg_mean_squared_

    avg_test_mse = np.mean(-1*cv_results['test_score'])
    avg_test_mse_hist.append(avg_test_mse)

    # print(lam, avg_test_mse)

min_idx = np.argmin(avg_test_mse_hist)
print("best avg test mse:    ", avg_test_mse_hist[min_idx])
print("corresponding lambda:", lambdas[min_idx])

lasso_opt = linear_model.Lasso(alpha=lambdas[min_idx]).fit(X_train, y_train)
print("num. nonzero coeffs.:", len(lasso_opt.coef_.nonzero()[0]))
```

```
best avg test mse:    1288186.8283163894
corresponding lambda: 1.0
num. nonzero coeffs.: 17
```

Problem 5 (Chapter 7, Exercise 8)

In [6]:

```
auto = pd.read_csv("data/Auto.csv")

auto = auto.drop(columns='name')

pd.plotting.scatter_matrix(auto, figsize=(8,8));
```



Looks like there are some nonlinear relationships for mpg and displacement predictors. We'll try to fit mpg against displacement. We'll fit using polynomial transformations of the data (which will include linear regression).

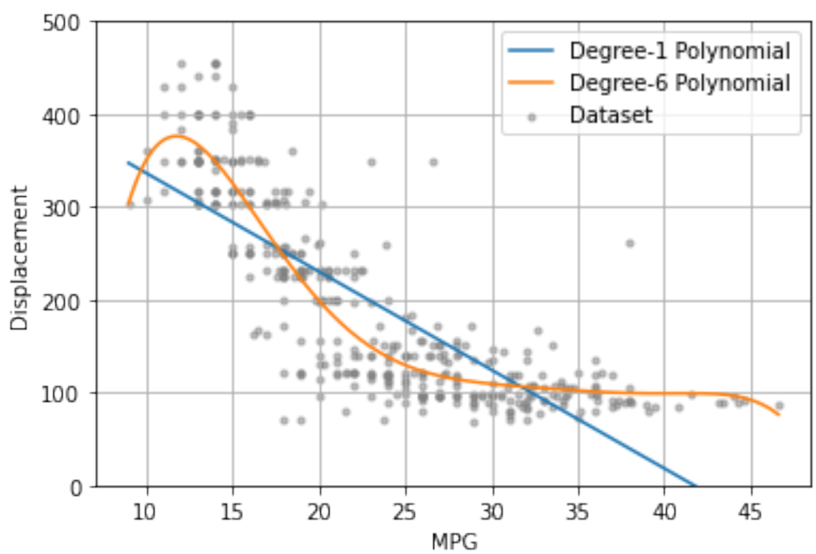
In [7]:

```
from sklearn.preprocessing import PolynomialFeatures

X = np.array(auto["mpg"]).reshape(-1, 1)
y = auto["displacement"]

ks                = np.arange(1, 10)
avg_test_mse_hist = []

for k in ks:

    poly   = linear_model.LinearRegression()

    X_poly = PolynomialFeatures(degree=k, include_bias=False).fit_transform(X

    cv_results = cross_validate(poly, X_poly, y, cv=10, scoring='neg_mean_squ

    avg_test_mse = np.mean(-1*cv_results['test_score'])
    avg_test_mse_hist.append(avg_test_mse)

    print(k, avg_test_mse)

min_idx = np.argmin(avg_test_mse_hist)
print("best avg test mse:    ", avg_test_mse_hist[min_idx])
print("corresponding degree:", ks[min_idx])
```

```
1 4235.632489029972
2 2581.1908702267283
3 2657.3874258934275
4 2562.680813891059
5 2449.084689600645
6 2429.726587020797
7 2458.309737199786
8 2650.559119261322
9 2754.904014448882
best avg test mse:    2429.726587020797
corresponding degree: 6
```

We see that among degree 1-10 polynomial expansions of the mpg predictor, that the linear regression (degree 1) has the highest (worst) CV error and that the degree 6 polynomial has the lowest (best) CV error.

In [8]:
```
import matplotlib.pyplot as plt

X_train, X_test, y_train, y_test = train_test_split(X, y)
X_plot = np.linspace(X.min(), X.max(), 200).reshape(-1, 1)

plt.figure()
plt.scatter(X, y, c="gray", alpha=0.5, s=10, label="Dataset")

for k in [1, 6]:

    X_poly_train = PolynomialFeatures(degree=k, include_bias=False).fit_trans
    poly         = linear_model.LinearRegression().fit(X_poly_train, y_train)

    X_plot_poly  = PolynomialFeatures(degree=k, include_bias=False).fit_trans
    y_plot_poly  = poly.predict(X_plot_poly)

    plt.plot(X_plot, y_plot_poly, label="Degree-%d Polynomial" %k)

plt.ylim(0, 500)
plt.xlabel("MPG")
plt.ylabel("Displacement")
plt.legend()
plt.grid()
```



Problem 7 (Chapter 9, Exercise 8)

In [9]:
```
oj = pd.read_csv("data/OJ.csv")

oj["Purchase"] = oj["Purchase"].astype('category').cat.codes
oj["Store7"]    = oj["Store7"].  astype('category').cat.codes
```

Problem 7(a)

In [10]:
```python
X = oj.drop(columns="Purchase")
y = oj["Purchase"]

X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=800/X.sh

print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

```
(800, 17) (270, 17) (800,) (270,)
```

Problem 7(b)

In [11]:
```python
from sklearn.svm import SVC

svc_linear = SVC(C=0.01, kernel="linear").fit(X_train, y_train)

print("fit status:                 ", "successful" if svc_linear.fit_status_ ==
print("number of support vectors:", svc_linear.n_support_)
print("fit primal intercept:     ", svc_linear.intercept_)
print("fit primal coefficients:  ", svc_linear.coef_)
```

```
fit status:                 successful
number of support vectors: [314 312]
fit primal intercept:      [3.2709307]
fit primal coefficients:   [[-0.00774799 -0.16455324  0.02911706 -0.06028802 -
0.02391942  0.15750947
  -0.04        0.21571864 -1.04794193 -0.21779748  0.05303648 -0.27083396
  -0.01872651  0.07113893 -0.01269375 -0.08940508 -0.03346765]]
```

Problem 7(c)

In [12]:
```python
train_error_rate = np.mean(np.abs(y_train - svc_linear.predict(X_train)))
test_error_rate  = np.mean(np.abs(y_test  - svc_linear.predict(X_test )))

print("train error rate:", train_error_rate)
print("test error rate: ", test_error_rate )
```

```
train error rate: 0.20875
test error rate:  0.21851851851851853
```

Problem 7(d)

In [13]:
```python
from warnings import filterwarnings
filterwarnings('ignore')

cs                        = np.logspace(-2, 1, 10)
avg_test_error_rate_hist = []

for c in cs:

    svc_linear = SVC(C=c, kernel="linear", max_iter=1E6)

    cv_results = cross_validate(svc_linear, X, y, cv=10, scoring='accuracy')

    avg_test_error_rate = np.mean(1 - cv_results['test_score'])
    avg_test_error_rate_hist.append(avg_test_error_rate)

    print("%.3f" % c, avg_test_error_rate)

min_idx = np.argmin(avg_test_error_rate_hist)
print("best avg test error rate:", avg_test_error_rate_hist[min_idx])
print("corresponding cost:      ", cs[min_idx])
```

```
0.010 0.22523364485981306
0.022 0.19813084112149534
0.046 0.18037383177570096
0.100 0.18411214953271032
0.215 0.17570093457943928
0.464 0.17102803738317757
1.000 0.1738317757009346
2.154 0.17570093457943928
4.642 0.2018691588785047
10.000 0.26728971962616827
best avg test error rate: 0.17102803738317757
corresponding cost:       0.46415888336127775
```

Problem 7(e)

In [14]:
```python
svc_linear = SVC(C=cs[min_idx], kernel="linear").fit(X_train, y_train)

train_error_rate = np.mean(np.abs(y_train - svc_linear.predict(X_train)))
test_error_rate  = np.mean(np.abs(y_test  - svc_linear.predict(X_test )))

print("train error rate:", train_error_rate)
print("test error rate: ", test_error_rate )
```

```
train error rate: 0.16375
test error rate:  0.16666666666666666
```

Problem 7(f)

In [15]:
```python
svc_rbf = SVC(C=0.01, kernel="rbf").fit(X_train, y_train)

print("fit status:               ", "successful" if svc_rbf.fit_status_ == 0 
print("number of support vectors:", svc_rbf.n_support_)
print("fit primal intercept:     ", svc_rbf.intercept_)
```

```
fit status:                  successful
number of support vectors: [316 316]
fit primal intercept:      [-0.99956289]
```

In [16]:
```python
train_error_rate = np.mean(np.abs(y_train - svc_rbf.predict(X_train)))
test_error_rate  = np.mean(np.abs(y_test  - svc_rbf.predict(X_test )))

print("train error rate:", train_error_rate)
print("test error rate: ", test_error_rate )
```

```
train error rate: 0.395
test error rate:  0.37407407407407406
```

In [17]:
```python
cs                       = np.logspace(2, 6, 13)
avg_test_error_rate_hist = []

for c in cs:

    svc_rbf    = SVC(C=c, kernel="rbf", max_iter=1E6)

    cv_results = cross_validate(svc_rbf, X, y, cv=10, scoring='accuracy')

    avg_test_error_rate = np.mean(1 - cv_results['test_score'])
    avg_test_error_rate_hist.append(avg_test_error_rate)

    print("%.3f" % c, avg_test_error_rate)

min_idx = np.argmin(avg_test_error_rate_hist)
print("best avg test error rate:", avg_test_error_rate_hist[min_idx])
print("corresponding cost:      ", cs[min_idx])
```

```
100.000 0.3766355140186916
215.443 0.2738317757009346
464.159 0.20560747663551404
1000.000 0.18504672897196267
2154.435 0.18224299065420563
4641.589 0.17476635514018693
10000.000 0.17289719626168226
21544.347 0.17196261682242991
46415.888 0.1766355140186916
100000.000 0.17009345794392522
215443.469 0.1738317757009346
464158.883 0.1738317757009346
1000000.000 0.17289719626168226
best avg test error rate: 0.17009345794392522
corresponding cost:       100000.0
```

In [18]:
```python
svc_rbf = SVC(C=cs[min_idx], kernel="rbf").fit(X_train, y_train)

train_error_rate = np.mean(np.abs(y_train - svc_rbf.predict(X_train)))
test_error_rate  = np.mean(np.abs(y_test  - svc_rbf.predict(X_test )))

print("train error rate:", train_error_rate)
print("test error rate: ", test_error_rate )
```

```
train error rate: 0.15875
test error rate:  0.1814814814814815
```

Problem 7(g)

In [19]:
```
svc_poly = SVC(C=0.01, kernel="poly", degree=2).fit(X_train, y_train)

print("fit status:                ", "successful" if svc_poly.fit_status_ == 0
print("number of support vectors:", svc_poly.n_support_)
print("fit primal intercept:      ", svc_poly.intercept_)
```

```
fit status:                successful
number of support vectors: [316 316]
fit primal intercept:      [-0.99748713]
```

In [20]:
```
train_error_rate = np.mean(np.abs(y_train - svc_poly.predict(X_train)))
test_error_rate  = np.mean(np.abs(y_test  - svc_poly.predict(X_test )))

print("train error rate:", train_error_rate)
print("test error rate: ", test_error_rate )
```

```
train error rate: 0.395
test error rate:  0.37407407407407406
```

In [21]:
```
cs                      = np.logspace(2, 6, 13)
avg_test_error_rate_hist = []

for c in cs:

    svc_poly   = SVC(C=c, kernel="poly", degree=2, max_iter=1E6)

    cv_results = cross_validate(svc_poly, X, y, cv=10, scoring='accuracy')

    avg_test_error_rate = np.mean(1 - cv_results['test_score'])
    avg_test_error_rate_hist.append(avg_test_error_rate)

    print("%.3f" % c, avg_test_error_rate)

min_idx = np.argmin(avg_test_error_rate_hist)
print("best avg test error rate:", avg_test_error_rate_hist[min_idx])
print("corresponding cost:      ", cs[min_idx])
```

```
100.000 0.38130841121495335
215.443 0.3074766355140187
464.159 0.20934579439252338
1000.000 0.1869158878504673
2154.435 0.18224299065420563
4641.589 0.1766355140186916
10000.000 0.17476635514018693
21544.347 0.17196261682242991
46415.888 0.17102803738317757
100000.000 0.17383177570093458
215443.469 0.22336448598130837
464158.883 0.202803738317757
1000000.000 0.25046728971962623
best avg test error rate: 0.17102803738317757
corresponding cost:       46415.888336127726
```

In [22]:
```
svc_poly = SVC(C=cs[min_idx], kernel="poly", degree=2).fit(X_train, y_train)

train_error_rate = np.mean(np.abs(y_train - svc_poly.predict(X_train)))
test_error_rate  = np.mean(np.abs(y_test  - svc_poly.predict(X_test )))

print("train error rate:", train_error_rate)
print("test error rate: ", test_error_rate )
```

```
train error rate: 0.16375
test error rate:  0.15925925925925927
```

Problem 7(h)

The radial basis function SVC seems to give the best results on this dataset as it achieves the lowest cross-validation test error rate of 17.01% misclassification.