CS 6375.001 Machine Learning

By

Prof. Anjum Chida

ASSIGNMENT – 4

VEDANT PARESH SHAH VXS200021

1. Support Vector Machines with Synthetic Data [50 points]

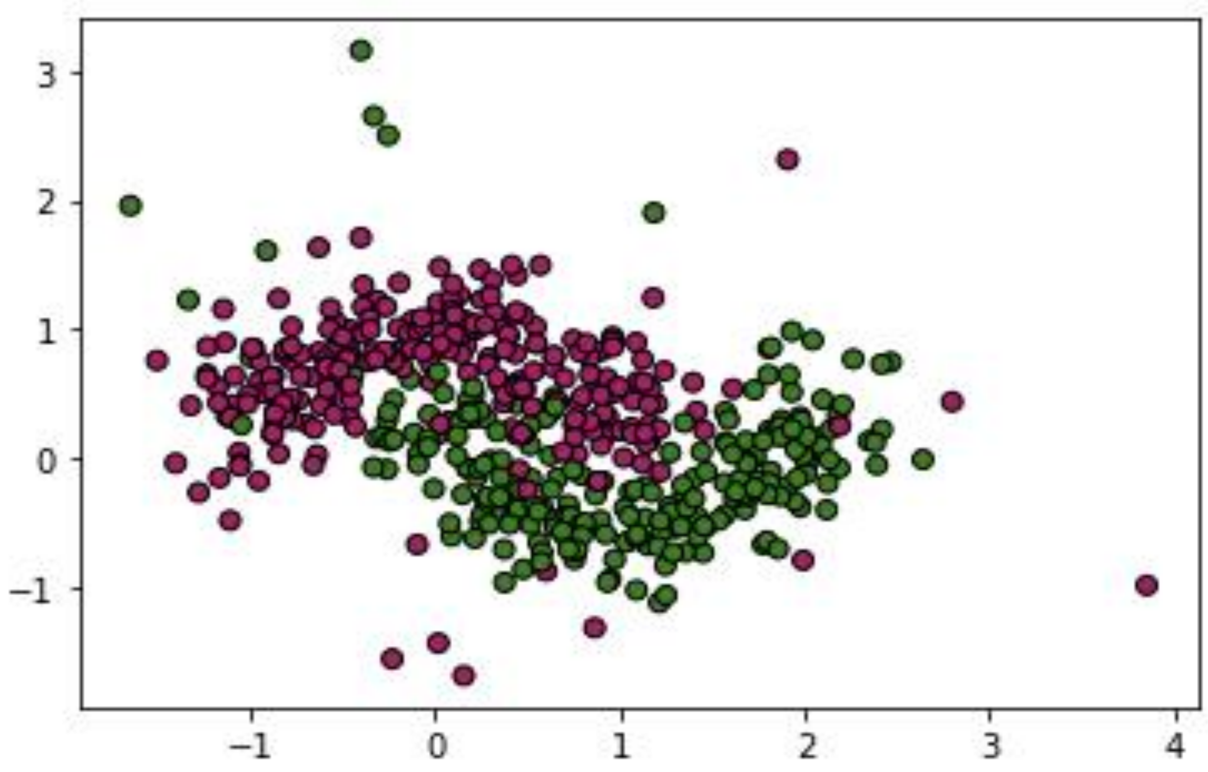
For this problem, we will generate synthetic data for a nonlinear binary classification problem and partition it into training, validation and test sets. Our goal is to understand the behavior of SVMs with Radial-Basis Function (RBF) kernels with different values of C and γ.

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]:
```python
#
#  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]:
```python
# Generate the data
n_samples = 500    # Total size of data set
(X_trn, y_trn), (X_val, y_val), (X_tst, y_tst) = generate_data(n_samples)
```

Here We have taken Size = 500 for the dataset



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).

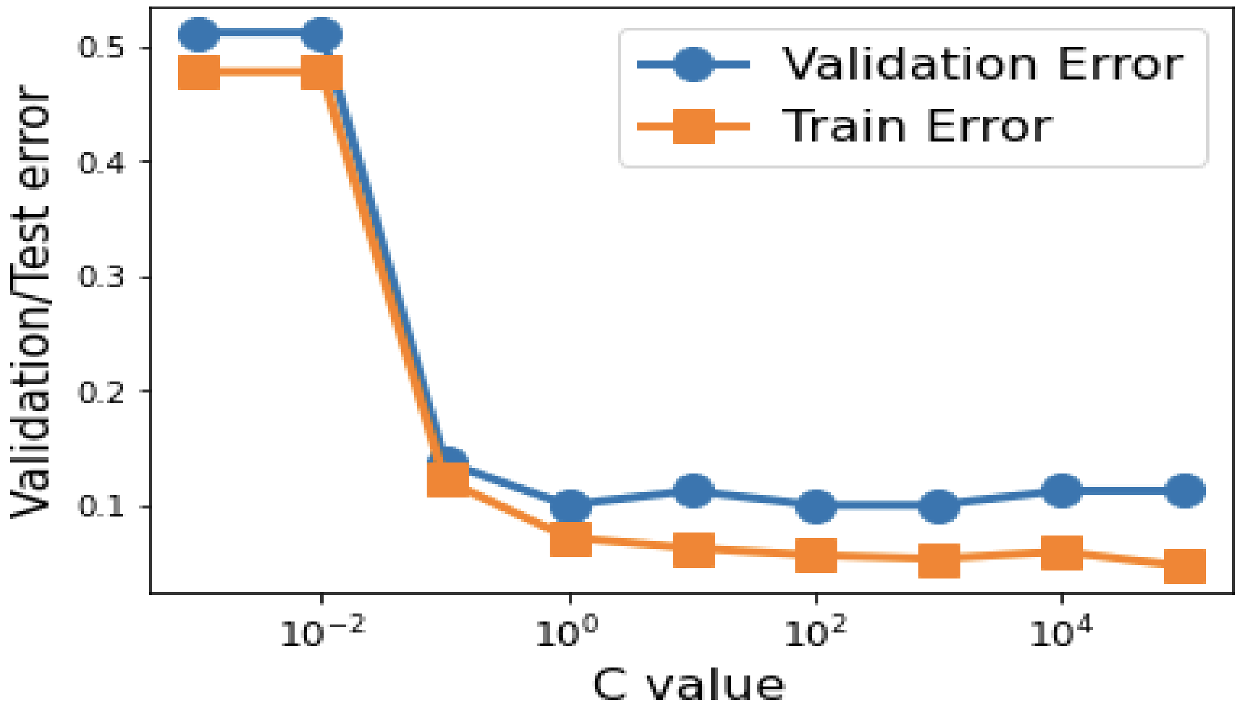
Plot: For each classifier, compute **both** the **training error** and the **validation error**. Plot them together, making sure to label the axes and each curve clearly.

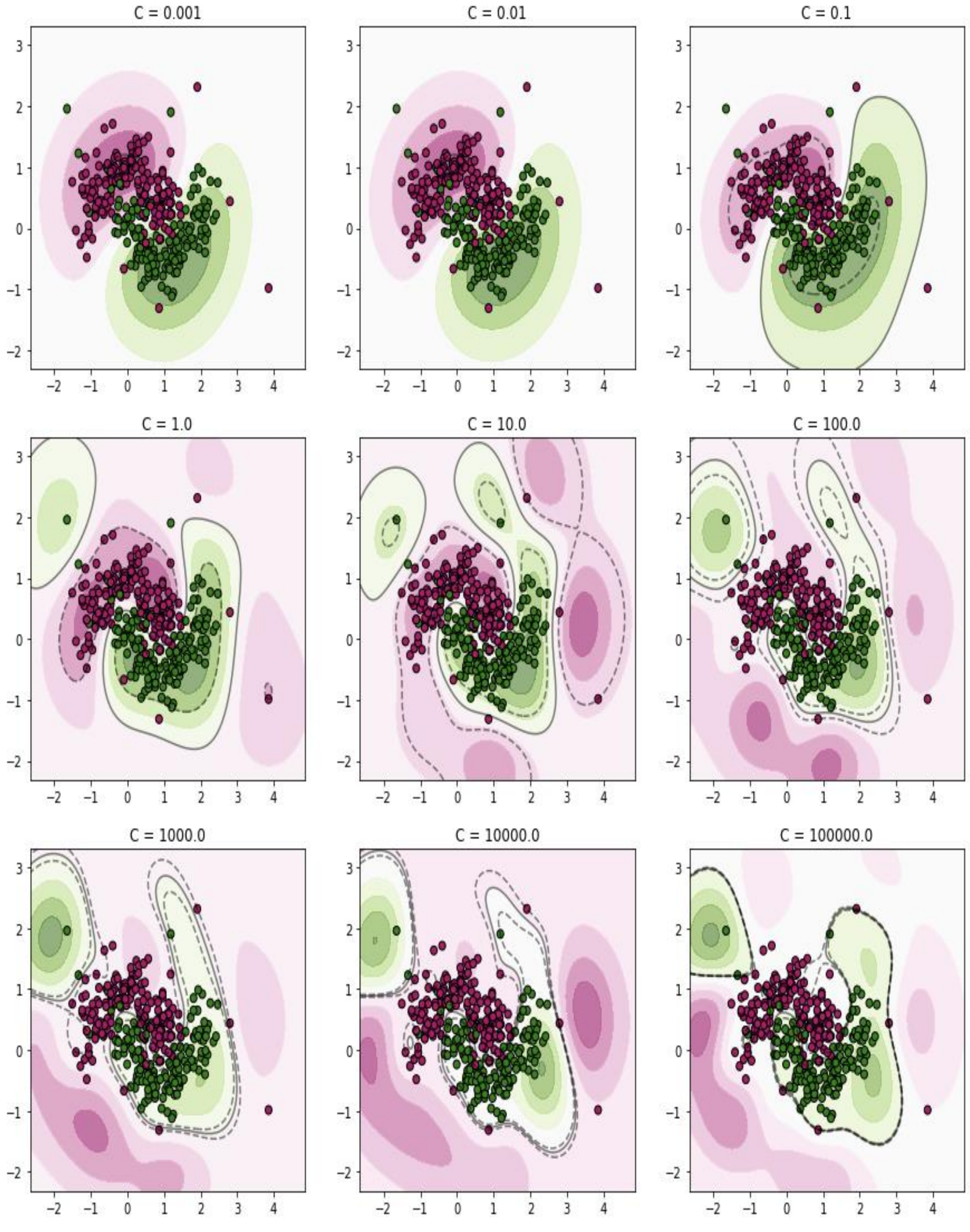
Discussion: How do the training error and the validation error change with C? Based on the visualization of the models and their resulting classifiers, how does changing C change the models? Explain in terms of minimizing the SVM's objective function $\frac{1}{2}\mathbf{w}'\mathbf{w} + C\,\Sigma_{i=1}^{n}\,\ell(\mathbf{w}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


        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')
```

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




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

<u>Output:</u>

```
List of Best C values on Validation: [1.0, 100.0, 1000.0]

Best C value is: 100.0

The accuracy for Best C value is: 91.0 %
```

<u>Observation</u>:

From the above Validation Error and Train Error Graph, we can observe that there is a very marginal difference between both of these Error's. The Errors are high with values of C = [10^{-3},10^{-2}]. After that, it drops drastically with the value of C = [10^{-1}]. Thereafter it drops marginally for values of C = [10^{0},10^{1},10^{2},10^{3},10^{4},10^{5}].

So, we can deduce that the Training Error decreases as the value of the C increases. But Validation Error decreases as the value of C increases till the Value of C = [10^{2}] and after that it increases marginally.

Thus, the Best Values of C are [1.0, 100.0, 1000.0]

From the Best Values the Best value of C is [100.0] with accuracy of 91%.

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 $C = 10$.

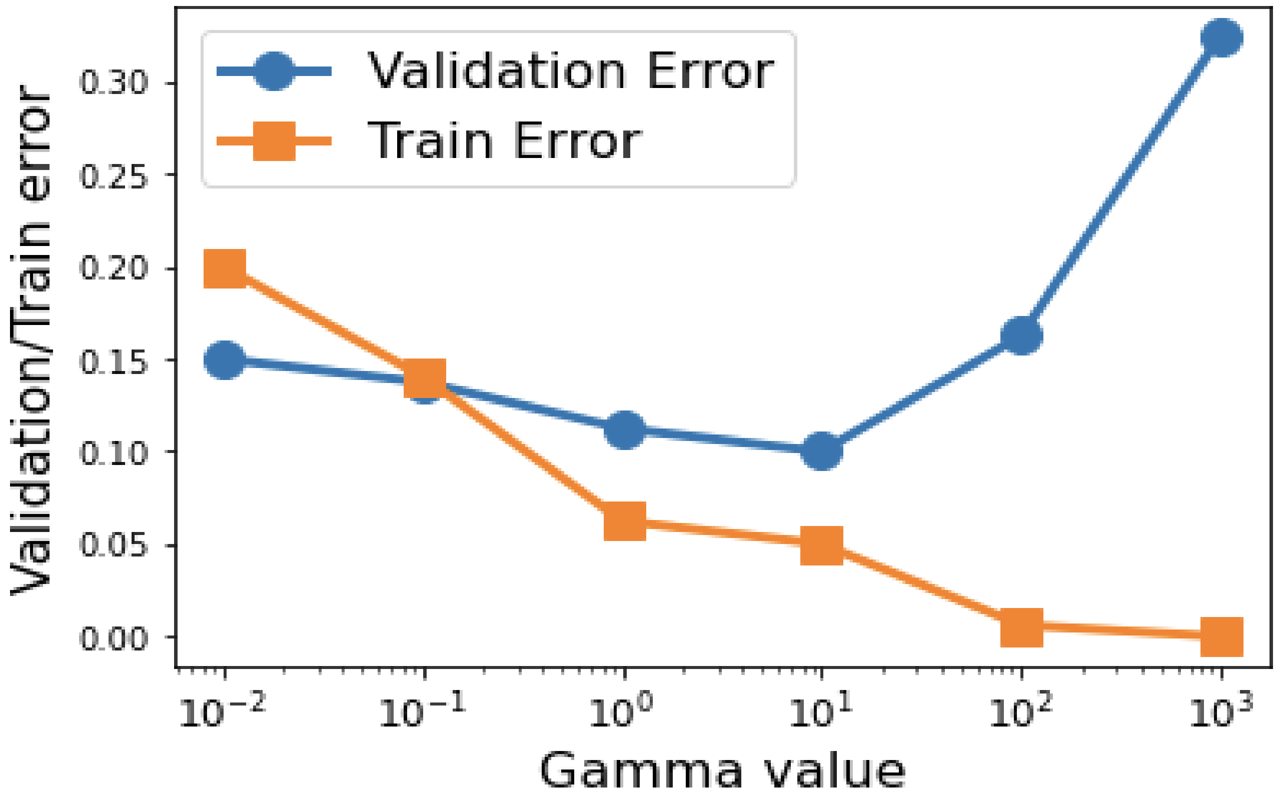
Plot: For each classifier, compute **both** the **training error** and the **validation error**. Plot them together, making sure to label the axes and each curve clearly.

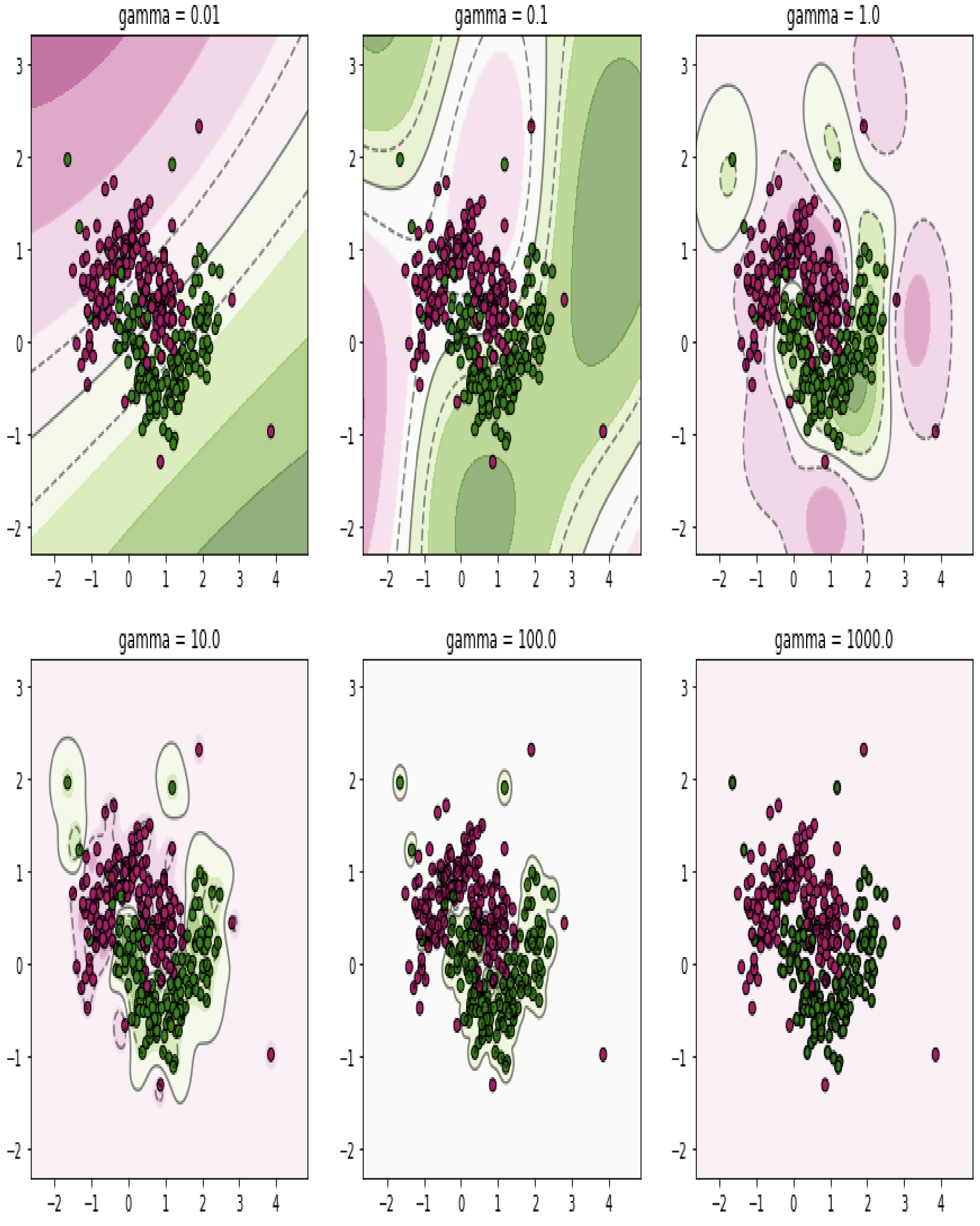
Discussion: How do the training error and the validation error change with γ? Based on the visualization of the models and their resulting classifiers, how does changing γ change the models? Explain in terms of the functional form of the RBF kernel, $\kappa(\mathbf{x}, \mathbf{z}) = \exp(-\gamma \cdot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

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 [ ]:
```python
# 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()
trnErr1 = dict()
valErr1 = dict()
tstErr1 = dict()

for G in gamma_values:
    clf = SVC(C=10, kernel='rbf', gamma=G)
    clf.fit(X_trn, y_trn)
    models[G] = clf
    trnErr1[G] = 1-clf.score(X_trn, y_trn)
    valErr1[G] = 1-clf.score(X_val, y_val)
    tstErr1[G] = 1-clf.score(X_tst, y_tst)

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

plt.figure()
plt.plot(valErr1.keys(), valErr1.values(), marker='o', linewidth=3, markersize=12)
plt.plot(trnErr1.keys(), trnErr1.values(), marker='s', linewidth=3, markersize =12)
plt.xlabel('Gamma value', fontsize=16)
plt.ylabel('Validation/Train error', fontsize=16)
plt.xticks(list(valErr1.keys()), fontsize=12)
plt.legend(['Validation Error', 'Train Error'], fontsize=16)
plt.xscale('log')
```

```
minErr1 = 1
bestG = []
for i in valErr1:
    if valErr1[i] < minErr1:
        minErr1 = valErr1[i]
        bestG = []
        bestG.append(i)
    elif minErr1==valErr1[i]:
        bestG.append(i)

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

minErr1 = 1
finalG = -1
for i in range(len(bestG)):
    if tstErr1[bestG[i]] < minErr1:
        finalG = bestG[i]
        minErr1 = tstErr1[bestG[i]]

print("Best Gamma value is:", finalG)
print("The accuracy for Best Gamma value is:", (1-tstErr1[finalG])*100,"%")
```





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

Output:

```
List of Best Gamma values on Validation: [10.0]

Best Gamma value is: 10.0

The accuracy for Best Gamma value is: 88.0 %
```

Observation:

From the above Validation Error and Train Error Graph, we can observe that there is a major difference between both of these Error's. The Errors are high with values of Gamma = [10^{-2}]. After that, it drops with the value of Gamma = [10^{1}]. Thereafter that training error keeps dropping but validation error keeps increasing.

So, we can deduce that the Training Error decreases as the value of the Gamma increases. But Validation Error decreases as the value of Gamma increases till the Value of Gamma = [10^{1}] and after that it increases marginally.

Thus, the Best Values of Gamma are [10.0]

From the Best Values the Best value of Gamma is [10.0] with accuracy of 88%.

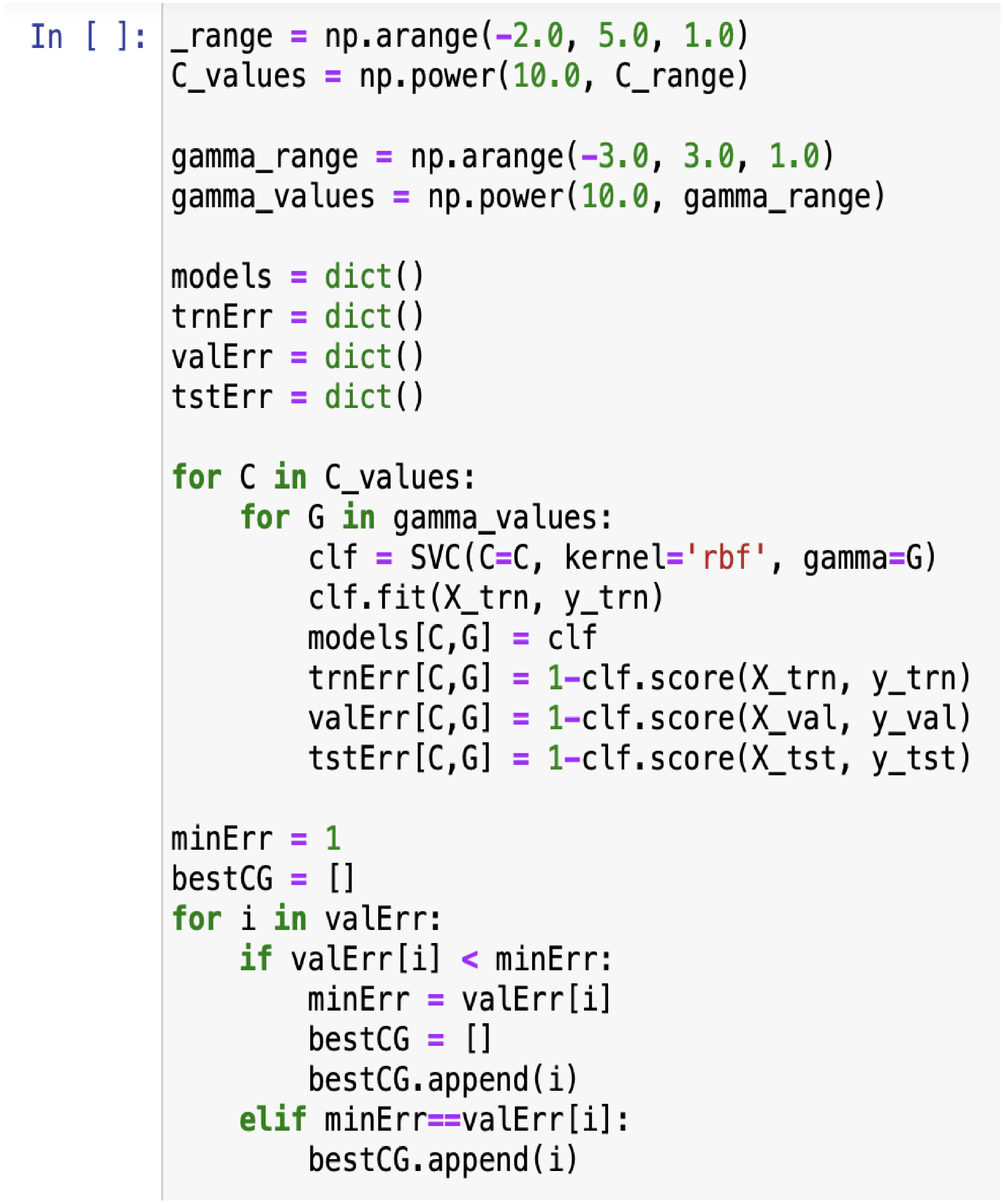
2. **Breast Cancer Diagnosis with Support Vector Machines**, 25 points.

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.

```
# 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("/Users/vedantshah/Downloads/wdbc_trn.csv", "rb"), delimiter=",")
X_trn=cancer_trn[:,1:]
y_trn=cancer_trn[:,0]
cancer_tst=np.loadtxt(open("/Users/vedantshah/Downloads/wdbc_tst.csv", "rb"), delimiter=",")
X_tst=cancer_tst[:,1:]
y_tst=cancer_tst[:,0]
cancer_val=np.loadtxt(open("/Users/vedantshah/Downloads/wdbc_val.csv", "rb"), delimiter=",")
X_val=cancer_val[:,1:]
y_val=cancer_val[:,0]
```

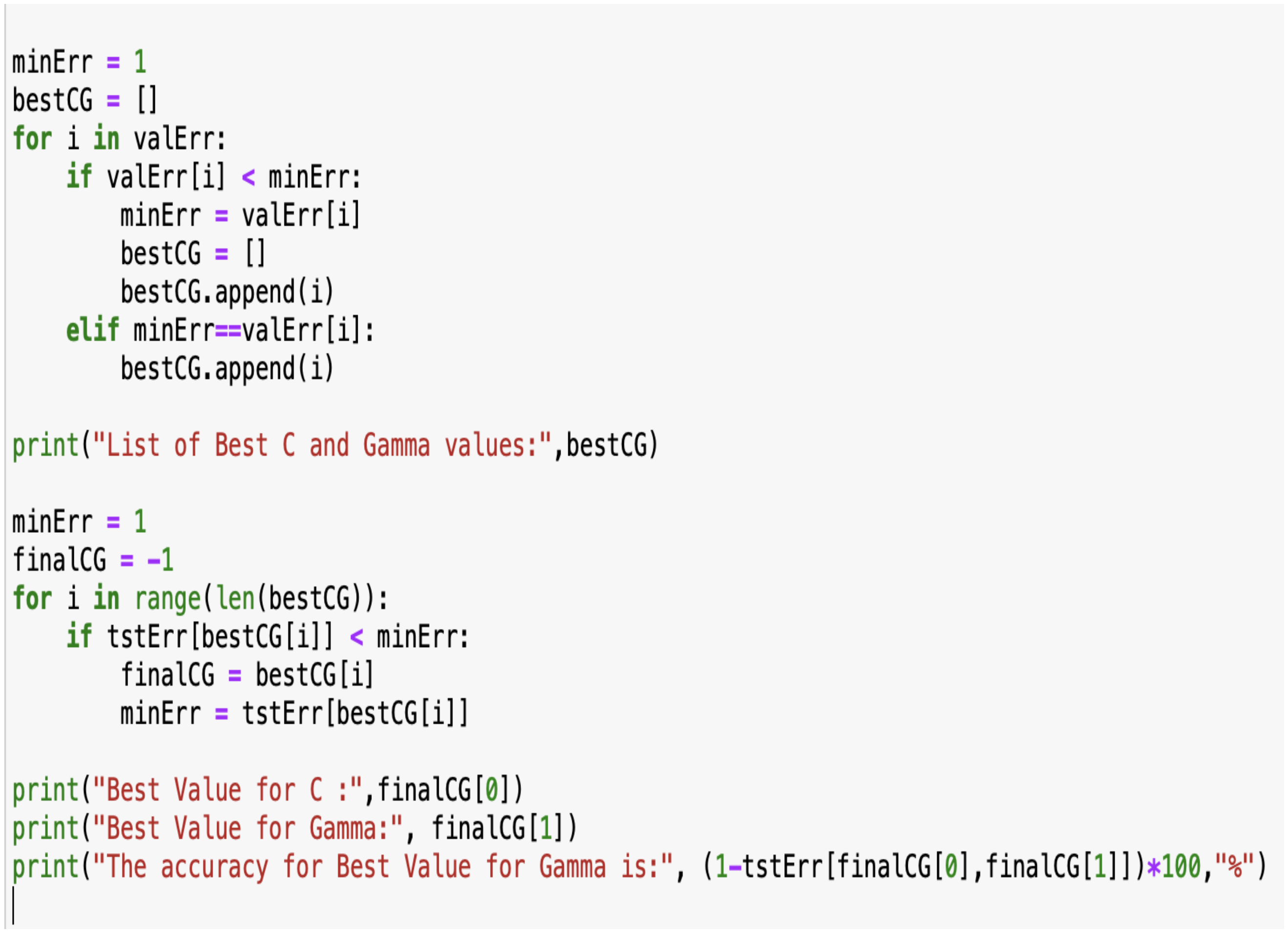
Use scikit-learn's SVC (https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html) function to learn SVM models with **radial-basis kernels** for **each combination** of $C \in \{10^{-2}, 10^{-1}, 1, 10^{1}, \cdots 10^{4}\}$ and $\gamma \in \{10^{-3}, 10^{-2}\ 10^{-1}, 1,\ 10,\ 10^{2}\}$. 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 [ ]:

```python
_range = np.arange(-2.0, 5.0, 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)

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

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)
```



```
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 Gamma 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])
print("Best Value for Gamma:", finalCG[1])
print("The accuracy for Best Value for Gamma is:", (1-tstErr[finalCG[0],finalCG[1]])*100,"%")
```

Output:

```
List of Best C and Gamma values: [(100.0, 0.01), (1000.0, 0.01),
(10000.0, 0.001), (10000.0, 0.01)]

Best Value for C : 100.0

Best Value for Gamma: 0.01

The accuracy for Best Value for Gamma is: 96.52173913043478 %
```

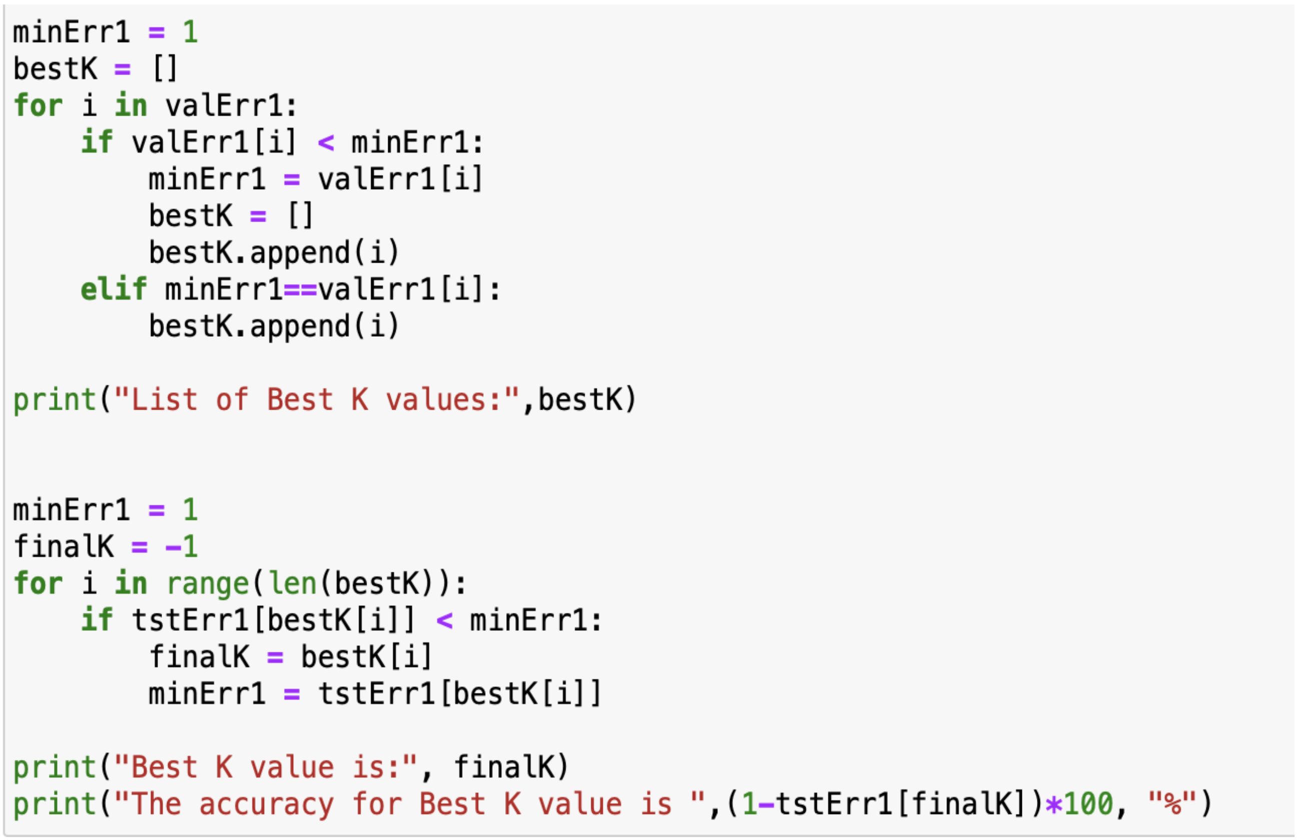
3. **Breast Cancer Diagnosis with k-Nearest Neighbors**, 25 points.

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

models1 = dict()
trnErr1 = dict()
valErr1 = dict()
tstErr1 = 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)
    models1[K] = kd
    trnErr1[K] = 1-kd.score(X_trn, y_trn)
    valErr1[K] = 1-kd.score(X_val, y_val)
    tstErr1[K] = 1-kd.score(X_tst, y_tst)

plt.figure()
plt.plot(valErr1.keys(), valErr1.values(), marker='o', linewidth=3, markersize=12)
plt.plot(trnErr1.keys(), trnErr1.values(), marker='s', linewidth=3, markersize =12)
plt.xlabel('K value', fontsize=16)
plt.ylabel('Validation/Test error', fontsize=16)
plt.xticks(list(valErr1.keys()), fontsize=12)
plt.legend(['Validation Error', 'Test Error'], fontsize=16)
plt.xscale('log')
```



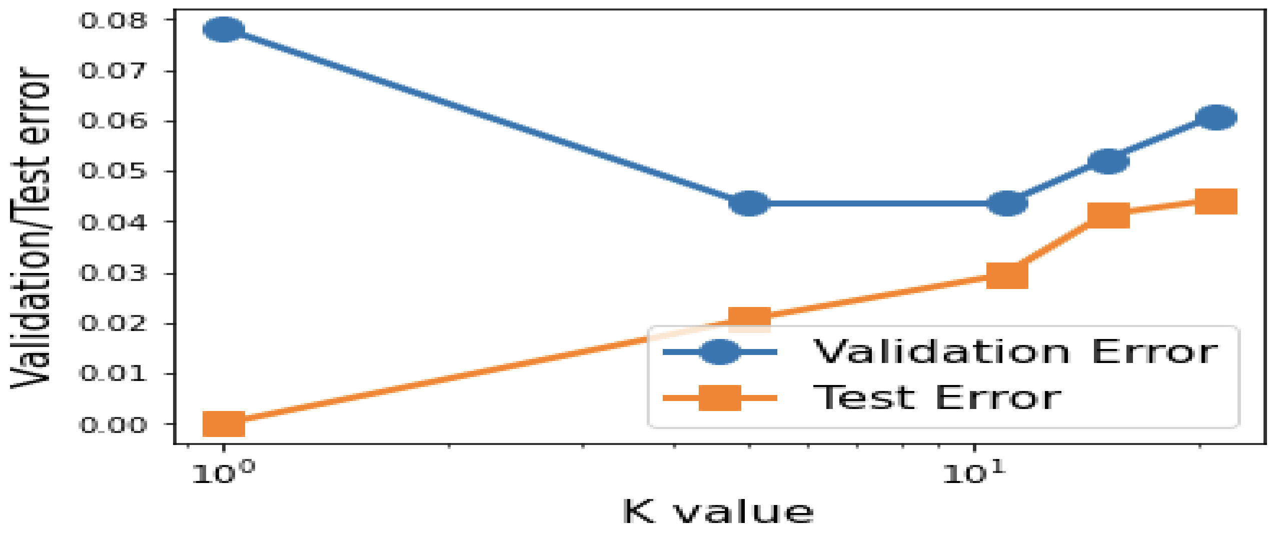
```
minErr1 = 1
bestK = []
for i in valErr1:
    if valErr1[i] < minErr1:
        minErr1 = valErr1[i]
        bestK = []
        bestK.append(i)
    elif minErr1==valErr1[i]:
        bestK.append(i)

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


minErr1 = 1
finalK = -1
for i in range(len(bestK)):
    if tstErr1[bestK[i]] < minErr1:
        finalK = bestK[i]
        minErr1 = tstErr1[bestK[i]]

print("Best K value is:", finalK)
print("The accuracy for Best K value is ",(1-tstErr1[finalK])*100, "%")
```

Output:

List of Best K values: [5, 11]

Best K value is: 11

The accuracy for Best K value is 97.3913043478261 %



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

Answer:

kNN gives a better accuracy of 97.391% whereas SVM gives an accuracy of 96.521% on the test data. Also, kNN works better for data with more number of attributes. The given dataset has 30 attributes and hence kNN is preffered.