Assignment 4

1. **Support Vector Machines with Synthetic Data**, 50 points.

For this problem, we will generate synthetic data for a nonlinear binary classification problem and partition it into training, validation and test sets. Our goal is to understand the behavior of SVMs with Radial-Basis Function (RBF) kernels with different values of C and γ.

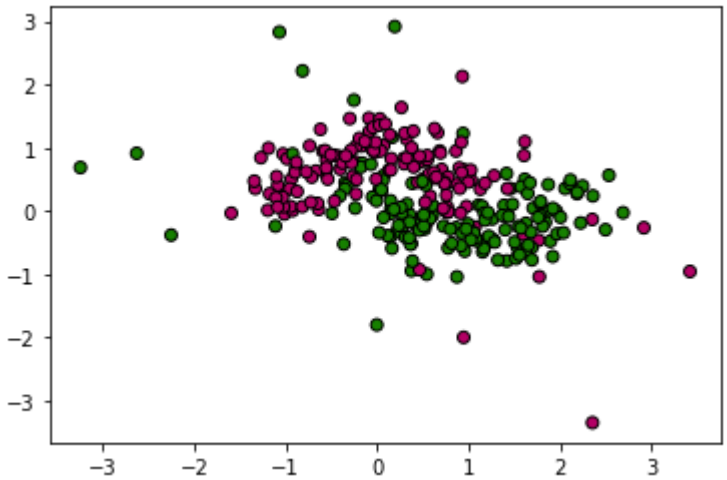
In [3]:
```
# DO NOT EDIT THIS FUNCTION; IF YOU WANT TO PLAY AROUND WITH DATA GENERATION,
# MAKE A COPY OF THIS FUNCTION AND THEN EDIT
#
import numpy as np
from sklearn.datasets import make_moons
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
# from sklearn.metrics import accuracy_score
from sklearn.neighbors import KNeighborsClassifier
from matplotlib.colors import ListedColormap

def generate_data(n_samples, tst_frac=0.2, val_frac=0.2):
    # Generate a non-linear data set
    X, y = make_moons(n_samples=n_samples, noise=0.25, random_state=42)

    # Take a small subset of the data and make it VERY noisy; that is, generate outlier
    m = 30
    np.random.seed(30)  # Deliberately use a different seed
    ind = np.random.permutation(n_samples)[:m]
    X[ind, :] += np.random.multivariate_normal([0, 0], np.eye(2), (m, ))
    y[ind] = 1 - y[ind]

    # Plot this data
    cmap = ListedColormap(['#b30065', '#178000'])
    plt.scatter(X[:, 0], X[:, 1], c=y, cmap=cmap, edgecolors='k')

    # First, we use train_test_split to partition (X, y) into training and test sets
    X_trn, X_tst, y_trn, y_tst = train_test_split(X, y, test_size=tst_frac,
                                                  random_state=42)

    # Next, we use train_test_split to further partition (X_trn, y_trn) into training a
    X_trn, X_val, y_trn, y_val = train_test_split(X_trn, y_trn, test_size=val_frac,
                                                  random_state=42)

    return (X_trn, y_trn), (X_val, y_val), (X_tst, y_tst)
```

In [4]:
```
#
#  DO NOT EDIT THIS FUNCTION; IF YOU WANT TO PLAY AROUND WITH VISUALIZATION,
#  MAKE A COPY OF THIS FUNCTION AND THEN EDIT
#

def visualize(models, param, X, y):
```

```
    # Initialize plotting
    if len(models) % 3 == 0:
        nrows = len(models) // 3
    else:
        nrows = len(models) // 3 + 1

    fig, axes = plt.subplots(nrows=nrows, ncols=3, figsize=(15, 5.0 * nrows))
    cmap = ListedColormap(['#b30065', '#178000'])

    # Create a mesh
    xMin, xMax = X[:, 0].min() - 1, X[:, 0].max() + 1
    yMin, yMax = X[:, 1].min() - 1, X[:, 1].max() + 1
    xMesh, yMesh = np.meshgrid(np.arange(xMin, xMax, 0.01),
                               np.arange(yMin, yMax, 0.01))

    for i, (p, clf) in enumerate(models.items()):
        # if i > 0:
        #   break
        r, c = np.divmod(i, 3)
        ax = axes[r, c]

        # Plot contours
        zMesh = clf.decision_function(np.c_[xMesh.ravel(), yMesh.ravel()])
        zMesh = zMesh.reshape(xMesh.shape)
        ax.contourf(xMesh, yMesh, zMesh, cmap=plt.cm.PiYG, alpha=0.6)

        if (param == 'C' and p > 0.0) or (param == 'gamma'):
            ax.contour(xMesh, yMesh, zMesh, colors='k', levels=[-1, 0, 1],
                       alpha=0.5, linestyles=['--', '-', '--'])

        # Plot data
        ax.scatter(X[:, 0], X[:, 1], c=y, cmap=cmap, edgecolors='k')
        ax.set_title('{0} = {1}'.format(param, p))
```

In [5]:

```
# Generate the data
n_samples = 300    # Total size of data set
(X_trn, y_trn), (X_val, y_val), (X_tst, y_tst) = generate_data(n_samples)
```



---

a. (25 points) The effect of the regularization parameter, C

Complete the Python code snippet below that takes the generated synthetic 2-d data as input and learns non-linear SVMs. Use scikit-learn's SVC function to learn SVM models with **radial-basis kernels** for fixed γ and various choices of $C \in \{10^{-3}, 10^{-2} \cdots, 1, \cdots 10^{5}\}$. The value of γ is fixed to $\gamma = \frac{1}{d \cdot \sigma_X}$, where d is the data dimension and σ_X is the standard deviation of the data set X. SVC can automatically use these setting for γ if you pass the argument gamma = 'scale' (see documentation for more details).

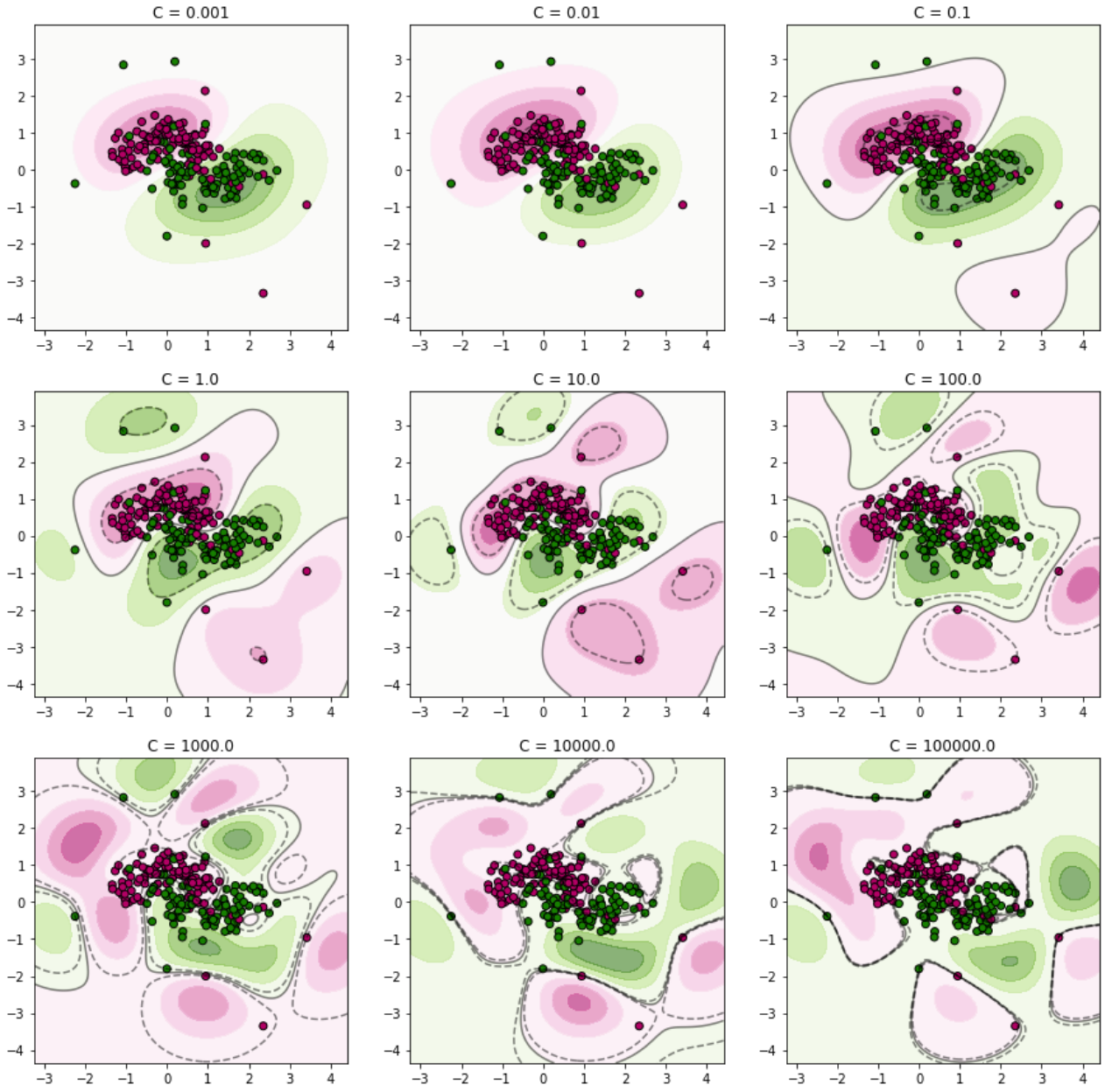
Plot: For each classifier, compute **both** the **training error** and the **validation error**. Plot them together, making sure to label the axes and each curve clearly.

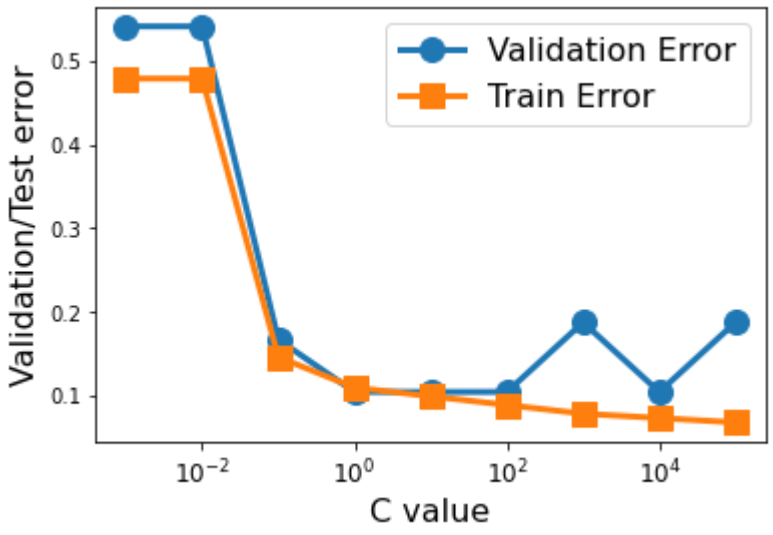
Discussion: How do the training error and the validation error change with C? Based on the visualization of the models and their resulting classifiers, how does changing C change the models? Explain in terms of minimizing the SVM's objective function $\frac{1}{2}\mathbf{w}'\mathbf{w} + C\,\Sigma_{i=1}^{n}\,\ell(\mathbf{w} \mid \mathbf{x}_i, y_i)$, where ℓ is the hinge loss for each training example $(\mathbf{x}_i, y_i)$.

Final Model Selection: Use the validation set to select the best the classifier corresponding to the best value, C_{best}. Report the accuracy on the **test set** for this selected best SVM model. *Note: You should report a single number, your final test set accuracy on the model corresponding to $C{best}$_.*

In [10]:
```
# Learn support vector classifiers with a radial-basis function kernel with
# fixed gamma = 1 / (n_features * X.std()) and different values of C
from sklearn.svm import SVC

C_range = np.arange(-3.0, 6.0, 1.0)
C_values = np.power(10.0, C_range)

models = dict()
trnErr = dict()
valErr = dict()
tstErr = dict()
for C in C_values:
    clf = SVC(C=C, kernel='rbf', gamma='scale')
    clf.fit(X_trn, y_trn)
    models[C] = clf
    trnErr[C] = 1-clf.score(X_trn, y_trn)
    valErr[C] = 1-clf.score(X_val, y_val)
    tstErr[C] = 1-clf.score(X_tst, y_tst)
    #
    #
    # Insert your code here to learn SVM models
    #
    #
visualize(models, 'C', X_trn, y_trn)
plt.figure()
plt.plot(valErr.keys(), valErr.values(), marker='o', linewidth=3, markersize=12)
plt.plot(trnErr.keys(), trnErr.values(), marker='s', linewidth=3, markersize =12)
plt.xlabel('C value', fontsize=16)
plt.ylabel('Validation/Test error', fontsize=16)
plt.xticks(list(valErr.keys()), fontsize=12)
plt.legend(['Validation Error', 'Train Error'], fontsize=16)
plt.xscale('log')
#
#
# Insert your code here to perform model selection
```

```
#
#
```


C = 0.001
C = 0.01
C = 0.1
C = 1.0
C = 10.0
C = 100.0
C = 1000.0
C = 10000.0
C = 100000.0


Validation Error
Train Error
Validation/Test error
C value

In [11]:
```
minErr = 1
bestC = []
for i in valErr:
    if valErr[i] < minErr:
        minErr = valErr[i]
        bestC = []
        bestC.append(i)
    elif minErr==valErr[i]:
        bestC.append(i)

print("List of Best C values on Validation:",bestC)

minErr = 1
finalC = -1
for i in range(len(bestC)):
    if tstErr[bestC[i]] < minErr:
        finalC = bestC[i]
        minErr = tstErr[bestC[i]]

print("Best C value is:", finalC, "the accuracy is:", (1-tstErr[finalC])*100,"%")
```

```
List of Best C values on Validation: [1.0, 10.0, 100.0, 10000.0]
Best C value is: 100.0 the accuracy is: 85.0 %
```

Discussion: With increasing C value the error decreases for train error. Whereas in validation the error decreases till C value 100. The C parameter is used to tell SVM that how much misclassification we want to avoid, so larger C values have smaller margin and smaller values have large margin. Above, plot is the example of how the larger C value on training set has lower training error.

---

b. (25 points) The effect of the RBF kernel parameter, γ

Complete the Python code snippet below that takes the generated synthetic 2-d data as input and learns various non-linear SVMs. Use scikit-learn's SVC function to learn SVM models with **radial-basis kernels** for fixed C and various choices of $\gamma \in \{10^{-2}, 10^{-1}\, 1, 10,\ 10^2\, 10^3\}$. The value of C is fixed to $C = 10$.

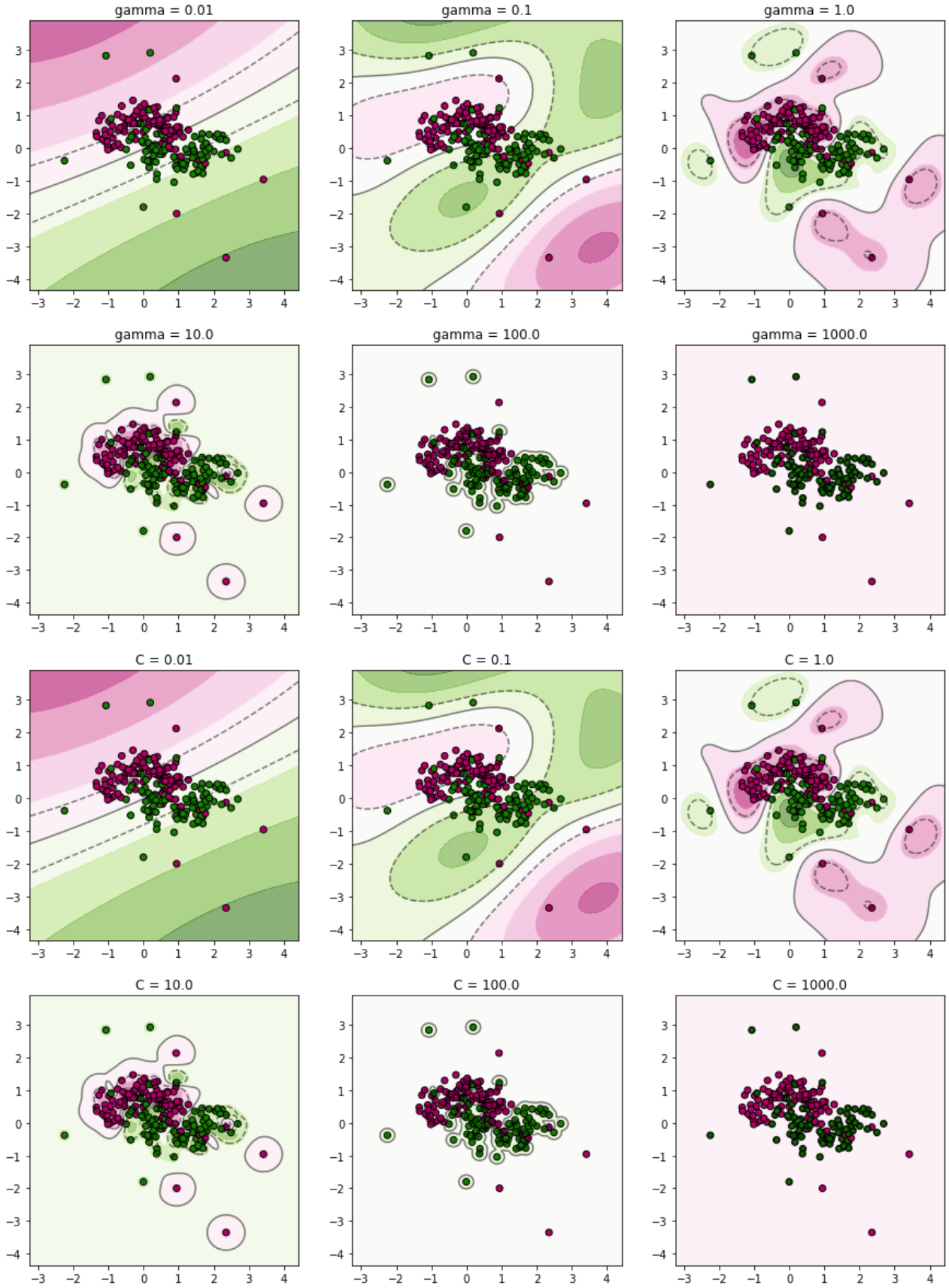
Plot: For each classifier, compute **both** the **training error** and the **validation error**. Plot them together, making sure to label the axes and each curve clearly.

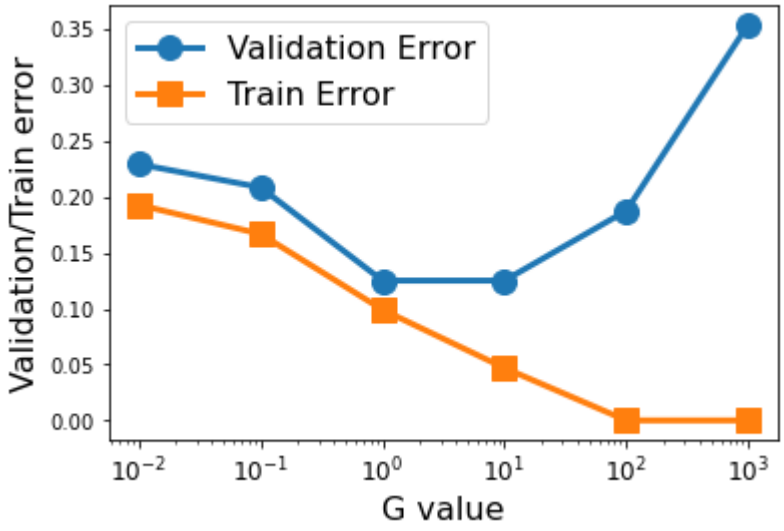
Discussion: How do the training error and the validation error change with γ? Based on the visualization of the models and their resulting classifiers, how does changing γ change the models? Explain in terms of the functional form of the RBF kernel, $\kappa(\mathbf{x},\, \mathbf{z}) \,=\, \exp(-\gamma \cdot \|\mathbf{x} - \mathbf{z}\|^2)$

Final Model Selection: Use the validation set to select the best the classifier corresponding to the best value, γ_{best}. Report the accuracy on the **test set** for this selected best SVM model. *Note: You should report a single number, your final test set accuracy on the model corresponding to γ_{best}.*

In [14]:
```
# Learn support vector classifiers with a radial-basis function kernel with
# fixed C = 10.0 and different values of gamma
gamma_range = np.arange(-2.0, 4.0, 1.0)
gamma_values = np.power(10.0, gamma_range)
```

```
models = dict()
trnErr = dict()
valErr = dict()
tstErr = dict()

for G in gamma_values:
    clf = SVC(C=10, kernel='rbf', gamma=G)
    clf.fit(X_trn, y_trn)
    models[G] = clf
    trnErr[G] = 1-clf.score(X_trn, y_trn)
    valErr[G] = 1-clf.score(X_val, y_val)
    tstErr[G] = 1-clf.score(X_tst, y_tst)

  #
  #
  # Insert your code here to learn SVM models
  #
  #

visualize(models, 'gamma', X_trn, y_trn)

#
#
# Insert your code here to perform model selection
#
#
visualize(models, 'C', X_trn, y_trn)
plt.figure()
plt.plot(valErr.keys(), valErr.values(), marker='o', linewidth=3, markersize=12)
plt.plot(trnErr.keys(), trnErr.values(), marker='s', linewidth=3, markersize =12)
plt.xlabel('G value', fontsize=16)
plt.ylabel('Validation/Train error', fontsize=16)
plt.xticks(list(valErr.keys()), fontsize=12)
plt.legend(['Validation Error', 'Train Error'], fontsize=16)
plt.xscale('log')
```


gamma = 0.01
gamma = 0.1
gamma = 1.0
gamma = 10.0
gamma = 100.0
gamma = 1000.0
C = 0.01
C = 0.1
C = 1.0
C = 10.0
C = 100.0
C = 1000.0



In [15]:

```
minErr = 1
bestG = []
for i in valErr:
    if valErr[i] < minErr:
        minErr = valErr[i]
        bestG = []
        bestG.append(i)
    elif minErr==valErr[i]:
        bestG.append(i)

print("List of Best G values on Validation:",bestG)

minErr = 1
finalG = -1
for i in range(len(bestG)):
    if tstErr[bestG[i]] < minErr:
        finalG = bestG[i]
        minErr = tstErr[bestG[i]]

print("Best G value is:", finalG, "the accuracy is:", (1-tstErr[finalG])*100,"%")
```

```
List of Best G values on Validation: [1.0, 10.0]
Best G value is: 1.0 the accuracy is: 83.33333333333334 %
```

Discussion: With increase in G values the error decreases for train error. While in validation the error decreases until 1 and after that error shoots up as the value of gamma becomes larger. Smaller gamma value means the model has broad decision region.

---

2. **Breast Cancer Diagnosis with Support Vector Machines**, 25 points.

For this problem, we will use the Wisconsin Breast Cancer) data set, which has already been pre-processed and partitioned into training, validation and test sets. Numpy's loadtxt command can be used to load CSV files.

In [16]:

```
# Load the Breast Cancer Diagnosis data set; download the files from eLearning
```

```
# CSV files can be read easily using np.loadtxt()
#
# Insert your code here.
#
wbc_trn=np.loadtxt(open("wdbc_trn.csv", "rb"), delimiter=",")
wbc_tst=np.loadtxt(open("wdbc_tst.csv", "rb"), delimiter=",")
wbc_val=np.loadtxt(open("wdbc_val.csv", "rb"), delimiter=",")
X_trn, y_trn = wbc_trn[:,1:], wbc_trn[:,0]
X_tst, y_tst = wbc_tst[:,1:], wbc_tst[:,0]
X_val, y_val = wbc_val[:,1:], wbc_val[:,0]
```

Use scikit-learn's SVC function to learn SVM models with **radial-basis kernels** for **each combination** of $C \in \{10^{-2}, 10^{-1}, 1, 10^{1}, \cdots 10^{4}\}$ and $\gamma \in \{10^{-3}, 10^{-2}\, 10^{-1}, 1,\ 10,\ 10^{2}\}$. Print the tables corresponding to the training and validation errors.

Final Model Selection: Use the validation set to select the best the classifier corresponding to the best parameter values, C_{best} and γ_{best}. Report the accuracy on the **test set** for this selected best SVM model. *Note: You should report a single number, your final test set accuracy on the model corresponding to $C{best}and\gamma{best}$.*

In [18]:

```
#
#
# Insert your code here to perform model selection
#
#

models = dict()
trnErr = dict()
valErr = dict()
tstErr = dict()
C_range = np.arange(-2.0, 4.0, 1.0)
C_values = np.power(10.0, C_range)

gamma_range = np.arange(-3.0, 3.0, 1.0)
gamma_values = np.power(10.0, gamma_range)

for C in C_values:
    for G in gamma_values:
        clf = SVC(C=C, kernel='rbf', gamma=G)
        clf.fit(X_trn, y_trn)
        models[C,G] = clf
        trnErr[C,G] = 1-clf.score(X_trn, y_trn)
        valErr[C,G] = 1-clf.score(X_val, y_val)
        tstErr[C,G] = 1-clf.score(X_tst, y_tst)

minErr = 1
bestCG = []
for i in valErr:
    if valErr[i] < minErr:
        minErr = valErr[i]
        bestCG = []
        bestCG.append(i)
    elif minErr==valErr[i]:
        bestCG.append(i)

print("List of Best C and G values:",bestCG)
```

```
minErr = 1
finalCG = -1
for i in range(len(bestCG)):
    if tstErr[bestCG[i]] < minErr:
        finalCG = bestCG[i]
        minErr = tstErr[bestCG[i]]

print("Best Value for C :",finalCG[0],"Best Value for G :", finalCG[1],"the accuracy is
```

```
List of Best C and G values: [(100.0, 0.01), (1000.0, 0.01)]
Best Value for C : 100.0 Best Value for G : 0.01 the accuracy is: 96.52173913043478 %
```

---

3. **Breast Cancer Diagnosis with k-Nearest Neighbors**, 25 points.

Use scikit-learn's k-nearest neighbor classifier to learn models for Breast Cancer Diagnosis with $k \in \{1, 5, 11, 15, 21\}$, with the kd-tree algorithm.

Plot: For each classifier, compute **both** the **training error** and the **validation error**. Plot them together, making sure to label the axes and each curve clearly.

Final Model Selection: Use the validation set to select the best the classifier corresponding to the best parameter value, k_{best}. Report the accuracy on the **test set** for this selected best kNN model. *Note: You should report a single number, your final test set accuracy on the model corresponding to $k*{best}$_.*

In [20]:

```
#
#
# Insert your code here to perform model selection
#
#
models = dict()
trnErr = dict()
valErr = dict()
tstErr = dict()

k_values = [1,5,11,15,21]

for K in k_values:
    kd = KNeighborsClassifier(n_neighbors=K,algorithm='kd_tree')
    kd.fit(X_trn,y_trn)
    models[K] = kd
    trnErr[K] = 1-kd.score(X_trn, y_trn)
    valErr[K] = 1-kd.score(X_val, y_val)
    tstErr[K] = 1-kd.score(X_tst, y_tst)

plt.figure()
plt.plot(valErr.keys(), valErr.values(), marker='o', linewidth=3, markersize=12)
plt.plot(trnErr.keys(), trnErr.values(), marker='s', linewidth=3, markersize =12)
plt.xlabel('K value', fontsize=16)
plt.ylabel('Validation/Test error', fontsize=16)
plt.xticks(list(valErr.keys()), fontsize=12)
plt.legend(['Validation Error', 'Test Error'], fontsize=16)
```

```
plt.xscale('log')


minErr = 1
bestK = []
for i in valErr:
    if valErr[i] < minErr:
        minErr = valErr[i]
        bestK = []
        bestK.append(i)
    elif minErr==valErr[i]:
        bestK.append(i)

print("List of Best K values:",bestK)


minErr = 1
finalK = -1
for i in range(len(bestK)):
    if tstErr[bestK[i]] < minErr:
        finalK = bestK[i]
        minErr = tstErr[bestK[i]]

print("Best K value is:", finalK , "the accuracy is ",(1-tstErr[finalK])*100, "%")
```

```
List of Best K values: [5, 11]
Best K value is: 11 the accuracy is  97.3913043478261 %
```

Discussion: Which of these two approaches, SVMs or kNN, would you prefer for this classification task? Explain.

Answer: kNN is a better approach as the accuracy of kNN is higher than SVM, as accuracy of kNN on test data is 97.39% and accuracy of SVM on test data is 96.52%. As the data given has 30 attri