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]:

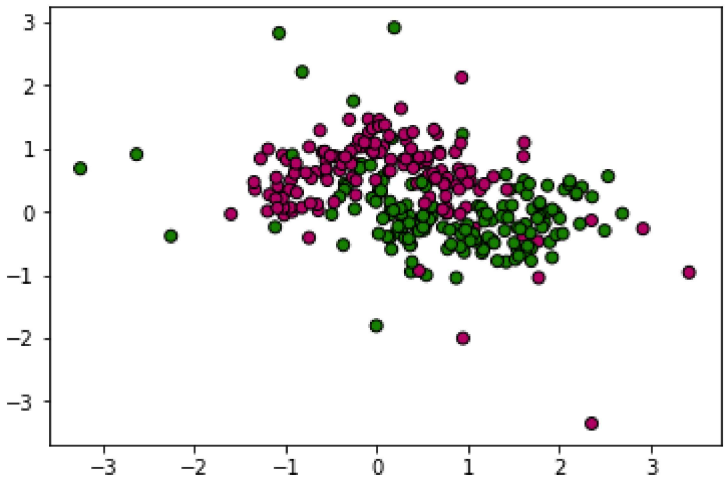
```
#
# DO NOT EDIT THIS FUNCTION; IF YOU WANT TO PLAY AROUND WITH DATA GENERATION,
# MAKE A COPY OF THIS FUNCTION AND THEN EDIT
#
import numpy as np
from sklearn.datasets import make_moons
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
from matplotlib.colors import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

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)

#print(X_trn.shape)
#print(y_trn.shape)
#print(y_trn)
```



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 (https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html) 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.

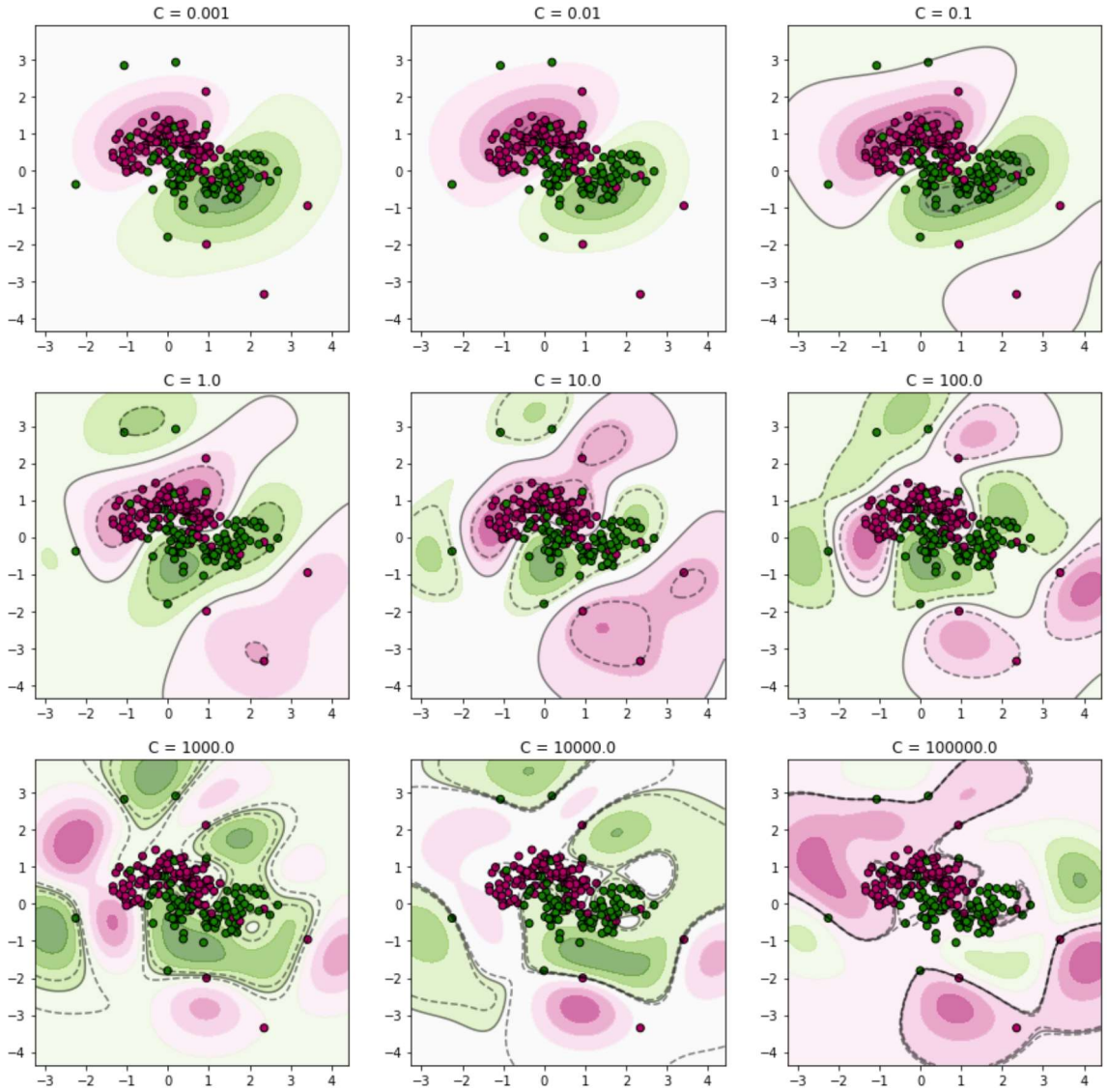
Discussion: How do the training error and the validation error change with C? Based on the visualization of the models and their resulting classifiers, how does changing C change the models? Explain in terms of minimizing the SVM's objective function $\frac{1}{2}\mathbf{w}'\mathbf{w} + C\,\Sigma_{i=1}^{n}\,\ell(\mathbf{w}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

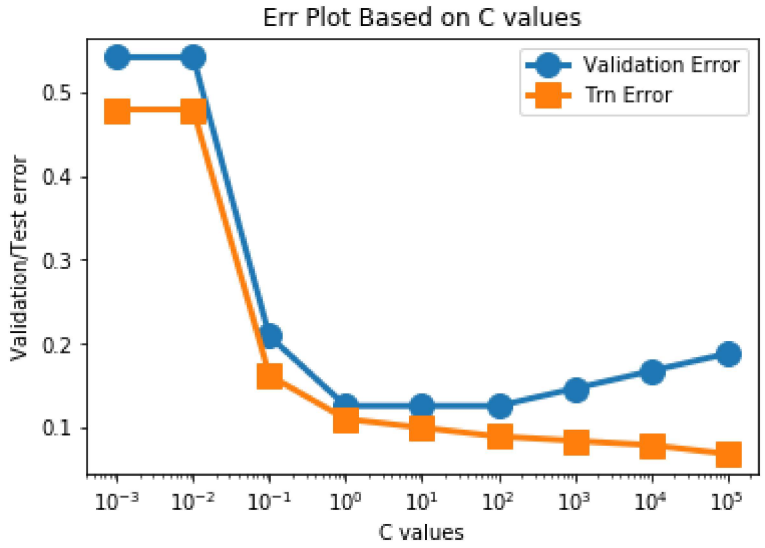
Final Model Selection: Use the validation set to select the best the classifier corresponding to the best value, C_{best}. Report the accuracy on the **test set** for this selected best SVM model. *Note: You should report a single number, your final test set accuracy on the model corresponding to $C{best}$_.*

In [4]:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets, svm
# 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()
#print(X_trn)

sdiv = np.std(X_trn)                                #Calculating the Standard deviation
g = 1/((X_trn.shape[1])*sdiv)                       #Calculating the g value based on the
given definition

for C in C_values:
  # Insert your code here to learn SVM models
    clf = svm.SVC(kernel='rbf', C= C, gamma = g)  #if we use 'scale' getting str error
 so calculated the g value above = 0.58
    k = clf.fit(X_trn, y_trn)
    models.update({C:k})
    s_trn= k.score(X_trn, y_trn)
    trnErr.update({C:1-s_trn})                      #(1-accuracy) gives the error over he
re
    s_val = k.score(X_val,y_val)
    valErr.update({C:1-s_val})
#print(valErr)
#print(trnErr)

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

#Plotting the Traning and Validation Errors
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 values')
plt.ylabel('Validation/Test error')
plt.xticks(list(valErr.keys()))
plt.legend(['Validation Error', 'Trn Error'])
plt.title("Err Plot Based on C values")
plt.xscale('log')

# Insert your code here to perform model selection
dif_Err = {key: abs(trnErr[key] - valErr.get(key, 0)) for key in trnErr.keys()}
#print(dif_Err)
BestValue_C= min(dif_Err, key=lambda k: dif_Err[k])       #Finding the best value of C
 by calculating the absolute
print("Best Value of C:",BestValue_C)

m = models[BestValue_C]
a =m.score(X_tst,y_tst)                                   #Calculating the accuracy on
 the best model selected
print("Accuracy on Test Set is:",a*100)
```

```
Best Value of C: 1.0
Accuracy on Test Set is: 83.33333333333334
```


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



Discussion part (a) : As we can see that here at (10pow0) i.e at "1" the error difference is less over here . And by on changing the "C" value we could see that the error is decreasing on the tranining set where as on the validation set it is increasing. By visualizing the above models we could see that the model is getting more complex as C value is increasing. The C value has to be stopped at 10pow0 even though the training error is decreasing, the validation error is increasing.

---

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 (https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html) 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.

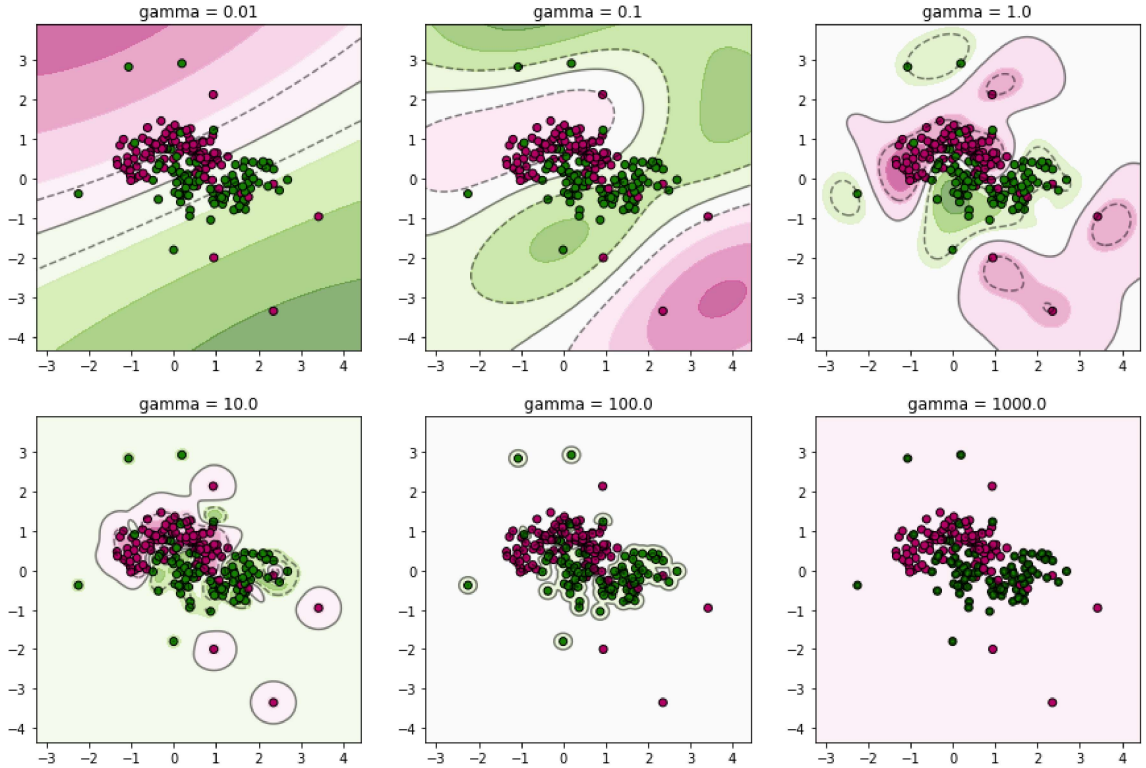
Discussion: How do the training error and the validation error change with γ? Based on the visualization of the models and their resulting classifiers, how does changing γ change the models? Explain in terms of the functional form of the RBF kernel, $\kappa(\mathbf{x},\ \mathbf{z})\ =\ \exp(-\gamma \cdot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

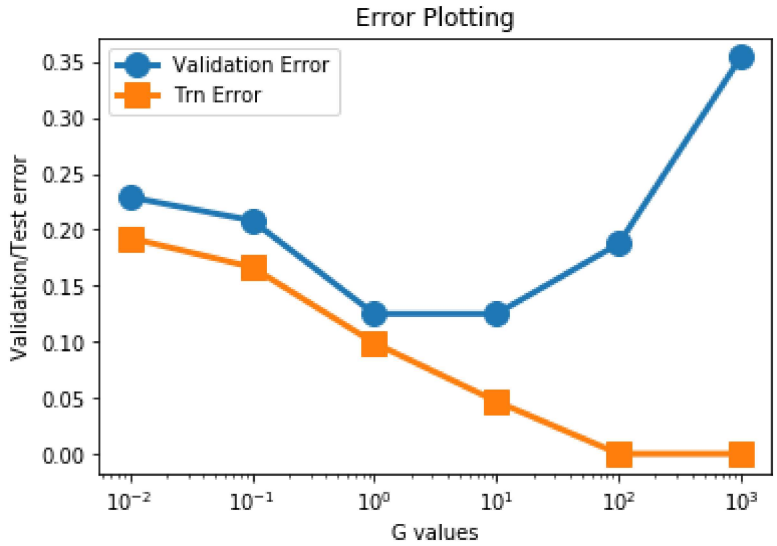
Final Model Selection: Use the validation set to select the best the classifier corresponding to the best value, γ_{best}. Report the accuracy on the **test set** for this selected best SVM model. *Note: You should report a single number, your final test set accuracy on the model corresponding to $\gamma*{best}$_.*

In [5]:

```
# 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()

for G in gamma_values:
  # Insert your code here to learn SVM models
    clf = svm.SVC(kernel='rbf', C= 10, gamma = G)            #Traning the model
    k = clf.fit(X_trn, y_trn)
    models.update({G:k})
    s_trn= k.score(X_trn, y_trn)
    trnErr.update({G:1-s_trn})                               #Updating the Error dictio
naries on predicted model
    s_val = k.score(X_val,y_val)
    valErr.update({G:1-s_val})
#print(valErr)
#print(trnErr)

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

#Plotting the Training and Validation Errors
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 values')
plt.ylabel('Validation/Test error')
plt.xticks(list(valErr.keys()))
plt.legend(['Validation Error', 'Trn Error'])
plt.title("Error Plotting")
plt.xscale('log')

# Insert your code here to perform model selection
dif_Err = {key: abs(trnErr[key] - valErr.get(key, 0)) for key in trnErr.keys()}
#print(dif_Err)
BestValue_G= min(dif_Err, key=lambda k: dif_Err[k])       #Finding the best value of G
 after calculating the absolute values
print("Best Value of Gamma:",BestValue_G)

m = models[BestValue_G]

acc =m.score(X_tst,y_tst)                                 #generates the accuracy on te
st data for Best Gamma value
print("Accuracy on Test Set is:",acc*100)
```

```
Best Value of Gamma: 1.0
Accuracy on Test Set is: 83.33333333333334
```





Discussion part (b) : As we can see that here at (10pow0) i.e at "1" the error difference is less over here . And on changing the gamma value we could see that the error is decreasing on the tranining set where as on the validation set it is increasing. By visualizing the above models we could see that the model is getting more complex as gamma value is increasing. The gamma value has to be stopped at 10pow0 even though the training error is decresing, the validation error is drastically increasing.

---

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

For this problem, we will use the Wisconsin Breast Cancer (https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic)) data set, which has already been pre-processed and partitioned into training, validation and test sets. Numpy's loadtxt (https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.loadtxt.html) command can be used to load CSV files.

In [6]:

```
# 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
#
import os
import numpy as np

with open('wdbc_trn.csv') as f_trn:                    #Loading data from Training datas
et
    ncols_trn = len(f_trn.readline().split(','))
print("Loading Training DataSet")
y_trn = np.loadtxt('wdbc_trn.csv', delimiter=',', usecols=0)
print("Y_trn :",y_trn.shape)
X_trn = np.loadtxt('wdbc_trn.csv', delimiter=',', usecols=range(1,ncols_trn))
print("X_trn:",X_trn.shape)

with open('wdbc_tst.csv') as f_tst:                    #Loading data from Testing datase
t
    ncols_tst = len(f_tst.readline().split(','))
print("Loading Testing DataSet")
y_tst = np.loadtxt('wdbc_tst.csv', delimiter=',', usecols=0)
print("Y_tst :",y_tst.shape)
X_tst = np.loadtxt('wdbc_tst.csv', delimiter=',', usecols=range(1,ncols_tst))
print("X_tst:",X_tst.shape)

with open('wdbc_val.csv') as f_val:                    #Loading data from Validation dat
aset
    ncols_val = len(f_val.readline().split(','))
print("Loading Validation DataSet")
y_val = np.loadtxt('wdbc_val.csv', delimiter=',', usecols=0)
print("Y_val :",y_val.shape)
X_val = np.loadtxt('wdbc_val.csv', delimiter=',', usecols=range(1,ncols_val))
print("X_val:",X_val.shape)
```

```
Loading Training DataSet
Y_trn : (339,)
X_trn: (339, 30)
Loading Testing DataSet
Y_tst : (115,)
X_tst: (115, 30)
Loading Validation DataSet
Y_val : (115,)
X_val: (115, 30)
```

Use scikit-learn's SVC (https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html) 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 [7]:

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

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import datasets, svm

C_range = np.arange(-2.0, 5.0, 1.0)            #Loading C parameter values on given con
dition of values
C_values = np.power(10.0, C_range)
#print(C_values)

gamma_range = np.arange(-3.0, 3.0, 1.0)        #Loading G parameter values on given con
dition of values
gamma_values = np.power(10.0, gamma_range)
#print(gamma_values)

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

for C in C_values:
    for G in gamma_values:
      # Insert your code here to learn SVM models
        clf = svm.SVC(kernel='rbf', C= C, gamma = G)
        k = clf.fit(X_trn, y_trn)                          #Training the model
        models.update({(C,G):k})
        s_trn= k.score(X_trn, y_trn)
        trnErr.update({(C,G):1-s_trn})
        s_val = k.score(X_val,y_val)               #Updating the Error dictionary value
s
        valErr.update({(C,G):1-s_val})
        s_tst = k.score(X_tst,y_tst)
        tstErr.update({(C,G):1-s_tst})

#print(models)
#print(trnErr)
#print(valErr)
#print(tstErr)

t_TrnErr = pd.DataFrame(list(trnErr.items()), columns=['C-G', 'ErrorValues'])
print("Training Error Table")
print(t_TrnErr)                                    #Printing the Training Error Table

print("\n")
t_ValErr = pd.DataFrame(list(valErr.items()), columns=['C-G', 'ErrorValues'])
print("Validation Error Table")                    #Printing the  Validation Error Table
print(t_ValErr)
print("\n")

#Model Selection and best pair of C-gamma selection
dif_Err = {key: abs(trnErr[key] - valErr.get(key, 0)) for key in trnErr.keys()}
BestValue_CG= min(dif_Err, key=lambda k: dif_Err[k])       #Finding the best value
print("Best combination of C-Gamma:",BestValue_CG)         #Selecting the best (C-Gamm
a) pair  - Got the pair as (0.01,0.001)
```

```
#Reporting the accuracy on the test Set

m = models[BestValue_CG]                                    #getting the best model
Acc_tst =m.score(X_tst,y_tst)                             #generates the accuracy on test data
 for Best Gamma value
print("Accuracy on Test Set is:",Acc_tst*100)
```

Training Error Table

	C-G	ErrorValues
0	(0.01, 0.001)	0.371681
1	(0.01, 0.01)	0.371681
2	(0.01, 0.1)	0.371681
3	(0.01, 1.0)	0.371681
4	(0.01, 10.0)	0.371681
5	(0.01, 100.0)	0.371681
6	(0.1, 0.001)	0.306785
7	(0.1, 0.01)	0.050147
8	(0.1, 0.1)	0.035398
9	(0.1, 1.0)	0.371681
10	(0.1, 10.0)	0.371681
11	(0.1, 100.0)	0.371681
12	(1.0, 0.001)	0.047198
13	(1.0, 0.01)	0.029499
14	(1.0, 0.1)	0.011799
15	(1.0, 1.0)	0.000000
16	(1.0, 10.0)	0.000000
17	(1.0, 100.0)	0.000000
18	(10.0, 0.001)	0.026549
19	(10.0, 0.01)	0.011799
20	(10.0, 0.1)	0.000000
21	(10.0, 1.0)	0.000000
22	(10.0, 10.0)	0.000000
23	(10.0, 100.0)	0.000000
24	(100.0, 0.001)	0.014749
25	(100.0, 0.01)	0.002950
26	(100.0, 0.1)	0.000000
27	(100.0, 1.0)	0.000000
28	(100.0, 10.0)	0.000000
29	(100.0, 100.0)	0.000000
30	(1000.0, 0.001)	0.005900
31	(1000.0, 0.01)	0.000000
32	(1000.0, 0.1)	0.000000
33	(1000.0, 1.0)	0.000000
34	(1000.0, 10.0)	0.000000
35	(1000.0, 100.0)	0.000000
36	(10000.0, 0.001)	0.000000
37	(10000.0, 0.01)	0.000000
38	(10000.0, 0.1)	0.000000
39	(10000.0, 1.0)	0.000000
40	(10000.0, 10.0)	0.000000
41	(10000.0, 100.0)	0.000000

Validation Error Table

	C-G	ErrorValues
0	(0.01, 0.001)	0.373913
1	(0.01, 0.01)	0.373913
2	(0.01, 0.1)	0.373913
3	(0.01, 1.0)	0.373913
4	(0.01, 10.0)	0.373913
5	(0.01, 100.0)	0.373913
6	(0.1, 0.001)	0.304348
7	(0.1, 0.01)	0.069565
8	(0.1, 0.1)	0.078261
9	(0.1, 1.0)	0.373913
10	(0.1, 10.0)	0.373913
11	(0.1, 100.0)	0.373913
12	(1.0, 0.001)	0.060870

```
13       (1.0, 0.01)     0.060870
14        (1.0, 0.1)     0.043478
15        (1.0, 1.0)     0.373913
16       (1.0, 10.0)     0.373913
17      (1.0, 100.0)     0.373913
18     (10.0, 0.001)     0.034783
19      (10.0, 0.01)     0.043478
20       (10.0, 0.1)     0.034783
21       (10.0, 1.0)     0.373913
22      (10.0, 10.0)     0.373913
23     (10.0, 100.0)     0.373913
24    (100.0, 0.001)     0.034783
25     (100.0, 0.01)     0.026087
26      (100.0, 0.1)     0.034783
27      (100.0, 1.0)     0.373913
28     (100.0, 10.0)     0.373913
29    (100.0, 100.0)     0.373913
30   (1000.0, 0.001)     0.034783
31    (1000.0, 0.01)     0.026087
32     (1000.0, 0.1)     0.034783
33     (1000.0, 1.0)     0.373913
34    (1000.0, 10.0)     0.373913
35   (1000.0, 100.0)     0.373913
36  (10000.0, 0.001)     0.026087
37   (10000.0, 0.01)     0.026087
38    (10000.0, 0.1)     0.034783
39    (10000.0, 1.0)     0.373913
40   (10000.0, 10.0)     0.373913
41  (10000.0, 100.0)     0.373913


Best combination of C-Gamma: (0.01, 0.001)
Accuracy on Test Set is: 62.60869565217392
```

---

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

Use scikit-learn's k-nearest neighbor (https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html) 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.

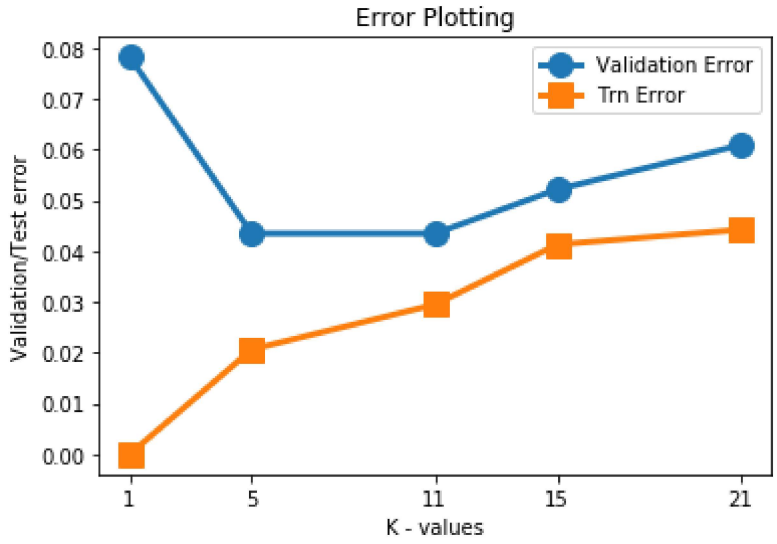
Final Model Selection: Use the validation set to select the best the classifier corresponding to the best parameter value, k_{best}. Report the accuracy on the **test set** for this selected best kNN model. *Note: You should report a single number, your final test set accuracy on the model corresponding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

In [8]:

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

import os
import numpy as np

with open('wdbc_trn.csv') as f_trn:                    #Loading data from Training datas
et
    ncols_trn = len(f_trn.readline().split(','))
#print("Loading Training DataSet")
y_trn = np.loadtxt('wdbc_trn.csv', delimiter=',', usecols=0)
#print("Y_trn :",y_trn.shape)
X_trn = np.loadtxt('wdbc_trn.csv', delimiter=',', usecols=range(1,ncols_trn))
#print("X_trn:",X_trn.shape)

with open('wdbc_tst.csv') as f_tst:                    #Loading data from Testing datase
t
    ncols_tst = len(f_tst.readline().split(','))
#print("Loading Testing DataSet")
y_tst = np.loadtxt('wdbc_tst.csv', delimiter=',', usecols=0)
#print("Y_tst :",y_tst.shape)
X_tst = np.loadtxt('wdbc_tst.csv', delimiter=',', usecols=range(1,ncols_tst))
#print("X_tst:",X_tst.shape)

with open('wdbc_val.csv') as f_val:                    #Loading data from Validation dat
aset
    ncols_val = len(f_val.readline().split(','))
#print("Loading Validation DataSet")
y_val = np.loadtxt('wdbc_val.csv', delimiter=',', usecols=0)
#print("Y_val :",y_val.shape)
X_val = np.loadtxt('wdbc_val.csv', delimiter=',', usecols=range(1,ncols_val))
#print("X_val:",X_val.shape)

import numpy as np
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier

k_values = {1,5,11,15,21}

models_k = dict()
trnErr = dict()
valErr = dict()
tstErr = dict()

for k in k_values:
    neigh = KNeighborsClassifier(n_neighbors=k,algorithm='kd_tree')     #Kd_Tree algori
thm
    c = neigh.fit(X_trn, y_trn)                                          #Model Tra
ining
    models_k.update({k:c})
    s_trn = c.score(X_trn, y_trn)
    trnErr.update({k:1-s_trn})
    s_val = c.score(X_val,y_val)
    valErr.update({k:1-s_val})
    s_tst = c.score(X_tst,y_tst)
    tstErr.update({k:1-s_tst})

#Plotting the training error and validation errors
```

```
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 - values')
plt.ylabel('Validation/Test error')
plt.xticks(list(valErr.keys()))
plt.legend(['Validation Error', 'Trn Error'])
plt.title("Error Plotting")


#Model Selection based on the best k value
dif_Err = {key: abs(trnErr[key] - valErr.get(key, 0)) for key in trnErr.keys()}
#print(dif_Err)
BestValue_K= min(dif_Err, key=lambda k: dif_Err[k])         #Finding the best value
print("Best K value:",BestValue_K)                          #Selecting the best k Value

#Reporting the accuracy on the test Set

m = models_k[BestValue_K]              #getting the model

Acc_tst =m.score(X_tst,y_tst)          #generates the accuracy on test data for Best Gam
ma value
print("Accuracy on Test Set is:",Acc_tst*100)
```

```
Best K value: 15
Accuracy on Test Set is: 94.78260869565217
```



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

Discussion: Here we got good accuracy in the KNN about 94% when compared to that of the SVM, it indicates that the given data is not easily separable using the decision planes. Basically it depends the type of the data set provided