Assignment 4

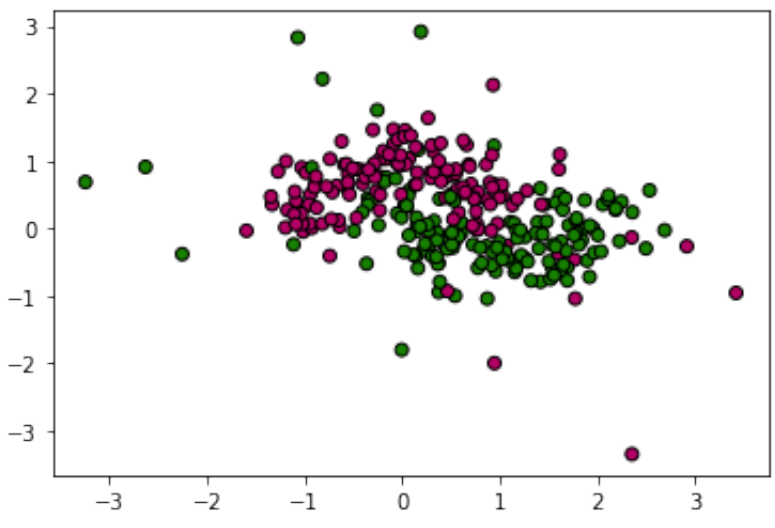
1. **Support Vector Machines with Synthetic Data**, 50 points.

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 [1]:
```
# 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 matplotlib.colors import ListedColormap
from sklearn.metrics import accuracy_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC

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 [2]:
```
#
#  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 [3]:
```
# 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).

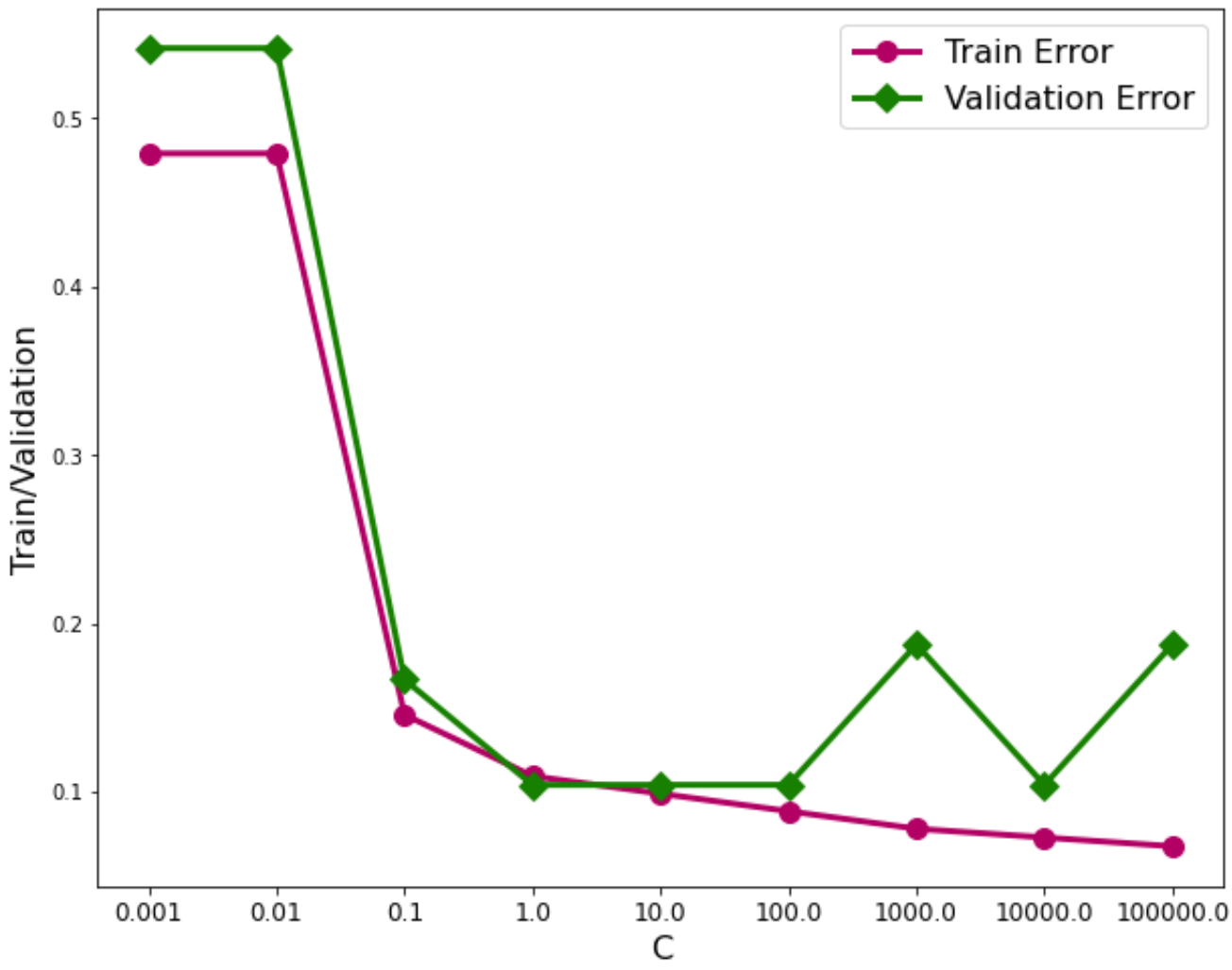
Plot: For each classifier, compute **both** the **training error** and the **validation error**. Plot them together, making sure to label the axes and each curve clearly.

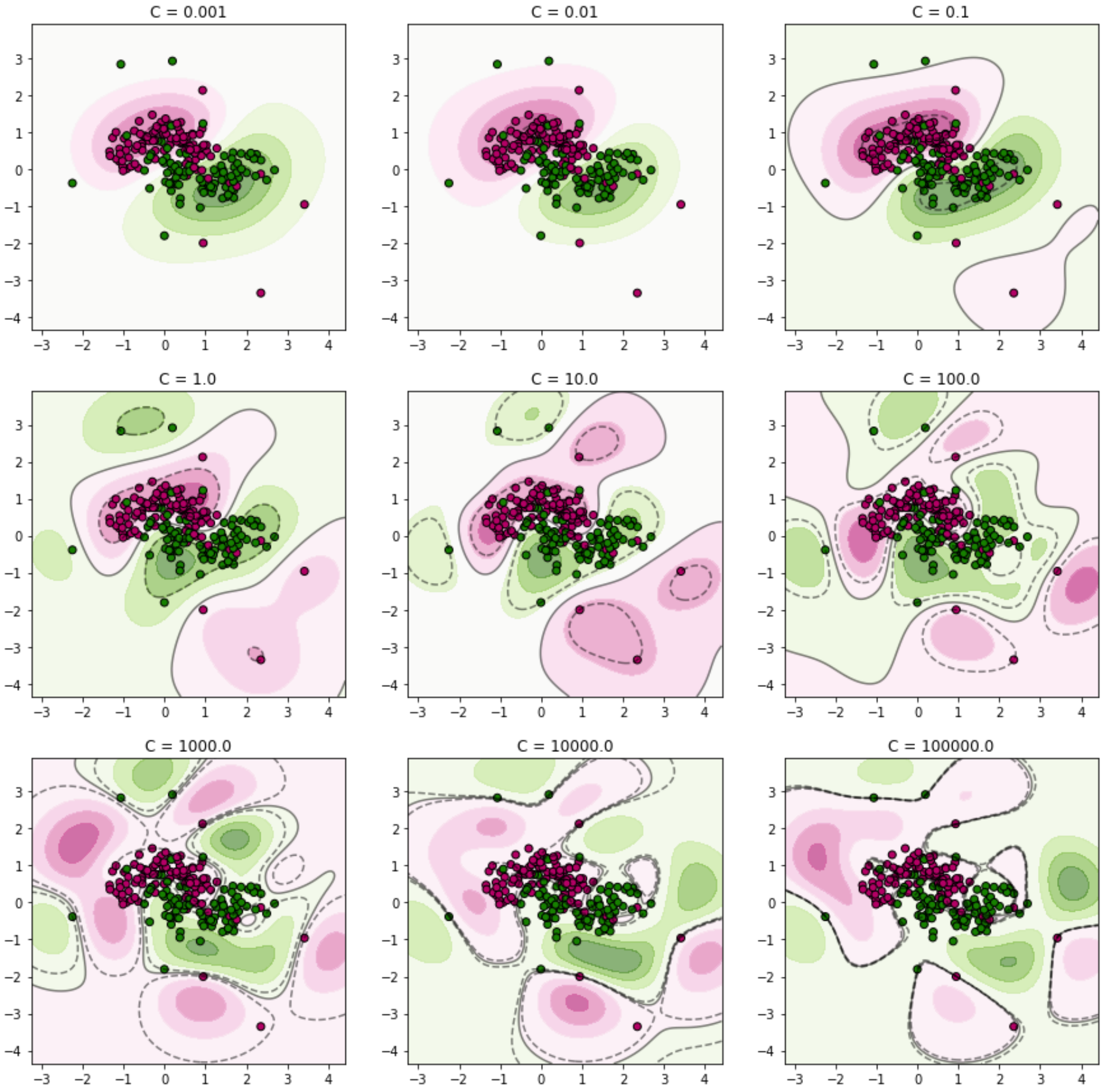
Discussion: How do the training error and the validation error change with C? Based on the visualization of the models and their resulting classifiers, how does changing C change the models? Explain in terms of minimizing the SVM's objective function $\frac{1}{2}\mathbf{w}'\mathbf{w} + C\,\Sigma_{i=1}^{n}\,\ell(\mathbf{w} \mid \mathbf{x}_i, y_i)$, 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 [4]:
```
# Learn support vector classifiers with a radial-basis function kernel with
# fixed gamma = 1 / (n_features * X.std()) and different values of C
C_range = np.arange(-3.0, 6.0, 1.0)
C_values = np.power(10.0, C_range)

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

idx = 0
for C in C_values:
    clf = SVC(C = C,gamma='scale',kernel='rbf',random_state=0)
    clf.fit(X_trn, y_trn)
    models[C]=clf
    y_trn_pred = clf.predict(X_trn)
    y_val_pred = clf.predict(X_val)
    trnErr[idx]= 1 - accuracy_score(y_trn,y_trn_pred)
    valErr[idx]= 1 - accuracy_score(y_val,y_val_pred)
    idx += 1

plt.figure(figsize=(10,8))
plt.plot(list(trnErr.keys()), list(trnErr.values()), marker='o', linewidth=3, markersize=10, color='#b30065')
plt.plot(list(valErr.keys()), list(valErr.values()), marker='D', linewidth=3, markersize=10, color='#178000')
plt.legend(['Train Error','Validation Error'], fontsize=16)
plt.xlabel('C', fontsize=16)
plt.ylabel('Train/Validation', fontsize=16)
plt.xticks(list(trnErr.keys()), ('0.001', '0.01', '0.1', '1.0', '10.0','100.0','1000.0','10000.0','100000.0'),font

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

minErr = 1
count_minErr = 1
bestVal = []
ls_cValues = list(C_values)

for v in valErr:
    if(valErr[v] < minErr):
        minErr = valErr[v]
        bestVal.clear()
        bestVal.append(ls_cValues[v])
    elif (valErr[v] == minErr):
        count_minErr +=1
        bestVal.append(ls_cValues[v])

print('Best C values: ',bestVal)
```

```
Best C values:  [1.0, 10.0, 100.0, 10000.0]
```


Train Error
Validation Error
Train/Validation
0.5
0.4
0.3
0.2
0.1
0.001
0.01
0.1
1.0
10.0
100.0
1000.0
10000.0
100000.0
C



Discussion:

The training error decreases with the increasing values of C. The validation error decreases till 1, stays that way till 100 and shows a oscillatory increase-decrease then after.

Here 'C' behaves as the regularization parameter,as we penalize our slack variables more and more with the increase in C value. C controls the cost of misclassification on the training data. The C parameter trades off correct classification of training examples against maximization of the decision function’s margin. For larger values of C, a smaller margin will be accepted if the decision function is better at classifying all training points correctly. A lower C will encourage a larger margin, therefore a simpler decision function(clearly visible in the visualized models), at the cost of training accuracy. Large C has higher chances of misclassification as it accounts for hard margin, making the algorithm to understand the input data better. This leads to overfitting, which explains the increase in validation error.

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 [5]:
```
clf = SVC(C = 100,gamma='scale',kernel='rbf')
clf.fit(X_trn, y_trn)
y_tst_pred = clf.predict(X_tst)

tstErr = 1-accuracy_score(y_tst,y_tst_pred)

print('Test Error was evaluated to be : '+ str(round(tstErr,5)))
print('Final test set accuracy:', accuracy_score(y_tst,y_tst_pred)*100)
```

```
Test Error was evaluated to be : 0.15
Final test set accuracy: 85.0
```

The best value of C that gives minimum test set error is 100 as the reported test error is ~0.15, with the accuracy of 85%.

---

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 $C = 10$.

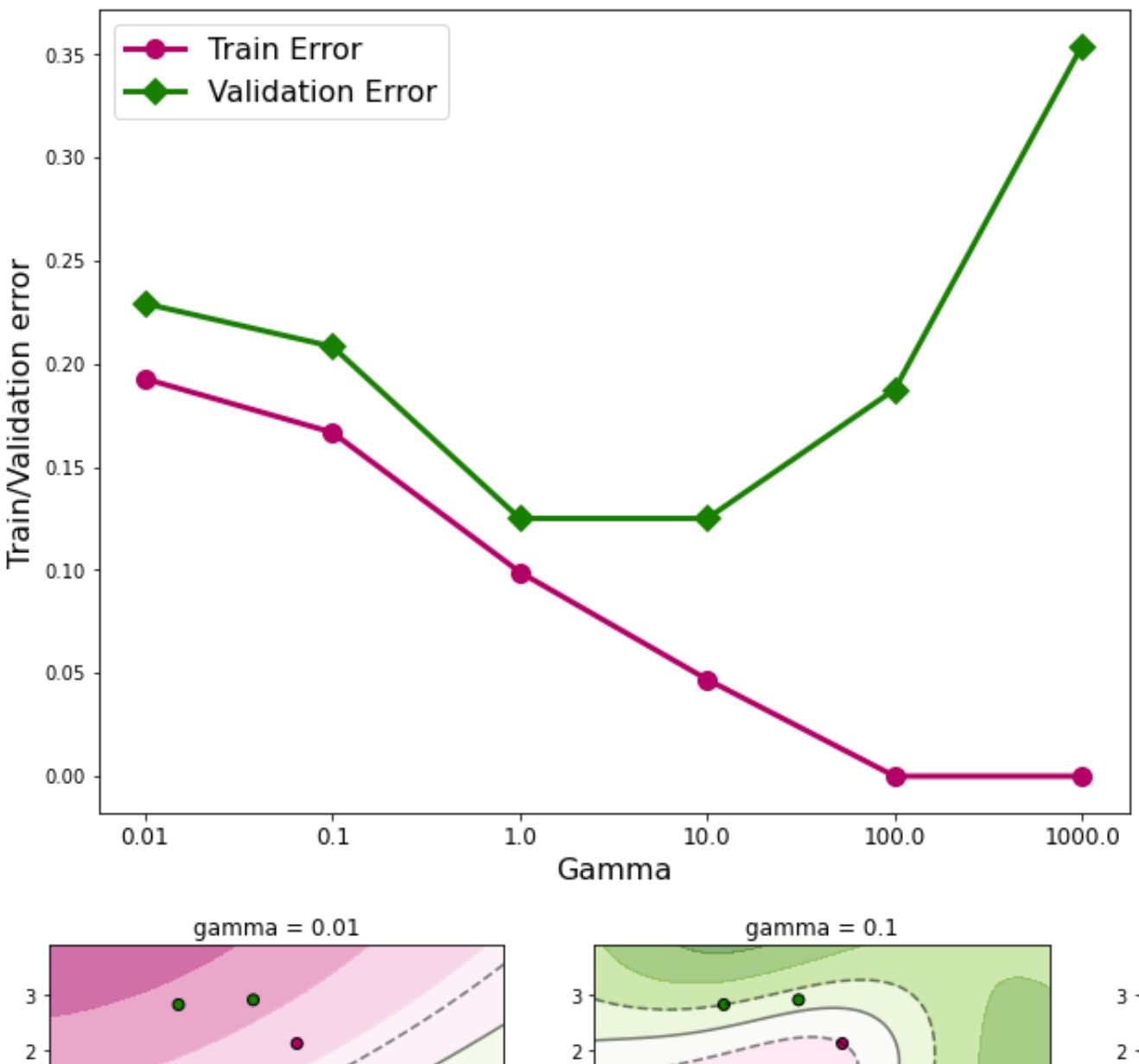
Plot: For each classifier, compute **both** the **training error** and the **validation error**. Plot them together, making sure to label the axes and each curve clearly.

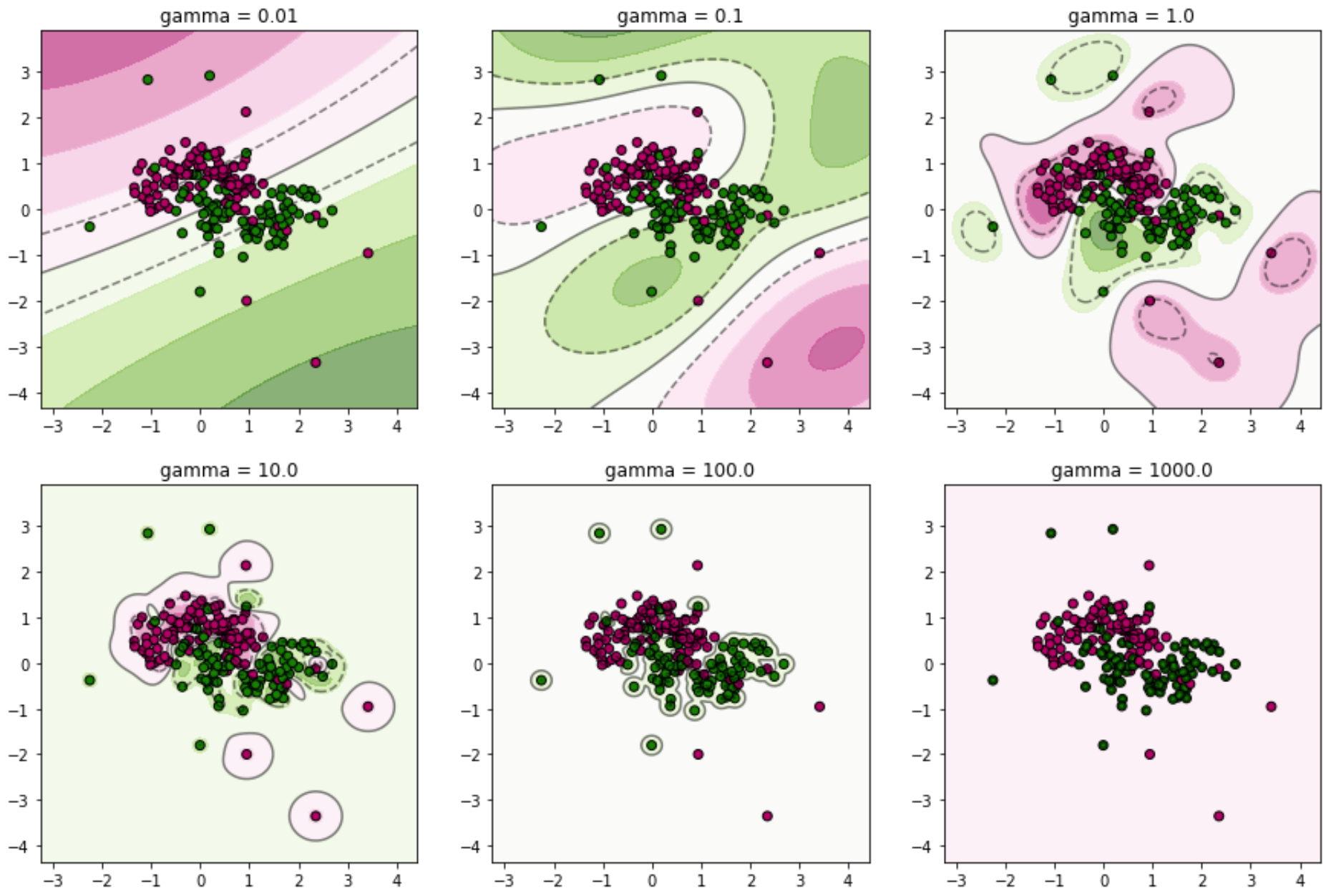
Discussion: How do the training error and the validation error change with γ? Based on the visualization of the models and their resulting classifiers, how does changing γ change the models? Explain in terms of the functional form of the RBF kernel, $\kappa(\mathbf{x},\,\mathbf{z}) \;=\; \exp(-\gamma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

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 $|gamma{best}$_.*

In [6]:
```
# 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()

idx = 0
for G in gamma_values:
    clf = SVC(C = 10,gamma=G,kernel='rbf')
    clf.fit(X_trn, y_trn)
    models[G]=clf
    y_trn_pred = clf.predict(X_trn)
    y_val_pred = clf.predict(X_val)
    trnErr[idx]=1-accuracy_score(y_trn,y_trn_pred)
    valErr[idx]=1-accuracy_score(y_val,y_val_pred)
    idx += 1

plt.figure(figsize=(10,8))
plt.plot(list(trnErr.keys()), list(trnErr.values()), marker='o', linewidth=3, markersize=10, color='#b30065')
plt.plot(list(valErr.keys()), list(valErr.values()), marker='D', linewidth=3, markersize=10, color='#178000')
plt.legend(['Train Error','Validation Error'], fontsize=16)
plt.xlabel('Gamma', fontsize=16)
plt.ylabel('Train/Validation error', fontsize=16)
plt.xticks(list(trnErr.keys()), ( '0.01', '0.1', '1.0', '10.0','100.0','1000.0'),fontsize=12)

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

minErr = 1
count_minErr = 1
bestVal = []
ls_gammaValues = list(gamma_values)

for v in valErr:
    if(valErr[v] < minErr):
        minErr = valErr[v]
        bestVal.clear()
        bestVal.append(ls_gammaValues[v])
    elif (valErr[v] == minErr):
        count_minErr +=1
        bestVal.append(ls_gammaValues[v])

print('Best gamma values: ',bestVal)
```

```
Best gamma values:  [1.0, 10.0]
```





---

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 [7]:
```
# Load the Breast Cancer Diagnosis data set; download the files from eLearning
# CSV files can be read easily using np.loadtxt()
#
cancer_trn = np.loadtxt(open("wdbc_trn.csv", "rb"), delimiter=",")
X_trn = cancer_trn[:,1:]
y_trn = cancer_trn[:,0]
cancer_tst = np.loadtxt(open("wdbc_tst.csv", "rb"), delimiter=",")
X_tst = cancer_tst[:,1:]
y_tst = cancer_tst[:,0]
cancer_val = np.loadtxt(open("wdbc_val.csv", "rb"), delimiter=",")
X_val = cancer_val[:,1:]
y_val = cancer_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 [8]:
```
C_range = np.arange(-2.0, 5.0, 1.0)
C_values = np.power(10.0, C_range)
ls_cValues = list(C_values)

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

bestC=[]
bestGamma=[]
tstErr=[]
minErr=1

for idxC,i in  enumerate(ls_cValues):
    for idxG,j in enumerate(ls_gammaValues):
        clf = SVC(C = i,gamma=j,kernel='rbf')
        clf.fit(X_trn, y_trn)
        y_trn_pred = clf.predict(X_trn)
        y_val_pred = clf.predict(X_val)
        y_tst_pred = clf.predict(X_tst)
        val_err = 1-accuracy_score(y_val,y_val_pred)
        tst_err = 1-accuracy_score(y_tst,y_tst_pred)
        if(val_err < minErr ):
            minErr = val_err
            bestC.clear()
            bestGamma.clear()
            tstErr.clear()
            bestC.append(ls_cValues[idxC])
            bestGamma.append(ls_gammaValues[idxG])
            tstErr.append(tst_err)
        elif (val_err == minErr ):
                bestC.append(ls_cValues[idxC])
                bestGamma.append(ls_gammaValues[idxG])
                tstErr.append(tst_err)

print('Best C value:', bestC)
print('Best Gamma value:', bestGamma)
print('Test Error:', [round(err, 2) for err in tstErr])
print('Minimum validation error:', round(minErr,2))
print('Max Validation accuracy:', str(round((1-minErr)*100,2)), '%')
```

```
Best C value: [100.0, 1000.0, 10000.0, 10000.0]
Best Gamma value: [0.01, 0.01, 0.001, 0.01]
Test Error: [0.03, 0.05, 0.06, 0.05]
Minimum validation error: 0.03
Max Validation accuracy: 97.39 %
```

---

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

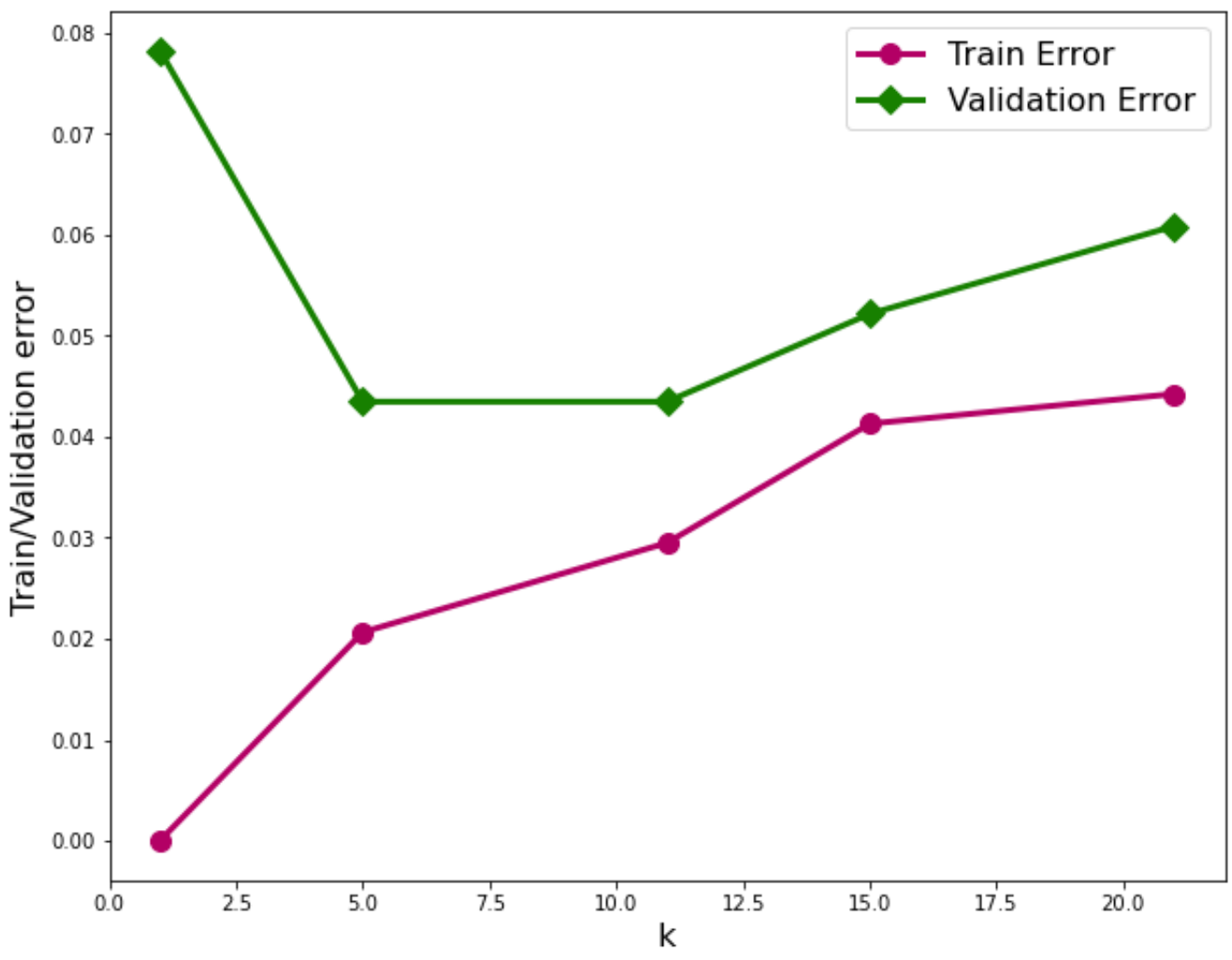
Use scikit-learn's k-nearest neighbor classifier to learn models for Breast Cancer Diagnosis with $k \in \{1, 5, 11, 15, 21\}$, with the kd-tree algorithm.

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 [9]:
```
k = [1,5,11,15,21]
trnErr = []
valErr = []

for idx,val in enumerate(k):
    neigh = KNeighborsClassifier(n_neighbors=val,algorithm='kd_tree')
    neigh.fit(X_trn,y_trn)
    y_trn_pred = neigh.predict(X_trn)
    y_val_pred = neigh.predict(X_val)
    trnErr.append(1-accuracy_score(y_trn,y_trn_pred))
    valErr.append(1-accuracy_score(y_val,y_val_pred))

plt.figure(figsize=(10,8))
plt.plot(k, trnErr, marker='o', linewidth=3, markersize=10, color='#b30065')
plt.plot(k, valErr, marker='D', linewidth=3, markersize=10, color='#178000')
plt.legend(['Train Error','Validation Error'], fontsize=16)
plt.xlabel('k', fontsize=16)
plt.ylabel('Train/Validation error', fontsize=16)
```

Out[9]:
```
Text(0, 0.5, 'Train/Validation error')
```



As we can see, we get least validation error when k is 5 and 11. With 11 as our neighbor we can get highest test accuracy.

In [10]:
```
neigh = KNeighborsClassifier(n_neighbors=11,algorithm='kd_tree')
neigh.fit(X_trn,y_trn)
y_tst_pred = neigh.predict(X_tst)

print('Test Accuracy:', round(accuracy_score(y_tst,y_tst_pred)*100,2),'%')
```

```
Test Accuracy: 97.39 %
```

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

The test set accuracies for KNN and SVM are 97.39% and 85% respectively, so KNN is definitely better and more accurate for this data set. The given dataset has 30 attributes and kNN works better with more number of attributes. Thus, for classification of this particular dataset kNN should be preferred.