SVMs & K-NN

Shayling Zhao

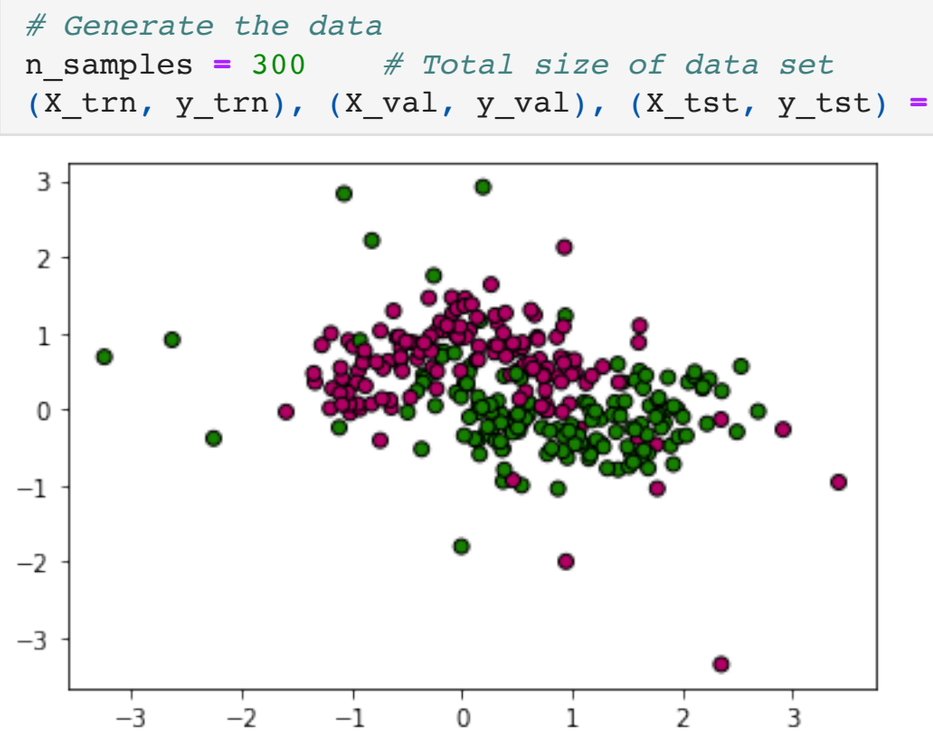
1. **Support Vector Machines with Synthetic Data**

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 [11]:
```
import numpy as np
from sklearn.datasets import make_moons
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
import os
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier

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 outliers
  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 and validation sets
  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 [12]:
```
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 [13]:
```
# 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)
```

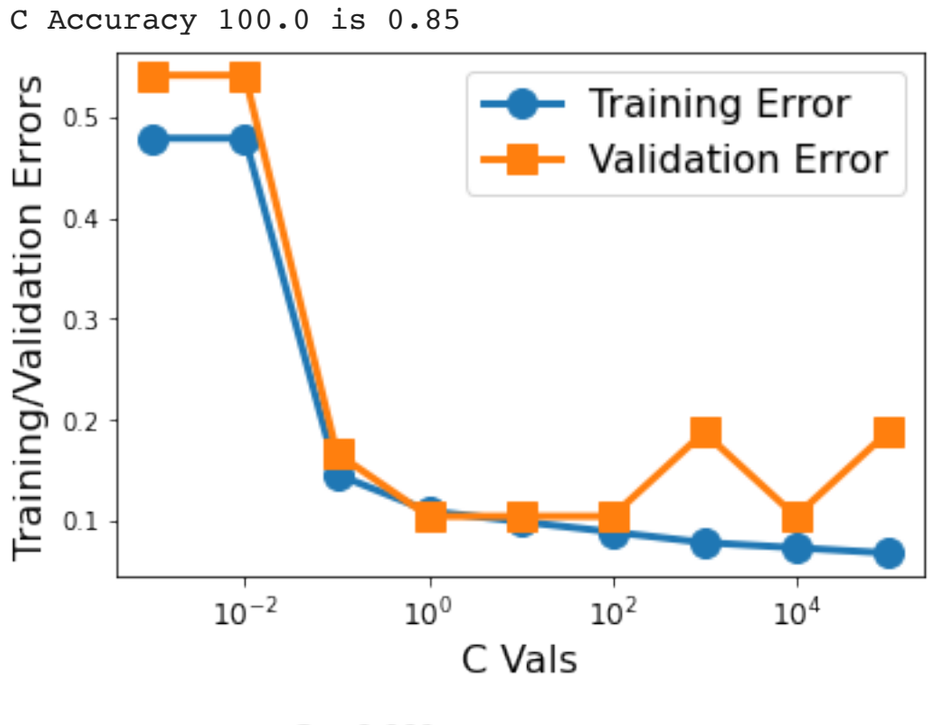


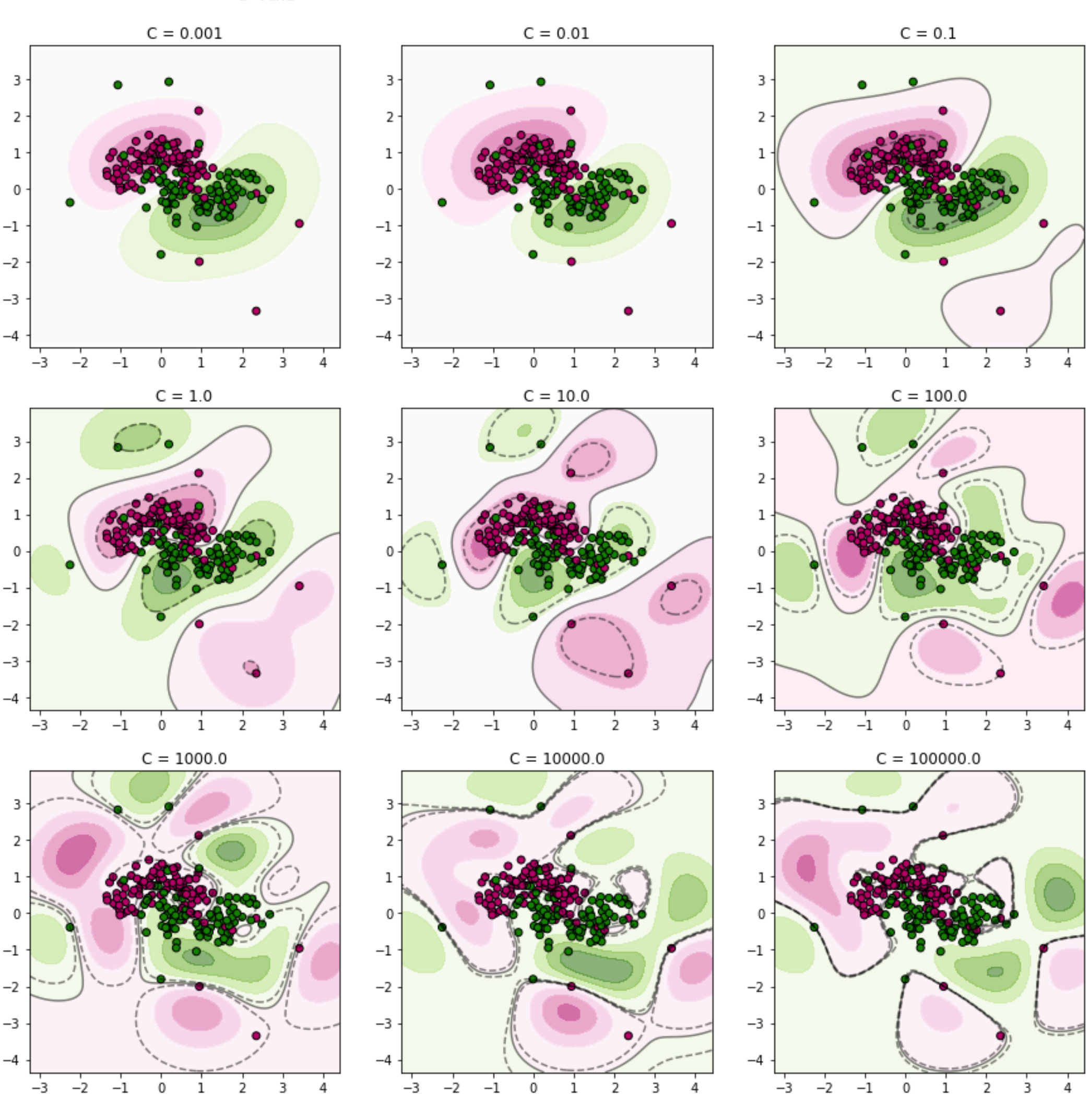
a. (25 points) The effect of the regularization parameter, C

Python code snippet below takes the generated synthetic 2-d data as input and learns non-linear SVMs. Used 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).

In [14]:
```
from sklearn.svm import SVC
# Learn support vector classifiers with a radial-basis function kernel with
# fixed gamma = 1 / (n_features * X.std()) and different values of C

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))

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

models = dict()
trnErr = dict()
valErr = dict()

for C in C_values:
    svcFile = SVC(C=C, kernel = 'rbf', gamma = 'scale')
    svcFile.fit(X_trn, y_trn)
    models[C] = svcFile
    trn_Error = 1-(svcFile.score(X_trn, y_trn))
    trnErr[C] = trn_Error
    val_Error = 1-(svcFile.score(X_val, y_val))
    valErr[C] = val_Error

plt.figure()
plt.plot(trnErr.keys(), trnErr.values(), marker='o', linewidth=3, markersize=12)
plt.plot(valErr.keys(), valErr.values(), marker='s', linewidth=3, markersize=12)
plt.xlabel('C Vals', fontsize=16)
plt.ylabel('Training/Validation Errors', fontsize=16)
plt.xticks(list(valErr.keys()), fontsize=12)
plt.legend(['Training Error', 'Validation Error'], fontsize=16)
plt.xscale('log')


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

minimum = 1;
for c in valErr:
    if models[c].score(X_tst, y_tst) > models[minimum].score(X_tst, y_tst):
        minimum = c

accuracy = models[minimum].score(X_tst, y_tst)
print("C Accuracy " + str(minimum) + " is " + str(accuracy))
```

```
C Accuracy 100.0 is 0.85
```





Discussion:

As C increases, training/validation error decreases. In the graphs shown, you can see that as C increases, the complexity of shapes around the data points also increases. This aligns with the idea that with lower C's, there is more error, and with higher C's, there is less errors (but more likely to overfit).

When looking at SVM's function, a larger C means that slack variables becomes more penalized and a smaller margin accepted. Therefore, minimization of the function occurs with higher C values, but need to be careful due to overfitting.

Final Model Selection:

In [15]:
```
minimum = 1;
for c in valErr:
    if valErr[c] < valErr[minimum]:
        minimum = c

error = 1 - (models[minimum].score(X_tst, y_tst))
print("Lowest error of C value " + str(minimum) + " is " + str(error))
```

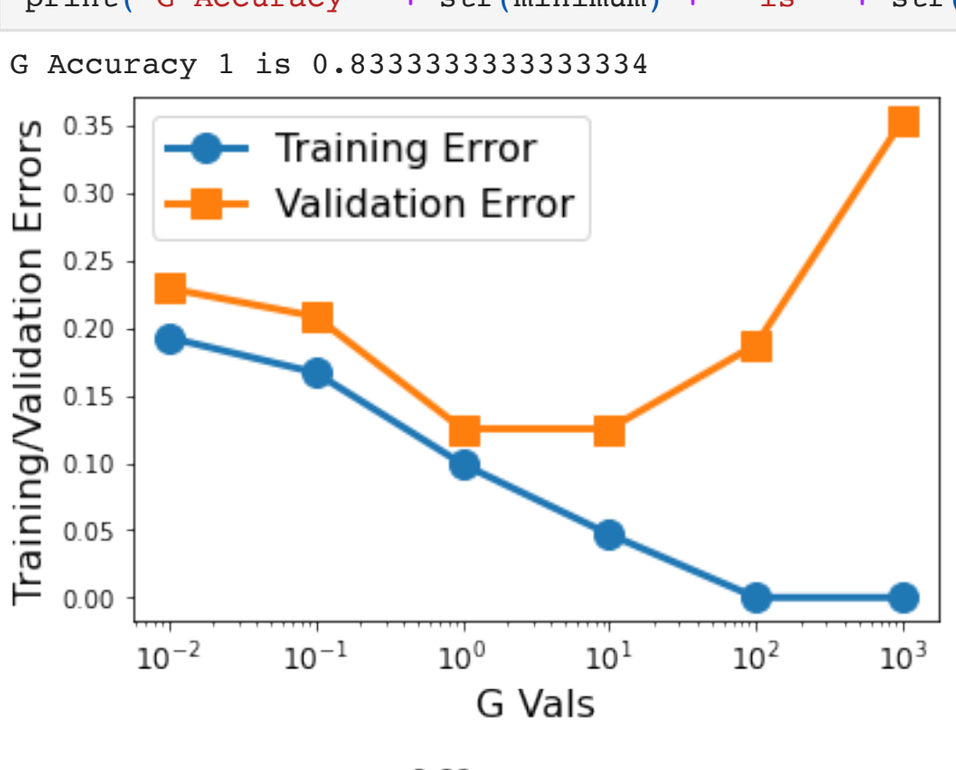
```
Lowest error of C value 1 is 0.16666666666666663
```

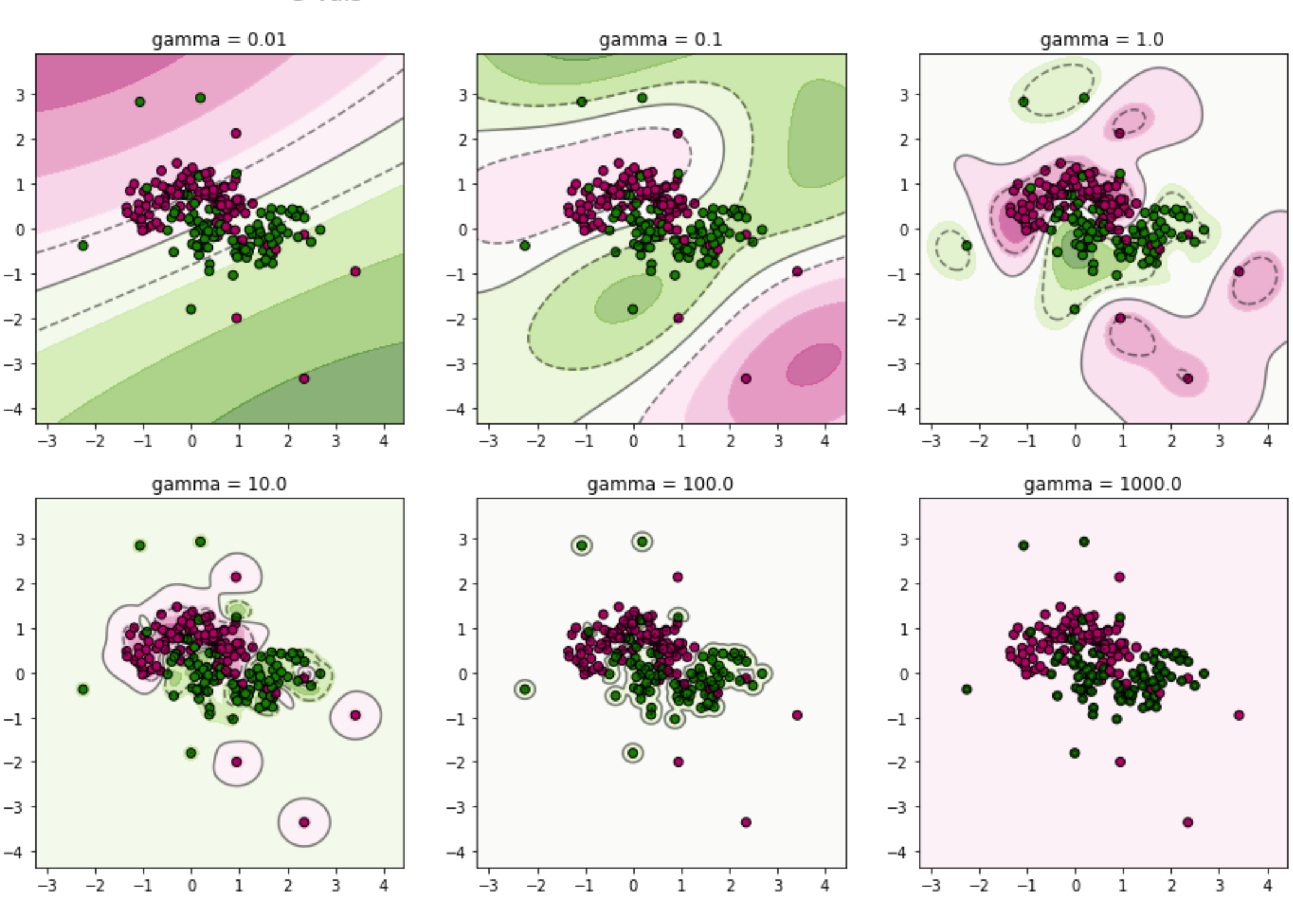
b. (25 points) The effect of the RBF kernel parameter, γ

Python code snippet below takes the generated synthetic 2-d data as input and learns various non-linear SVMs. Used 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 $C = 10$.

In [16]:
```
# Learn support vector classifiers with a radial-basis function kernel with
# fixed C = 10.0 and different values of gamma

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))

gamma_range = np.arange(-2.0, 4.0, 1.0)
gamma_values = np.power(10.0, gamma_range)

models = dict()
trnErr = dict()
valErr = dict()

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

plt.figure()
plt.plot(trnErr.keys(), trnErr.values(), marker='o', linewidth=3, markersize=12)
plt.plot(valErr.keys(), valErr.values(), marker='s', linewidth=3, markersize=12)
plt.xlabel('G Vals', fontsize=16)
plt.ylabel('Training/Validation Errors', fontsize=16)
plt.xticks(list(valErr.keys()), fontsize=12)
plt.legend(['Training Error', 'Validation Error'], fontsize=16)
plt.xscale('log')

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

minimum = 1;
for g in valErr:
    if models[g].score(X_tst, y_tst) > models[minimum].score(X_tst, y_tst):
        minimum = g

accuracy = models[minimum].score(X_tst, y_tst)
print("G Accuracy " + str(minimum) + " is " + str(accuracy))
```

```
G Accuracy 1 is 0.8333333333333334
```





Discussion:

As gamma increases, the training error decreases. As gamma decreases, the validation error decreases but increases as gamma values increase. In SVC, a consequence of higher gamma values is that the model tries to fit the training set.

As seen in the models, the area around points/the data set are more constricted/specific as values of gamma increases. In the training/validation error chart, the validation error starts off in a decrease, but changes course to increase after the gamma values becomes 10 or more. Meanwhile, the training error continues on a consistent decline as the gamma values increase.

In relation to the functional form of the RBF kernel, small gamma yields a broad decision region while large gamma yields a more constricted/specific decision region because the higher the gamma value, the equation will yeild a lower K(x,z).

Final Model Selection:

In [17]:
```
minimum = 1;
for g in valErr:
    if valErr[g] < valErr[minimum]:
        minimum = g

error = 1 - (models[minimum].score(X_tst, y_tst))
print("Lowest error of G value " + str(minimum) + " is " + str(error))
```

```
Lowest error of G value 1 is 0.16666666666666663
```

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

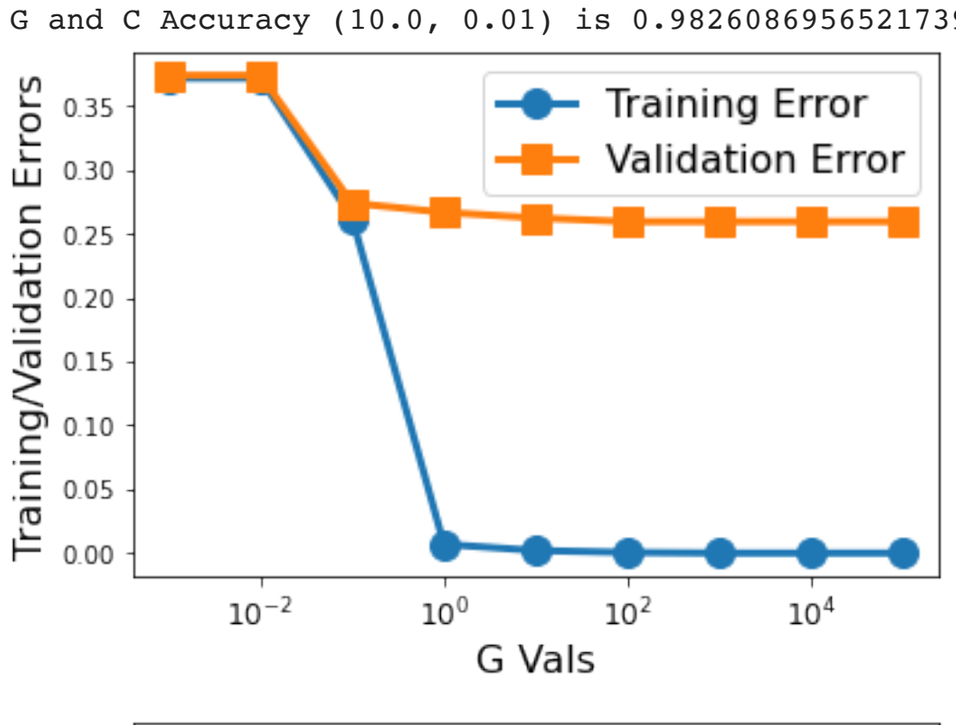
For this problem, we will use the Wisconsin Breast Cancer) data set, which has already been pre-processed and partitioned into training, validation and test sets. Numpy's loadtxt command can be used to load CSV files.

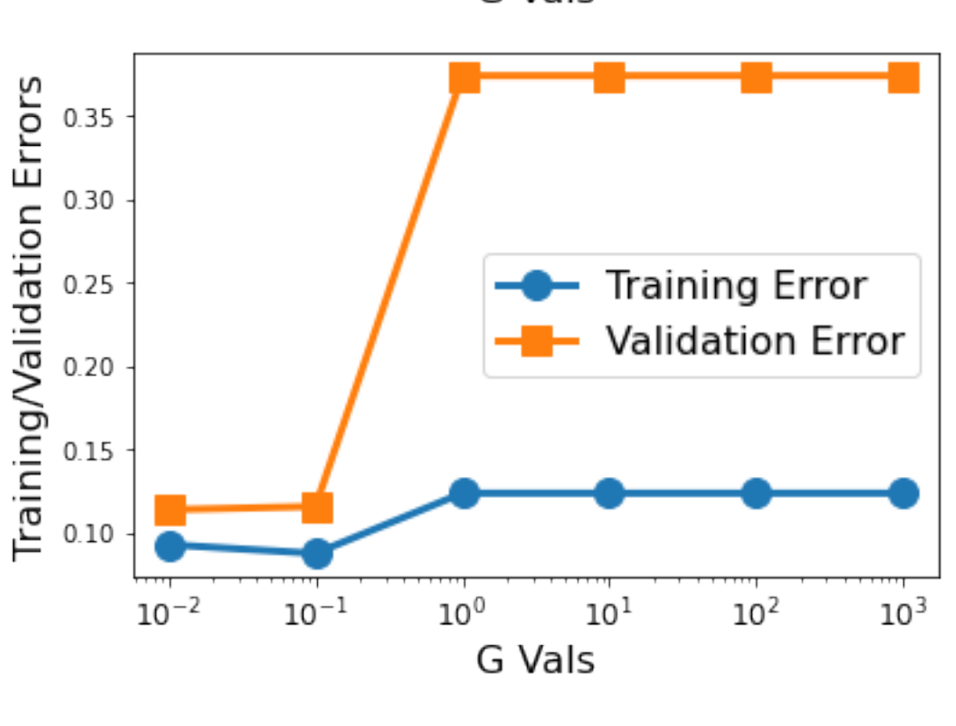
In [18]:
```
train = np.loadtxt(r'/Users/shaylingzhao/Desktop/wdbc_trn.csv', delimiter=",")
X_trn = train[:,1:]
y_trn = train[:,0]
test = np.loadtxt(r'/Users/shaylingzhao/Desktop/wdbc_tst.csv', delimiter=",")
X_tst = test[:,1:]
y_tst = test[:,0]
value = np.loadtxt(r'/Users/shaylingzhao/Desktop/wdbc_val.csv', delimiter=",")
X_val = value[:,1:]
y_val = value[:,0]
```

Used 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 $\gamma \in \{10^{-3}, 10^{-2} \, 10^{-1}, 1, \, 10, \, 10^2\}$.

In [19]:
```
C_range = np.arange(-3.0, 6.0, 1.0)
C_values = np.power(10.0, C_range)

gamma_range = np.arange(-2.0, 4.0, 1.0)
gamma_values = np.power(10.0, gamma_range)

models = dict()
trnErr = dict()
valErr = dict()

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

cTrnErr = {}
cValErr = {}
for C in C_values:
    train_sum = 0
    val_sum = 0
    for G in gamma_values:
        train_sum += trnErr[(C, G)]
        val_sum += valErr[(C, G)]
    cTrnErr[C] = train_sum/len(gamma_values)
    cValErr[C] = val_sum/len(gamma_values)


plt.figure()
plt.plot(cTrnErr.keys(), cTrnErr.values(), marker='o', linewidth=3, markersize=12)
plt.plot(cValErr.keys(), cValErr.values(), marker='s', linewidth=3, markersize=12)
plt.xlabel('G Vals', fontsize=16)
plt.ylabel('Training/Validation Errors', fontsize=16)
plt.xticks(list(cValErr.keys()), fontsize=12)
plt.legend(['Training Error', 'Validation Error'], fontsize=16)
plt.xscale('log')

gTrnErr = {}
gValErr = {}
for G in gamma_values:
    gtrain_sum = 0
    gval_sum = 0
    for C in C_values:
        gtrain_sum += trnErr[(C, G)]
        gval_sum += valErr[(C, G)]
    gTrnErr[G] = gtrain_sum/len(C_values)
    gValErr[G] = gval_sum/len(C_values)

plt.figure()
plt.plot(gTrnErr.keys(), gTrnErr.values(), marker='o', linewidth=3, markersize=12)
plt.plot(gValErr.keys(), gValErr.values(), marker='s', linewidth=3, markersize=12)
plt.xlabel('G Vals', fontsize=16)
plt.ylabel('Training/Validation Errors', fontsize=16)
plt.xticks(list(gValErr.keys()), fontsize=12)
plt.legend(['Training Error', 'Validation Error'], fontsize=16)
plt.xscale('log')

minimum = list(valErr.keys())[0]
for key in valErr.keys():
    if models[key].score(X_tst, y_tst) > models[minimum].score(X_tst, y_tst):
        minimum = key

accuracy = models[minimum].score(X_tst, y_tst)
print("G and C Accuracy " + str(minimum) + " is " + str(accuracy))
```

```
G and C Accuracy (10.0, 0.01) is 0.9826086956521739
```





Final Model Selection:

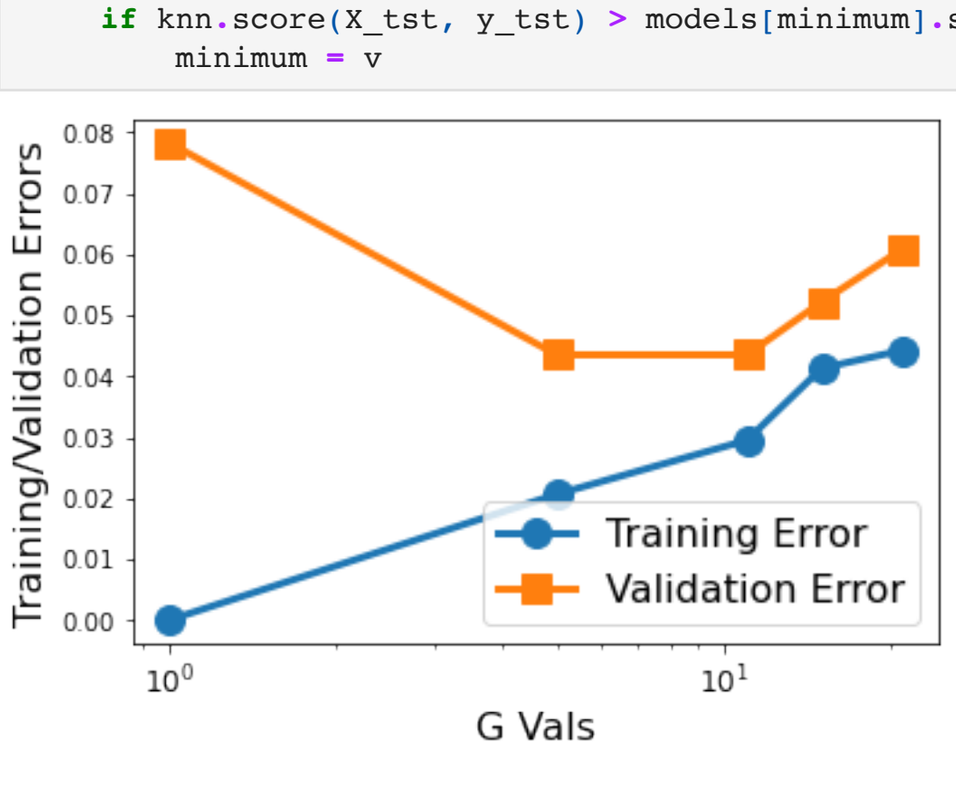
In [20]:
```
minimum = list(valErr.keys())[0]
for key in valErr.keys():
    if valErr[key] < valErr[minimum]:
        minimum = key

error = (models[minimum].score(X_tst, y_tst))
print("Lowest error of G value " + str(minimum) + " is " + str(1-error))
print("So, the highest test accuracy is " + str(error))
```

```
Lowest error of G value (100.0, 0.01) is 0.034782608695652195
So, the highest test accuracy is 0.9652173913043478
```

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

Used 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.

In [21]:
```
from sklearn.neighbors import KNeighborsClassifier

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

models = dict()
trnErr = dict()
valErr = dict()

for value in k:
    knn = KNeighborsClassifier(n_neighbors = value, algorithm = 'kd_tree')
    knn.fit(X_trn, y_trn)
    models[value] = knn
    trn_Error = 1-(knn.score(X_trn, y_trn))
    trnErr[value] = trn_Error
    val_Error = 1-(knn.score(X_val, y_val))
    valErr[value] = val_Error

plt.figure()
plt.plot(trnErr.keys(), trnErr.values(), marker='o', linewidth=3, markersize=12)
plt.plot(valErr.keys(), valErr.values(), marker='s', linewidth=3, markersize=12)
plt.xlabel('G Vals', fontsize=16)
plt.ylabel('Training/Validation Errors', fontsize=16)
plt.xticks(list(valErr.keys()), fontsize=12)
plt.legend(['Training Error', 'Validation Error'], fontsize=16)
plt.xscale('log')

minimum = 1;
for v in valErr:
    if knn.score(X_tst, y_tst) > models[minimum].score(X_tst, y_tst):
        minimum = v
```



Final Model Selection:

In [22]:
```
accuracy = models[minimum].score(X_tst, y_tst)
print("Best K value = " + str(minimum) + " and has an accuracy of " + str(accuracy))
```

```
Best K value = 11 and has an accuracy of 0.9739130434782609
```

Discussion:

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

the kNN classification task is preferred because it yielded a higher accuracy percentage of 97.39% compared to the SVM that yielded an accuracy percentage of 95.65%. kNN also works better with data that has many attributes while SVM needs to match C with gamma which could require additional work/time.