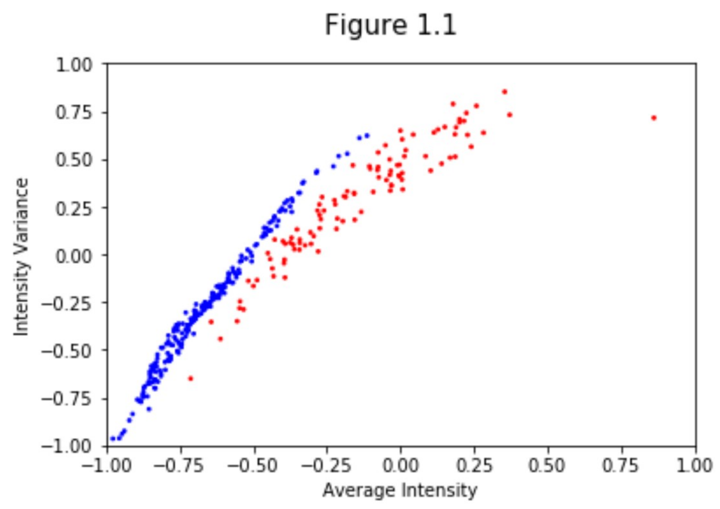
In [47]:
```python
#From the console, run the following
#pip install numpy
#pip install scipy
#pip install scikit-learn
#pip install matplotlib

# Import required packages here (after they are installed)
import numpy as np
import pandas as pd
from sklearn.neighbors import KNeighborsClassifier
import matplotlib.pyplot as mp
from pylab import show
from sklearn.model_selection import cross_val_score
from sklearn.decomposition import KernelPCA
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC

# Load data. csv file should be in the same folder as the notebook for this to wor
k, otherwise
# give data path.
data = np.loadtxt("data.csv")
```

In [48]:
```python
#shuffle the data and select training and test data
np.random.seed(100)
np.random.shuffle(data)


features = []
digits = []


for row in data:
    #import the data and select only the 1's and 5's
    if(row[0]==1 or row[0]==5):
        features.append(row[1:])
        digits.append(str(row[0]))


#Select the proportion of data to use for training.
#Notice that we have set aside 80% of the data for testing
numTrain = int(len(features)*.2)

trainFeatures = features[:numTrain]
testFeatures = features[numTrain:]
trainDigits = digits[:numTrain]
testDigits = digits[numTrain:]

#print(trainFeatures)
#trainFeatures[0]
```

In [49]:

```
#Convert the 256D data (trainFeatures) to 2D data
#We need X and Y for plotting and simpleTrain for building the model.
#They contain the same points in a different arrangement

X = []
Y = []
simpleTrain = []

#Colors will be passed to the graphing library to color the points.
#1's are blue: "b" and 5's are red: "r"
colors = []
#legends = []
for index in range(len(trainFeatures)):
    #print(index)
    #break
    #produce the 2D dataset for graphing/training and scale the data so it is in th
e [-1,1] square
    xNew = 2*np.average(trainFeatures[index])+.75
    yNew = 3*np.var(trainFeatures[index])-1.5
    X.append(xNew)
    Y.append(yNew)
    simpleTrain.append([xNew,yNew])
    #trainDigits will still be the value we try to classify. Here it is the string
"1.0" or "5.0"
    if(trainDigits[index]=="1.0"):
        colors.append("b")
        #legends.append("1")
    else:
        colors.append("r")
        #legends.append("5")

#plot the data points
### https://matplotlib.org/api/_as_gen/matplotlib.pyplot.scatter.html
fig = mp.figure()
mp.scatter(X,Y,s=3,c=colors)

#specify the axes
mp.xlim(-1,1)
mp.xlabel("Average Intensity")
mp.ylim(-1,1)
mp.ylabel("Intensity Variance")

#Labeling the plot
#mp.legend(['1'])
#mp.legend(legends)
fig.suptitle('Figure 1.1', fontsize=15)

#display the current graph
show()
```



1. FEATURE EXTRACTION

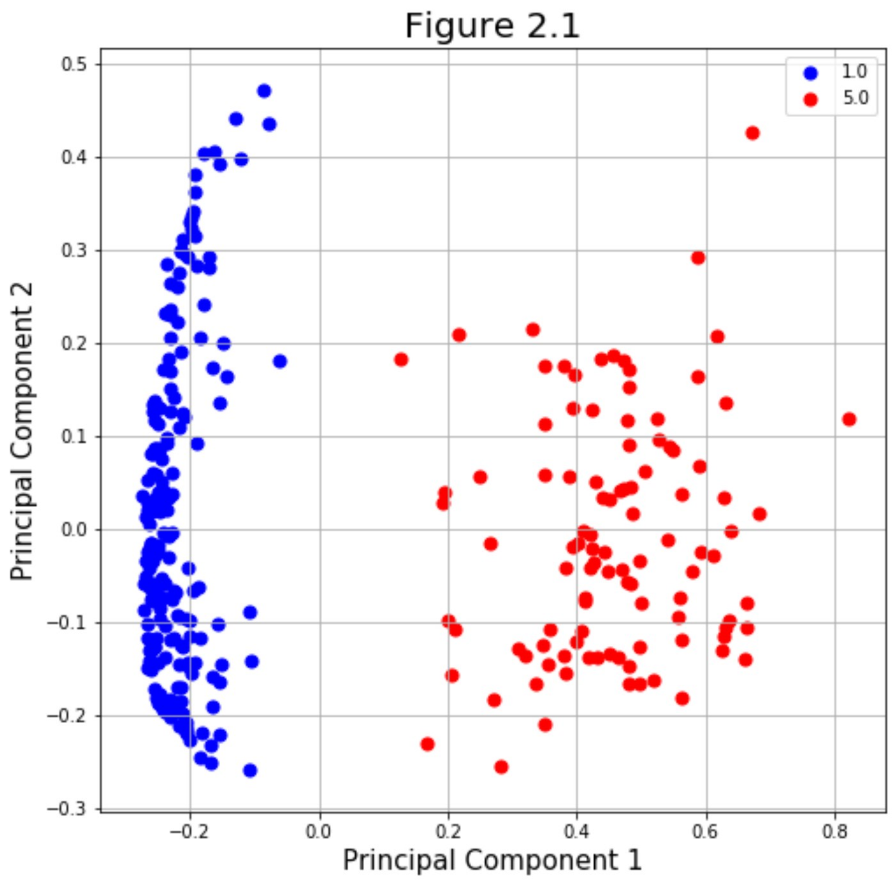
```
In [4]: len(trainFeatures)

Out[4]: 312
```

1a) KERNEL DEGREE = 1

```
In [5]: kpca = KernelPCA(n_components=2, kernel='poly', degree=1)
        PrincipalsComponents = kpca.fit_transform(trainFeatures)
        PrincipalsComponentsDF = pd.DataFrame(data = PrincipalsComponents, columns = ['prin
        cipal component 1', 'principal component 2'])

        print(PrincipalsComponentsDF.head())
        trainDigitsDF = pd.DataFrame(trainDigits, columns = ['Digit'])
        finalDF = pd.concat([PrincipalsComponentsDF, trainDigitsDF], axis = 1)
        print(finalDF.head())
        print("Length of Train Features:", len(trainFeatures))
        print("Length of Train Digits:", len(trainDigits))
```

```
   principal component 1  principal component 2
0               0.662562              -0.078380
1              -0.249190              -0.186351
2               0.387871               0.056345
3               0.636381              -0.098758
4              -0.171666               0.280755
   principal component 1  principal component 2 Digit
0               0.662562              -0.078380   5.0
1              -0.249190              -0.186351   1.0
2               0.387871               0.056345   5.0
3               0.636381              -0.098758   5.0
4              -0.171666               0.280755   1.0
Length of Train Features: 312
Length of Train Digits: 312
```

In [6]:
```
fig = mp.figure(figsize = (8,8))
ax = fig.add_subplot(1,1,1)
ax.set_xlabel('Principal Component 1', fontsize = 15)
ax.set_ylabel('Principal Component 2', fontsize = 15)
ax.set_title('Figure 2.1', fontsize = 20)
targets = ['1.0', '5.0']
colors = ['b', 'r']
for target, color in zip(targets,colors):
    indicesToKeep = finalDF['Digit'] == target
    ax.scatter(finalDF.loc[indicesToKeep, 'principal component 1']
               , finalDF.loc[indicesToKeep, 'principal component 2']
               , c = color
               , s = 50)
ax.legend(targets)
ax.grid()
```



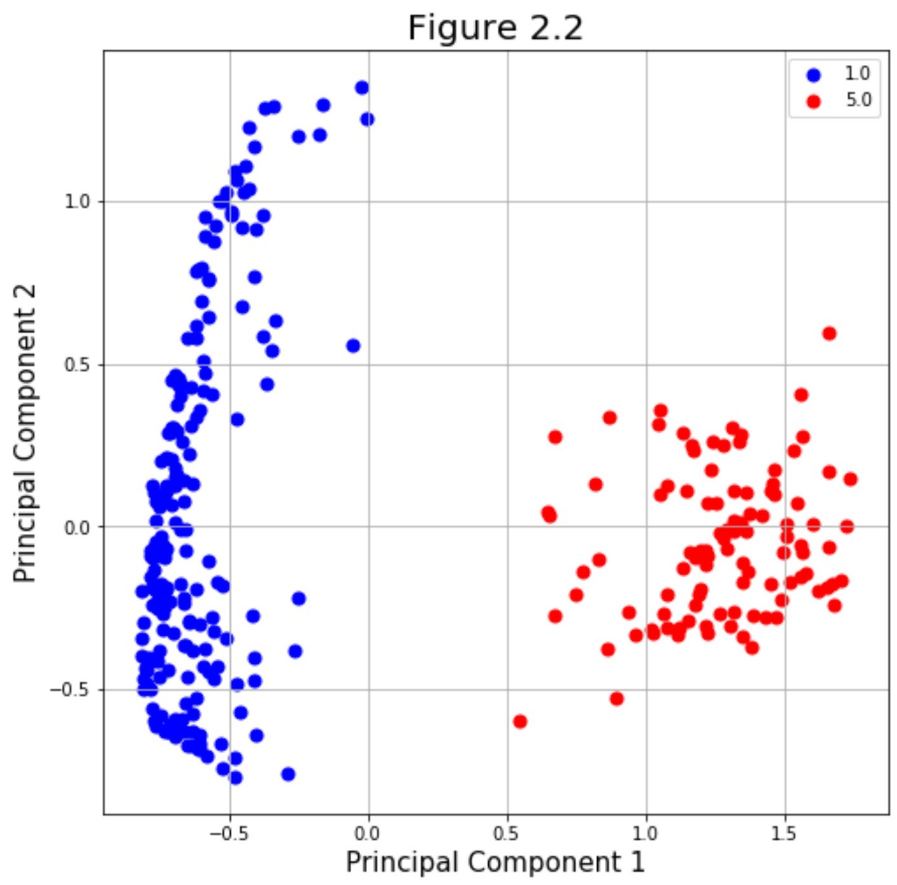
1a) Yes, the kernel PCA features seem to better seperate the data than the extracted features in HW1. It is because, we can see that the variation among each class is better seperated in Figure 2.1 than Figure 1.1 While in HW1 extracted features, the classes are very closely differentiated. i.e., their is a very small discrimination space between the blue and red points. Whereas, in Kernel PCA, due to the transformation, the data along PCA1 axis, their is a large discrimination distance between the two classes (blue and red). And hence, PCA explains the seperation of data better.

1b) KERNEL DEGREE = 2

```
In [7]: kpca3 = KernelPCA(n_components=2, kernel='poly', degree=3)
        PrincipalsComponents_d3 = kpca3.fit_transform(trainFeatures)
        PrincipalsComponentsDF_d3 = pd.DataFrame(data = PrincipalsComponents_d3, columns =
        ['principal component 1', 'principal component 2'])

        print(PrincipalsComponentsDF_d3.head())
        trainDigitsDF = pd.DataFrame(trainDigits, columns = ['Digit'])
        finalDF_d3 = pd.concat([PrincipalsComponentsDF_d3, trainDigitsDF], axis = 1)
        print(finalDF_d3.head())
        print("Length of Train Features:", len(trainFeatures))
        print("Length of Train Digits:", len(trainDigits))
```

```
   principal component 1  principal component 2
0               1.704466              -0.165921
1              -0.762189              -0.610113
2               1.149909               0.109357
3               1.653838              -0.186883
4              -0.400848               0.914819
   principal component 1  principal component 2 Digit
0               1.704466              -0.165921   5.0
1              -0.762189              -0.610113   1.0
2               1.149909               0.109357   5.0
3               1.653838              -0.186883   5.0
4              -0.400848               0.914819   1.0
Length of Train Features: 312
Length of Train Digits: 312
```

```
In [8]: fig = mp.figure(figsize = (8,8))
        ax = fig.add_subplot(1,1,1)
        ax.set_xlabel('Principal Component 1', fontsize = 15)
        ax.set_ylabel('Principal Component 2', fontsize = 15)
        ax.set_title('Figure 2.2', fontsize = 20)
        targets = ['1.0', '5.0']
        colors = ['b', 'r']
        for target, color in zip(targets,colors):
            indicesToKeep = finalDF_d3['Digit'] == target
            ax.scatter(finalDF_d3.loc[indicesToKeep, 'principal component 1']
                       , finalDF_d3.loc[indicesToKeep, 'principal component 2']
                       , c = color
                       , s = 50)
        ax.legend(targets)
        ax.grid()
```



1. LOGISTIIC REGRESSION

2a) Logistic Regression c=0.01 : L2 Regularization

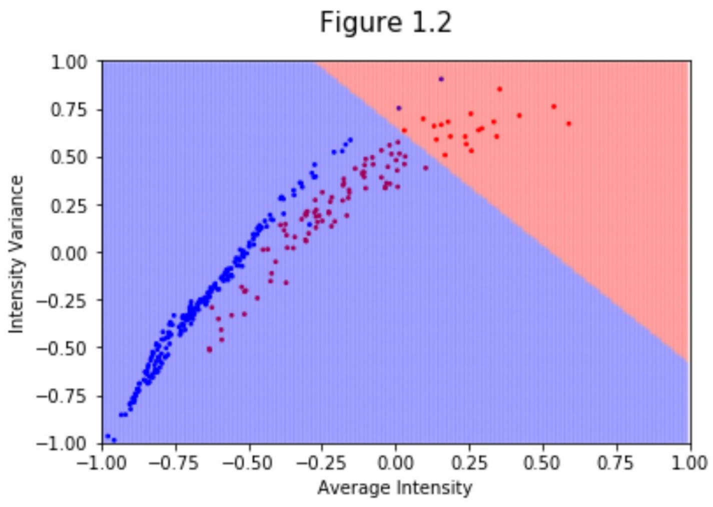
In [11]:
```
logistic_m1 = LogisticRegression(penalty='l2', C=0.01)
clsf = logistic_m1.fit(simpleTrain, trainDigits)
m_score = clsf.score(simpleTrain, trainDigits)
print("Score of the model is:", m_score)
```

```
Score of the model is: 0.7275641025641025

C:\Users\Anu\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:432: F
utureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a solve
r to silence this warning.
  FutureWarning)
```

In [12]:
```
# create the model
# https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsCla
ssifier.html

# Declare Model
model = logistic_m1
# Fit model to our data
model.fit(simpleTrain,trainDigits)

# Lists to hold inpoints, predictions and assigned colors
xPred = []
yPred = []
cPred = []
# Use input points to get predictions here
for xP in range(-100,100):
    xP = xP/100.0
    for yP in range(-100,100):
        yP = yP/100.0
        xPred.append(xP)
        yPred.append(yP)
        if(model.predict([[xP,yP]])=="1.0"):
            cPred.append("b")
        else:
            cPred.append("r")
```

```
C:\Users\Anu\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:432: F
utureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a solve
r to silence this warning.
  FutureWarning)
```

```
In [13]: ## Visualize Results
         #plot the points
         fig = mp.figure()
         mp.scatter(X,Y,s=3,c=colors)

         #plot the regions
         mp.scatter(xPred,yPred,s=3,c=cPred,alpha=.1)

         #setup the axes
         mp.xlim(-1,1)
         mp.xlabel("Average Intensity")
         mp.ylim(-1,1)
         mp.ylabel("Intensity Variance")

         #Label the figure
         fig.suptitle('Figure 1.2', fontsize=15)
         show()
```



2b) Logistic Regression c=2.0 : L2 Regularization

```
In [14]: logistic_m2 = LogisticRegression(penalty='l2', C=2.0)
         clsf = logistic_m2.fit(simpleTrain, trainDigits)
         m_score = clsf.score(simpleTrain, trainDigits)
         print("Score of the model is:", m_score)
```

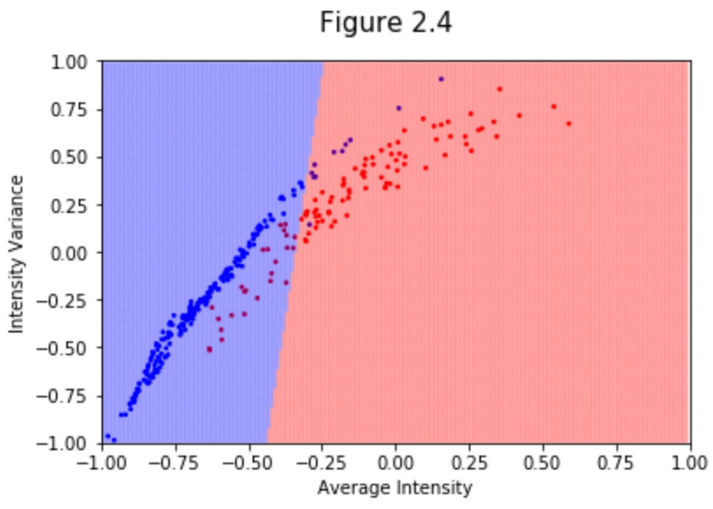
```
Score of the model is: 0.8846153846153846

C:\Users\Anu\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:432: F
utureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a solve
r to silence this warning.
  FutureWarning)
```

In [15]:

```
# create the model
# https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsCla
ssifier.html

# Declare Model
model = logistic_m2
# Fit model to our data
model.fit(simpleTrain,trainDigits)

# Lists to hold inpoints, predictions and assigned colors
xPred = []
yPred = []
cPred = []
# Use input points to get predictions here
for xP in range(-100,100):
    xP = xP/100.0
    for yP in range(-100,100):
        yP = yP/100.0
        xPred.append(xP)
        yPred.append(yP)
        if(model.predict([[xP,yP]])=="1.0"):
            cPred.append("b")
        else:
            cPred.append("r")
```

```
C:\Users\Anu\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:432: F
utureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a solve
r to silence this warning.
  FutureWarning)
```

In [16]:
```
## Visualize Results
#plot the points
fig = mp.figure()
mp.scatter(X,Y,s=3,c=colors)

#plot the regions
mp.scatter(xPred,yPred,s=3,c=cPred,alpha=.1)

#setup the axes
mp.xlim(-1,1)
mp.xlabel("Average Intensity")
mp.ylim(-1,1)
mp.ylabel("Intensity Variance")

#Label the figure
fig.suptitle('Figure 2.4', fontsize=15)
show()
```



Graduate Student question: L1 Regularization (c=0.01)

In [17]:
```
logistic_m3 = LogisticRegression(penalty='l1', C=0.01)
clsf = logistic_m3.fit(simpleTrain, trainDigits)
m_score = clsf.score(simpleTrain, trainDigits)
print("Score of the model is:", m_score)
```

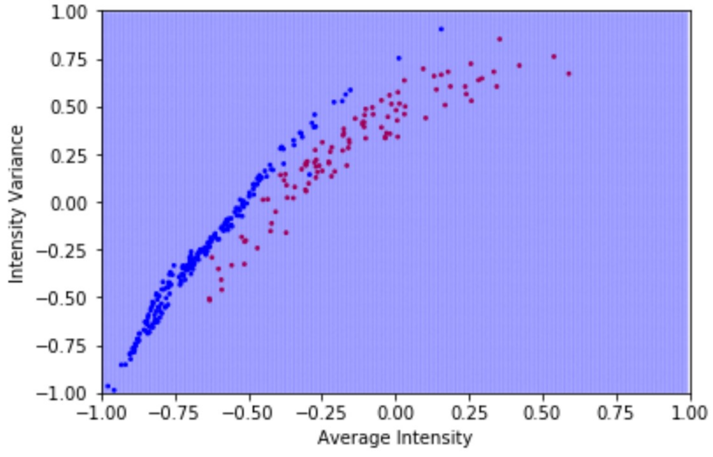
```
Score of the model is: 0.6698717948717948

C:\Users\Anu\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:432: F
utureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a solve
r to silence this warning.
  FutureWarning)
```

In [18]:
```
# create the model
# https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsCla
ssifier.html

# Declare Model
model = logistic_m3
# Fit model to our data
model.fit(simpleTrain,trainDigits)

# Lists to hold inpoints, predictions and assigned colors
xPred = []
yPred = []
cPred = []
# Use input points to get predictions here
for xP in range(-100,100):
    xP = xP/100.0
    for yP in range(-100,100):
        yP = yP/100.0
        xPred.append(xP)
        yPred.append(yP)
        if(model.predict([[xP,yP]])=="1.0"):
            cPred.append("b")
        else:
            cPred.append("r")
```

```
C:\Users\Anu\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:432: F
utureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a solve
r to silence this warning.
  FutureWarning)
```

In [19]:
```
## Visualize Results
#plot the points
fig = mp.figure()
mp.scatter(X,Y,s=3,c=colors)

#plot the regions
mp.scatter(xPred,yPred,s=3,c=cPred,alpha=.1)

#setup the axes
mp.xlim(-1,1)
mp.xlabel("Average Intensity")
mp.ylim(-1,1)
mp.ylabel("Intensity Variance")

#Label the figure
fig.suptitle('Figure 2.3.1', fontsize=15)
show()
```

Figure 2.3.1



Graduate Student question: L1 Regularization (c=2.0)

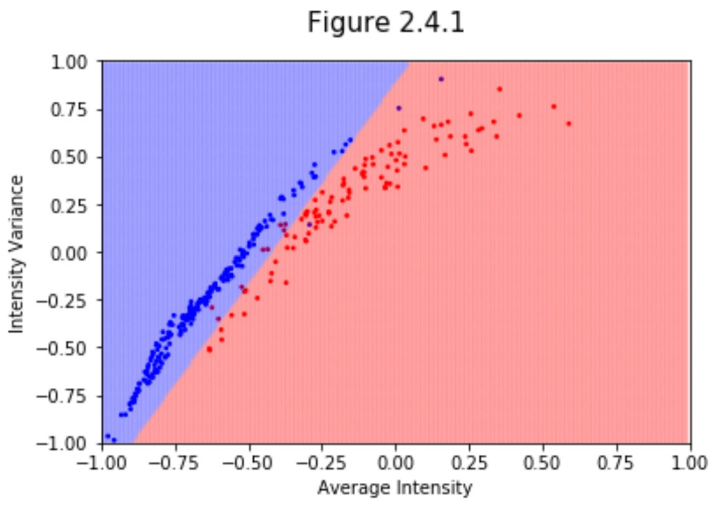
In [20]:
```
logistic_m4 = LogisticRegression(penalty='l1', C=2.0)
clsf = logistic_m4.fit(simpleTrain, trainDigits)
m_score = clsf.score(simpleTrain, trainDigits)
print("Score of the model is:", m_score)
```

```
Score of the model is: 0.9647435897435898

C:\Users\Anu\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:432: F
utureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a solve
r to silence this warning.
  FutureWarning)
```

In [21]:
```
# create the model
# https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsCla
ssifier.html

# Declare Model
model = logistic_m4
# Fit model to our data
model.fit(simpleTrain,trainDigits)

# Lists to hold inpoints, predictions and assigned colors
xPred = []
yPred = []
cPred = []
# Use input points to get predictions here
for xP in range(-100,100):
    xP = xP/100.0
    for yP in range(-100,100):
        yP = yP/100.0
        xPred.append(xP)
        yPred.append(yP)
        if(model.predict([[xP,yP]])=="1.0"):
            cPred.append("b")
        else:
            cPred.append("r")
```

```
C:\Users\Anu\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:432: F
utureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a solve
r to silence this warning.
  FutureWarning)
```

In [22]:
```
## Visualize Results
#plot the points
fig = mp.figure()
mp.scatter(X,Y,s=3,c=colors)

#plot the regions
mp.scatter(xPred,yPred,s=3,c=cPred,alpha=.1)

#setup the axes
mp.xlim(-1,1)
mp.xlabel("Average Intensity")
mp.ylim(-1,1)
mp.ylabel("Intensity Variance")

#Label the figure
fig.suptitle('Figure 2.4.1', fontsize=15)
show()
```



I think this regularization method can make it more likely to overfit the model. But, it depends on the value of C. In the same L1 Regularization for c=0.01, the model is underfitting than L2 regularization. Because L1 just gives either 0,1 for the parameter weights, for c=0.01 it might have given 0 for one of the features and that might have lead to underfitting. But whereas for c= 2.0, the L1 might have considred both the features with similar weights, the model likely seems to be overfitting. But it does a good job in seperating the data.
Yes, L1 vs L2 regularization has a huge impact here. I believe that c=2.0 (which means less penalty) with L2 regularization gives different coefficient values to intensity variance and average intensity and hence the decision regions are not classified so well. While in L1 regularization either the parameter is kept or removed (0 or 1). In this scenario, L1 regularization kept both the parameters while c=2.0 and since both have equal coefficients, the decision region was well classified. Similary while c=0.01 (which means more penalty) with L1 regularization, one of the parameters might have been removed which made the decision region all blue and lead to overfitting.

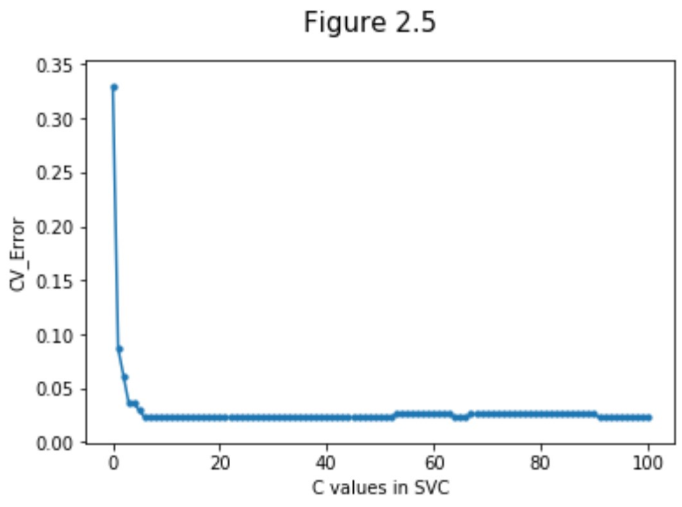
SUPPORT VECTOR MACHINES

3a) Linear soft margin svm

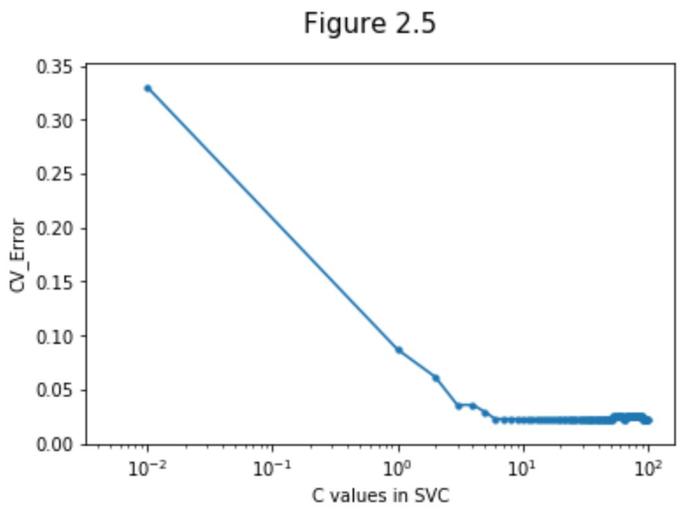
In [50]:
```
min_c = 0.01
max_c = 100
c=0
svc_models = []
cv_scores_list = []
c_values = []

while c < max_c:
    if (c == 0):
        c = min_c
        #print(c)
        svc_model = SVC(C=c, gamma='scale', kernel='linear')
        #print(svc_model)
        svc_models.append(svc_model)
        c = 0
    c = c+1
    #print(c)
    svc_model = SVC(C=c, gamma='scale', kernel='linear')
    #print(svc_model)
    svc_models.append(svc_model)

#train model with cv of 10
cv_errors = []
c_value = 0

for svcm in svc_models:
    cv_scores = cross_val_score(svcm, simpleTrain, trainDigits, cv=10)
    cv_scores_list.append(cv_scores)

#print each cv score (accuracy) and average them
for i in range(len(svc_models)):
    if (i == 0):
        c_value = min_c
        c_values.append(c_value)
        print('c:', c_value)
        print(svc_models[i])
        print(cv_scores_list[i])
        print('cv_scores mean:{}'.format(np.mean(cv_scores_list[i])))
        error = 1-np.mean(cv_scores_list[i])
        cv_errors.append(error)
        print('Cross validation error:', error)
        c_value = 0
    else:
        c_value = i
        c_values.append(c_value)
        print('c:', c_value)
        print(svc_models[i])
        print(cv_scores_list[i])
        print('cv_scores mean:{}'.format(np.mean(cv_scores_list[i])))
        error = 1-np.mean(cv_scores_list[i])
        cv_errors.append(error)
        print('Cross validation error:', error)
```

```
c: 0.01
SVC(C=0.01, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='linear',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
[0.65625    0.65625    0.65625    0.67741935 0.67741935 0.67741935
 0.67741935 0.67741935 0.67741935 0.66666667]
cv_scores mean:0.6699932795698925
Cross validation error: 0.3300067204301075
c: 1
SVC(C=1, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='linear',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
[0.9375     0.90625    0.96875    1.         0.83870968 0.87096774
 0.93548387 0.93548387 0.87096774 0.86666667]
cv_scores mean:0.9130779569892473
Cross validation error: 0.08692204301075268
c: 2
SVC(C=2, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='linear',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
[1.         0.96875    0.96875    1.         0.90322581 0.87096774
 0.93548387 0.93548387 0.93548387 0.86666667]
cv_scores mean:0.9384811827956989
Cross validation error: 0.06151881720430108
c: 3
SVC(C=3, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='linear',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
[1.         1.         0.96875    1.         0.93548387 0.90322581
 0.93548387 1.         0.96774194 0.93333333]
cv_scores mean:0.9644018817204302
Cross validation error: 0.035598118279569824
c: 4
SVC(C=4, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='linear',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
[1.         1.         0.96875    1.         0.90322581 0.90322581
 0.96774194 1.         0.96774194 0.93333333]
cv_scores mean:0.9644018817204302
Cross validation error: 0.035598118279569824
c: 5
SVC(C=5, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='linear',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
[1.         1.         0.96875    1.         0.90322581 0.90322581
 1.         1.         0.96774194 0.96666667]
cv_scores mean:0.9709610215053763
Cross validation error: 0.02903897849462367
c: 6
SVC(C=6, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='linear',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
[1.         1.         1.         1.         0.90322581 0.93548387
 1.         1.         0.96774194 0.96666667]
cv_scores mean:0.9773118279569892
Cross validation error: 0.022688172043010768
c: 7
```

In [51]:
```
#plot the data points
### https://matplotlib.org/api/_as_gen/matplotlib.pyplot.scatter.html
fig = mp.figure()
mp.scatter(c_values,cv_errors,s=10)
mp.plot(c_values,cv_errors)

#specify the axes
mp.xlabel("C values in SVC")
mp.ylabel("CV_Error")

#Labeling the plot
#mp.legend(['1'])
#mp.legend(legends)
fig.suptitle('Figure 2.5', fontsize=15)

#display the current graph
show()
```



In [52]:
```
#plot the data points log scale
### https://matplotlib.org/api/_as_gen/matplotlib.pyplot.scatter.html
fig = mp.figure()
mp.scatter(c_values,cv_errors,s=10)
mp.xscale('log')
mp.plot(c_values,cv_errors)

#specify the axes
mp.xlabel("C values in SVC")
mp.ylabel("CV_Error")

#Labeling the plot
#mp.legend(['1'])
#mp.legend(legends)
fig.suptitle('Figure 2.5', fontsize=15)

#display the current graph
show()
```

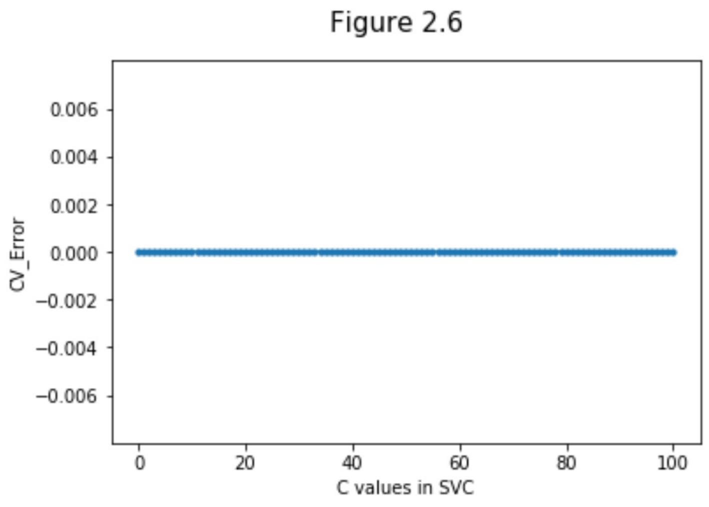


3b) 256 Dimensions - SVC

In [26]:
```
from sklearn.svm import SVC
min_c = 0.01
max_c = 100
c=0
svc_models = []
cv_scores_list = []
c_values = []

while c < max_c:
    if (c == 0):
        c = min_c
        #print(c)
        svc_model = SVC(C=c, gamma='scale', kernel='linear')
        #print(svc_model)
        svc_models.append(svc_model)
        c = 0
    c = c+1
    #print(c)
    svc_model = SVC(C=c, gamma='scale', kernel='linear')
    #print(svc_model)
    svc_models.append(svc_model)

#train model with cv of 10
cv_errors = []
c_value = 0

for svcm in svc_models:
    cv_scores = cross_val_score(svcm, trainFeatures, trainDigits, cv=10)
    cv_scores_list.append(cv_scores)

#print each cv score (accuracy) and average them
for i in range(len(svc_models)):
    if (i == 0):
        c_value = min_c
        c_values.append(c_value)
        print('c:', c_value)
        print(svc_models[i])
        print(cv_scores_list[i])
        print('cv_scores mean:{}'.format(np.mean(cv_scores_list[i])))
        error = 1-np.mean(cv_scores_list[i])
        cv_errors.append(error)
        print('Cross validation error:', error)
        c_value = 0
    else:
        c_value = i
        c_values.append(c_value)
        print('c:', c_value)
        print(svc_models[i])
        print(cv_scores_list[i])
        print('cv_scores mean:{}'.format(np.mean(cv_scores_list[i])))
        error = 1-np.mean(cv_scores_list[i])
        cv_errors.append(error)
        print('Cross validation error:', error)
```

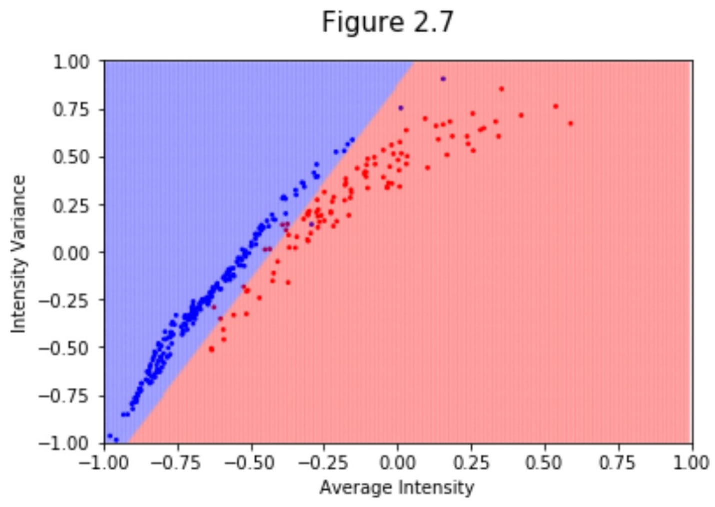
```
c: 0.01
SVC(C=0.01, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='linear',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
[1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
cv_scores mean:1.0
Cross validation error: 0.0
c: 1
SVC(C=1, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='linear',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
[1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
cv_scores mean:1.0
Cross validation error: 0.0
c: 2
SVC(C=2, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='linear',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
[1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
cv_scores mean:1.0
Cross validation error: 0.0
c: 3
SVC(C=3, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='linear',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
[1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
cv_scores mean:1.0
Cross validation error: 0.0
c: 4
SVC(C=4, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='linear',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
[1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
cv_scores mean:1.0
Cross validation error: 0.0
c: 5
SVC(C=5, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='linear',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
[1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
cv_scores mean:1.0
Cross validation error: 0.0
c: 6
SVC(C=6, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='linear',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
[1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
cv_scores mean:1.0
Cross validation error: 0.0
c: 7
SVC(C=7, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='linear',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
[1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
cv_scores mean:1.0
Cross validation error: 0.0
```

In [27]:
```
#plot the data points
### https://matplotlib.org/api/_as_gen/matplotlib.pyplot.scatter.html
fig = mp.figure()
mp.scatter(c_values,cv_errors,s=10)
mp.plot(c_values,cv_errors)

#specify the axes
mp.xlabel("C values in SVC")
mp.ylabel("CV_Error")

#Labeling the plot
#mp.legend(['1'])
#mp.legend(legends)
fig.suptitle('Figure 2.6', fontsize=15)

#display the current graph
show()
```



In [28]:
```
# create the model
# https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsCla
ssifier.html

# Declare Model
model = SVC(C=6, gamma='scale', kernel='linear')
# Fit model to our data
model.fit(simpleTrain,trainDigits)

# Lists to hold inpoints, predictions and assigned colors
xPred = []
yPred = []
cPred = []
# Use input points to get predictions here
for xP in range(-100,100):
    xP = xP/100.0
    for yP in range(-100,100):
        yP = yP/100.0
        xPred.append(xP)
        yPred.append(yP)
        if(model.predict([[xP,yP]])=="1.0"):
            cPred.append("b")
        else:
            cPred.append("r")
```

In [29]:
```
## Visualize Results
#plot the points
fig = mp.figure()
mp.scatter(X,Y,s=3,c=colors)

#plot the regions
mp.scatter(xPred,yPred,s=3,c=cPred,alpha=.1)

#setup the axes
mp.xlim(-1,1)
mp.xlabel("Average Intensity")
mp.ylim(-1,1)
mp.ylabel("Intensity Variance")

#Label the figure
fig.suptitle('Figure 2.7', fontsize=15)
show()
```



In this I have choosen value of C as 6 even though C=6, C=10, C=100 have same cross validation errors. Because, choosing high value of C might lead to overfitting and and choosing low value of C can cause underfitting (like c=0.01). Even though the cross valodation errors are same, to be on a better bias variance tradeoff level, I choose C=6 which is not to high and not to low and gives the least cross validation error too.

3c) Polynomial Kernel

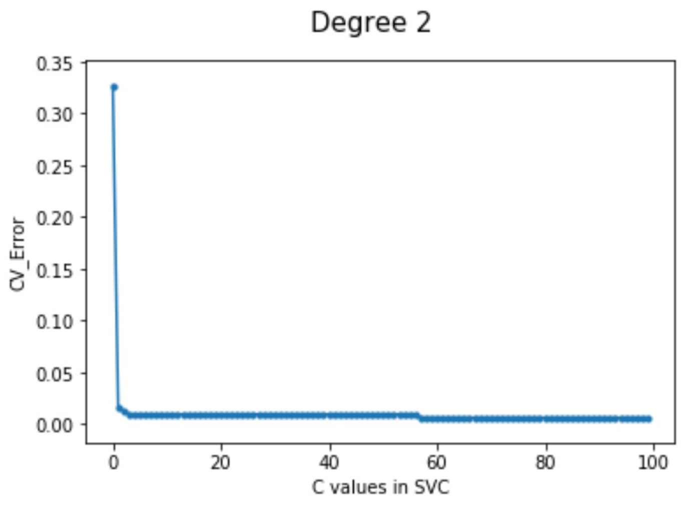
In [77]:
```
#Polynomial Kernel degree 2
c_vals = []
err_vals = []
svc_model_d2 = SVC(C=0.01, gamma="scale", kernel='poly', degree=2)
c_vals.append(0.01)
cv_score = cross_val_score(svc_model_d2, simpleTrain, trainDigits, cv=10)
print('cv_scores mean:{}'.format(np.mean(cv_score)))
error = 1-np.mean(cv_score)
print('Cross validation error:', error)
err_vals.append(error)
```

```
cv_scores mean:0.6731182795698925
Cross validation error: 0.32688172043010755
```

In [78]:
```
#Polynomial Kernel degree 2 (smallest cross validatio error)
model_min_error = 1
model_cv_score = 0
model_c_value = 0
for i in range(1,100):
    svc_model_d2 = SVC(C=i, gamma="scale", kernel='poly', degree=2)
    c_vals.append(i)
    cv_score = cross_val_score(svc_model_d2, simpleTrain, trainDigits, cv=10)
    cv_score = np.mean(cv_score)
    error = 1-np.mean(cv_score)
    err_vals.append(error)
    #print('Cross validation error:', error)
    if (error < model_min_error):
        model_min_error = error
        model_cv_score = cv_score
        model_c_value = i
print("Optimum cv_score:", model_cv_score)
print("Minimum cv_error:", model_min_error)
print("Value of c:", model_c_value)
```

```
Optimum cv_score: 0.9936491935483872
Minimum cv_error: 0.006350806451612789
Value of c: 57
```

In [79]:
```
#plot the data points
### https://matplotlib.org/api/_as_gen/matplotlib.pyplot.scatter.html
fig = mp.figure()
mp.scatter(c_vals,err_vals,s=10)
mp.plot(c_vals,err_vals)

#specify the axes
mp.xlabel("C values in SVC")
mp.ylabel("CV_Error")

#Labeling the plot
#mp.legend(['1'])
#mp.legend(legends)
fig.suptitle('Degree 2', fontsize=15)

#display the current graph
show()
```



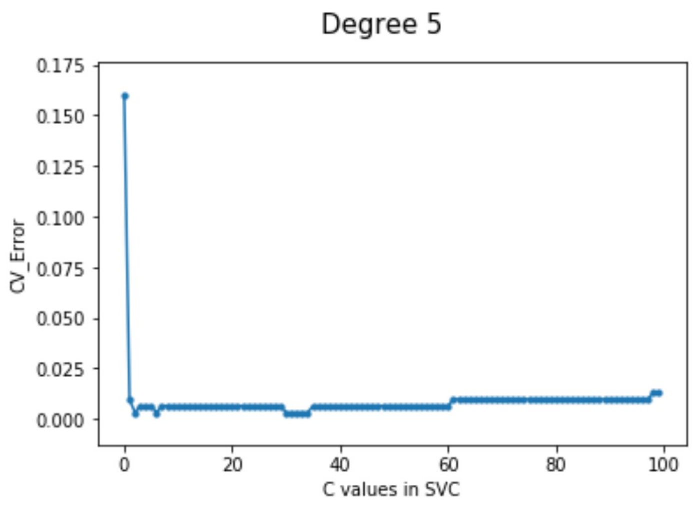
In [80]:
```
#Polynomial Kernel degree 5
c_vals = []
err_vals = []
svc_model_d5 = SVC(C=0.01, gamma="scale", kernel='poly', degree=5)
c_vals.append(0.01)
cv_score = cross_val_score(svc_model_d5, simpleTrain, trainDigits, cv=10)
print('cv_scores mean:{}'.format(np.mean(cv_score)))
error = 1-np.mean(cv_score)
print('Cross validation error:', error)
err_vals.append(error)
```

```
cv_scores mean:0.8396841397849464
Cross validation error: 0.16031586021505362
```

In [81]:
```
#Polynomial Kernel degree 5 (smallest cross validatio error)
model_min_error = 1
model_cv_score = 0
model_c_value = 0
for i in range(1,100):
    svc_model_d5 = SVC(C=i, gamma="scale", kernel='poly', degree=5)
    c_vals.append(i)
    cv_score = cross_val_score(svc_model_d5, simpleTrain, trainDigits, cv=10)
    cv_score = np.mean(cv_score)
    error = 1-np.mean(cv_score)
    err_vals.append(error)
    #print('Cross validation error:', error)
    if (error < model_min_error):
        model_min_error = error
        model_cv_score = cv_score
        model_c_value = i
print("Optimum cv_score:", model_cv_score)
print("Minimum cv_error:", model_min_error)
print("Value of c:", model_c_value)
```

```
Optimum cv_score: 0.9967741935483871
Minimum cv_error: 0.003225806451612856
Value of c: 2
```

In [82]:
```
#plot the data points
### https://matplotlib.org/api/_as_gen/matplotlib.pyplot.scatter.html
fig = mp.figure()
mp.scatter(c_vals,err_vals,s=10)
mp.plot(c_vals,err_vals)

#specify the axes
mp.xlabel("C values in SVC")
mp.ylabel("CV_Error")

#Labeling the plot
#mp.legend(['1'])
#mp.legend(legends)
fig.suptitle('Degree 5', fontsize=15)

#display the current graph
show()
```



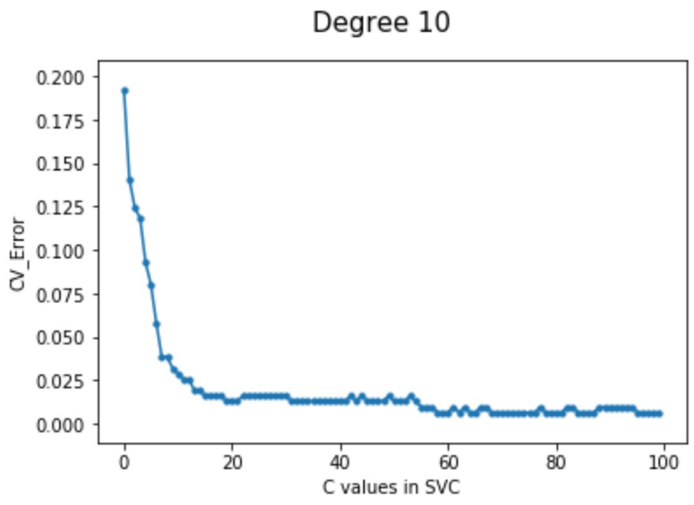
In [53]:
```
#Polynomial Kernel degree 10
c_vals = []
err_vals = []
svc_model_d10 = SVC(C=0.01, gamma="scale", kernel='poly', degree=10)
c_vals.append(0.01)
cv_score = cross_val_score(svc_model_d10, simpleTrain, trainDigits, cv=10)
print('cv_scores mean:{}'.format(np.mean(cv_score)))
error = 1-np.mean(cv_score)
print('Cross validation error:', error)
err_vals.append(error)
```

```
cv_scores mean:0.8081451612903227
Cross validation error: 0.19185483870967734
```

In [54]:
```
#Polynomial Kernel degree 10 (smallest cross validation error)
model_min_error = 1
model_cv_score = 0
model_c_value = 0
for i in range(1,100):
    svc_model_d10 = SVC(C=i, gamma="scale", kernel='poly', degree=10)
    c_vals.append(i)
    cv_score = cross_val_score(svc_model_d10, simpleTrain, trainDigits, cv=10)
    cv_score = np.mean(cv_score)
    error = 1-np.mean(cv_score)
    err_vals.append(error)
    #print('Cross validation error:', error)
    if (error < model_min_error):
        model_min_error = error
        model_cv_score = cv_score
        model_c_value = i
print("Optimum cv_score:", model_cv_score)
print("Minimum cv_error:", model_min_error)
print("Value of c:", model_c_value)
```

```
Optimum cv_score: 0.9936491935483872
Minimum cv_error: 0.006350806451612789
Value of c: 58
```

In [55]:
```
#plot the data points
### https://matplotlib.org/api/_as_gen/matplotlib.pyplot.scatter.html
fig = mp.figure()
mp.scatter(c_vals,err_vals,s=10)
mp.plot(c_vals,err_vals)

#specify the axes
mp.xlabel("C values in SVC")
mp.ylabel("CV_Error")

#Labeling the plot
#mp.legend(['1'])
#mp.legend(legends)
fig.suptitle('Degree 10', fontsize=15)

#display the current graph
show()
```



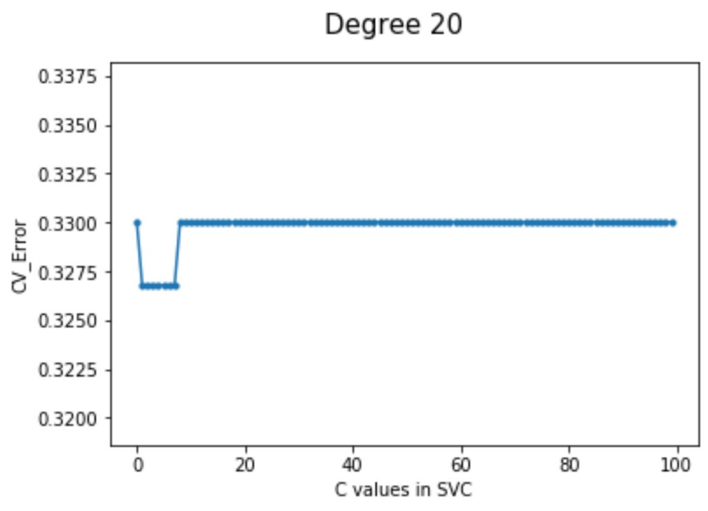
In [86]:
```
#Polynomial Kernel degree 20
c_vals = []
err_vals = []
svc_model_d20 = SVC(C=0.01, gamma=1, kernel='poly', degree=20)
c_vals.append(0.01)
cv_score = cross_val_score(svc_model_d20, simpleTrain, trainDigits, cv=10)
print('cv_scores mean:{}'.format(np.mean(cv_score)))
error = 1-np.mean(cv_score)
print('Cross validation error:', error)
err_vals.append(error)
```

```
cv_scores mean:0.6699932795698925
Cross validation error: 0.3300067204301075
```

In [87]:
```
#Polynomial Kernel degree 20 (smallest cross validatio error)
model_min_error = 1
model_cv_score = 0
model_c_value = 0
for i in range(1,100):
    svc_model_d20 = SVC(C=i, gamma=1, kernel='poly', degree=20)
    c_vals.append(i)
    cv_score = cross_val_score(svc_model_d20, simpleTrain, trainDigits, cv=10)
    cv_score = np.mean(cv_score)
    error = 1-np.mean(cv_score)
    err_vals.append(error)
    #print('Cross validation error:', error)
    if (error < model_min_error):
        model_min_error = error
        model_cv_score = cv_score
        model_c_value = i
print("Optimum cv_score:", model_cv_score)
print("Minimum cv_error:", model_min_error)
print("Value of c:", model_c_value)
```

```
Optimum cv_score: 0.6732190860215055
Minimum cv_error: 0.3267809139784945
Value of c: 1
```

```
In [76]: #plot the data points
         ### https://matplotlib.org/api/_as_gen/matplotlib.pyplot.scatter.html
         fig = mp.figure()
         mp.scatter(c_vals,err_vals,s=10)
         mp.plot(c_vals,err_vals)

         #specify the axes
         mp.xlabel("C values in SVC")
         mp.ylabel("CV_Error")

         #Labeling the plot
         #mp.legend(['1'])
         #mp.legend(legends)
         fig.suptitle('Degree 20', fontsize=15)

         #display the current graph
         show()
```

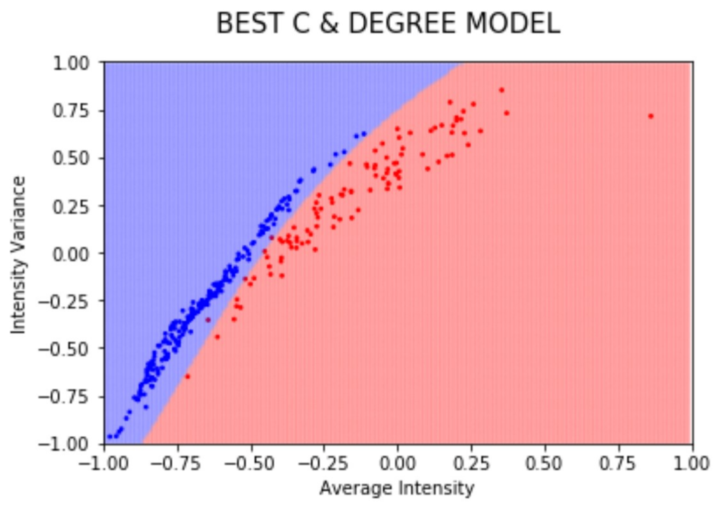


Tradeoff between degree and c: Very high values of C or high degree can cause overfitting. Similarly very low C values can lead to underfitting and low degree can also lead to underfitting in some cases where the data is not in a linear fashion. Here, I see that with low degree eg: degree=2 & c=57 gives an optimal model with lowest cross validation error. As, we increase degree to 5, we were able to achieve optimal model at C=2 with lowest cross validation error. And with degree=10 & 20, we were able to achieve optimal model at C=1. So, as degree increases we can decrease c value and as degree decreases, we need to increase c value to achieve an optimal model. Note: Since gamma="scale" is taking long time to run, I have changed it to 1 for degree 20.

3d) Best value of c & degree

In [4]:
```python
# create the model
# https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsCla
ssifier.html

# Declare Model
model = SVC(C=1, gamma='scale', kernel='poly', degree=3)
# Fit model to our data
model.fit(simpleTrain,trainDigits)

# Lists to hold inpoints, predictions and assigned colors
xPred = []
yPred = []
cPred = []
# Use input points to get predictions here
for xP in range(-100,100):
    xP = xP/100.0
    for yP in range(-100,100):
        yP = yP/100.0
        xPred.append(xP)
        yPred.append(yP)
        if(model.predict([[xP,yP]])=="1.0"):
            cPred.append("b")
        else:
            cPred.append("r")
```

In [5]:
```python
## Visualize Results
#plot the points
fig = mp.figure()
mp.scatter(X,Y,s=3,c=colors)

#plot the regions
mp.scatter(xPred,yPred,s=3,c=cPred,alpha=.1)

#setup the axes
mp.xlim(-1,1)
mp.xlabel("Average Intensity")
mp.ylim(-1,1)
mp.ylabel("Intensity Variance")

#Label the figure
fig.suptitle('BEST C & DEGREE MODEL', fontsize=15)
show()
```



```
In [6]: model = SVC(C=1, gamma='scale', kernel='poly', degree=3)
        cv_scores = cross_val_score(model, simpleTrain, trainDigits, cv=10)
        print(cv_scores)
```

```
[1.         1.         1.         1.         0.96774194 0.93548387
 1.         1.         0.96774194 0.96666667]
```

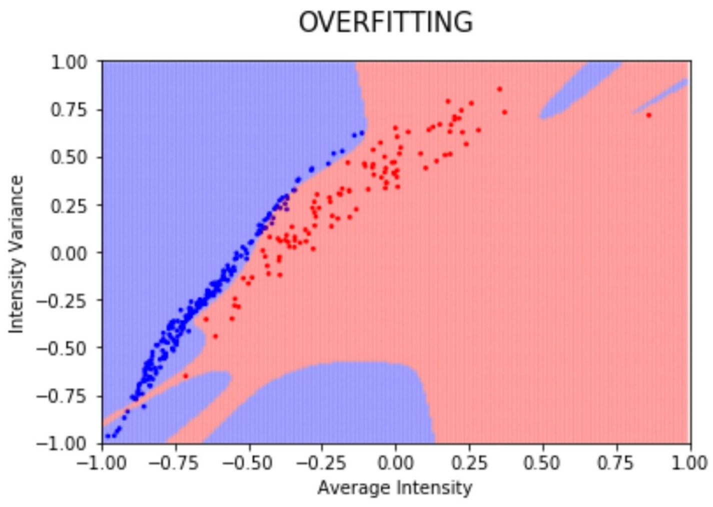
I think the best values are degree=3 and c=1. Here in the decision region we can see that the degree 3 doesnot overfit the data and with c=1 there are not too many misclassifications too. Also, from the cross validation score we can see that the errors are constant and not varing too much. Hence, I considered degree=3 and c=1.

3) Graduate Student question

OVERFITTING MODEL

```
In [44]: # create the model
         # https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsCla
         ssifier.html

         # Declare Model
         model = SVC(C=100, gamma='scale', kernel='poly', degree=15)
         # Fit model to our data
         model.fit(simpleTrain,trainDigits)

         # Lists to hold inpoints, predictions and assigned colors
         xPred = []
         yPred = []
         cPred = []
         # Use input points to get predictions here
         for xP in range(-100,100):
             xP = xP/100.0
             for yP in range(-100,100):
                 yP = yP/100.0
                 xPred.append(xP)
                 yPred.append(yP)
                 if(model.predict([[xP,yP]])=="1.0"):
                     cPred.append("b")
                 else:
                     cPred.append("r")
```

In [45]:
```
## Visualize Results
#plot the points
fig = mp.figure()
mp.scatter(X,Y,s=3,c=colors)

#plot the regions
mp.scatter(xPred,yPred,s=3,c=cPred,alpha=.1)

#setup the axes
mp.xlim(-1,1)
mp.xlabel("Average Intensity")
mp.ylim(-1,1)
mp.ylabel("Intensity Variance")

#Label the figure
fig.suptitle('OVERFITTING', fontsize=15)
show()
```



In [46]:
```
model = SVC(C=100, gamma='scale', kernel='poly', degree=15)
cv_scores = cross_val_score(model, simpleTrain, trainDigits, cv=10)
print(cv_scores)
```

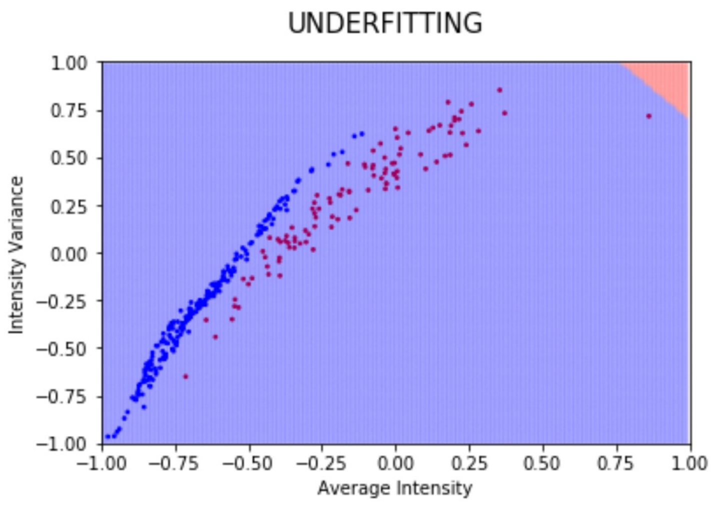
```
[0.96875    0.96875    0.9375     0.96774194 0.93548387 0.87096774
 1.         0.96774194 0.90322581 0.96666667]
```

Overfitting:I have choosen a 'polynomial' kernel with degree 15 and c=100 to be the overfitted model. We can increase the degree more to get a more overfitted model but as it was taking huge time to run, I took degree as 20. We can see from the graph is very curvy and susceptible to misclassifications. Also, the cross validation errors are very low and varying a bit(highest is 1 and lowest is 0.87) which explains that the model is more susceptible to misclassifications.

UNDERFITTING

In [4]:
```
# create the model
# https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsCla
ssifier.html

# Declare Model
model = SVC(C=0.01, gamma='scale', kernel='linear')
# Fit model to our data
model.fit(simpleTrain,trainDigits)

# Lists to hold inpoints, predictions and assigned colors
xPred = []
yPred = []
cPred = []
# Use input points to get predictions here
for xP in range(-100,100):
    xP = xP/100.0
    for yP in range(-100,100):
        yP = yP/100.0
        xPred.append(xP)
        yPred.append(yP)
        if(model.predict([[xP,yP]])=="1.0"):
            cPred.append("b")
        else:
            cPred.append("r")
```

In [5]:
```
## Visualize Results
#plot the points
fig = mp.figure()
mp.scatter(X,Y,s=3,c=colors)

#plot the regions
mp.scatter(xPred,yPred,s=3,c=cPred,alpha=.1)

#setup the axes
mp.xlim(-1,1)
mp.xlabel("Average Intensity")
mp.ylim(-1,1)
mp.ylabel("Intensity Variance")

#Label the figure
fig.suptitle('UNDERFITTING', fontsize=15)
show()
```



```
In [6]: model = SVC(C=0.01, gamma='scale', kernel='linear')
        cv_scores = cross_val_score(model, simpleTrain, trainDigits, cv=10)
        print(cv_scores)
```

```
[0.65625    0.65625    0.65625    0.67741935 0.67741935 0.67741935
 0.67741935 0.67741935 0.67741935 0.66666667]
```

Underfitting: I have choosen 'linear' kernel with c=0.01 for an underfitting model. This model is underfitting and assumes almost everything as a blue point except for a very small region calssifies as red points. Even though the cross validation score averages upto 60% and I believe that is because blue points are containing 60% of the data paoints and the remaining 40% are red points. And for any underfitted model, the worst cross validation error can be 60% when whole region is classified as blue.

EXTRA CREDIT

Kernel = rbf, gamma = scale, Degree = 2, different c values

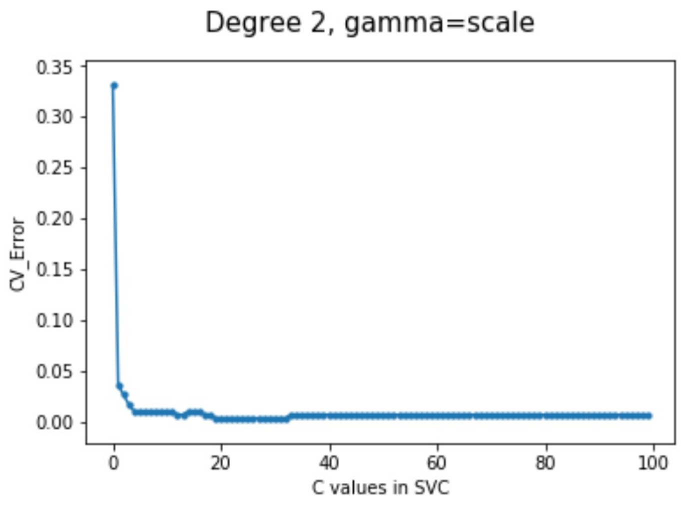
```
In [26]: #RBF Kernel degree 2 scale
         c_vals = []
         err_vals = []
         svc_model_d20 = SVC(C=0.01, gamma="scale", kernel='rbf', degree=2)
         c_vals.append(0.01)
         cv_score = cross_val_score(svc_model_d20, simpleTrain, trainDigits, cv=10)
         print('cv_scores mean:{}'.format(np.mean(cv_score)))
         error = 1-np.mean(cv_score)
         print('Cross validation error:', error)
         err_vals.append(error)
```

```
cv_scores mean:0.6699932795698925
Cross validation error: 0.3300067204301075
```

```
In [27]: model_min_error = 1
         model_cv_score = 0
         model_c_value = 0
         for i in range(1,100):
             svc_model_d5 = SVC(C=i, gamma="scale", kernel='rbf', degree=2)
             c_vals.append(i)
             cv_score = cross_val_score(svc_model_d5, simpleTrain, trainDigits, cv=10)
             cv_score = np.mean(cv_score)
             error = 1-np.mean(cv_score)
             err_vals.append(error)
             #print('Cross validation error:', error)
             if (error < model_min_error):
                 model_min_error = error
                 model_cv_score = cv_score
                 model_c_value = i
         print("Optimum cv_score:", model_cv_score)
         print("Minimum cv_error:", model_min_error)
         print("Value of c:", model_c_value)
```

```
Optimum cv_score: 0.9967741935483871
Minimum cv_error: 0.003225806451612856
Value of c: 19
```

In [28]:
```
#plot the data points
### https://matplotlib.org/api/_as_gen/matplotlib.pyplot.scatter.html
fig = mp.figure()
mp.scatter(c_vals,err_vals,s=10)
mp.plot(c_vals,err_vals)

#specify the axes
mp.xlabel("C values in SVC")
mp.ylabel("CV_Error")

#Labeling the plot
#mp.legend(['1'])
#mp.legend(legends)
fig.suptitle('Degree 2, gamma=scale', fontsize=15)

#display the current graph
show()
```



Kernel = rbf, gamma = scale, Degree = 5, different c values

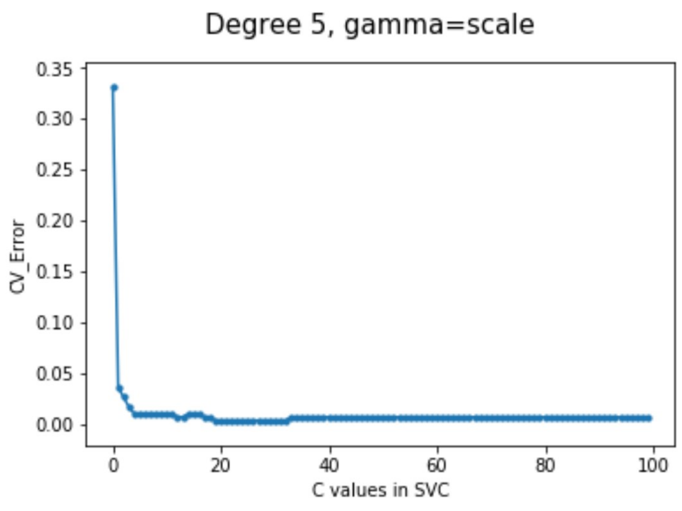
In [29]:
```
#RBF Kernel degree 5 scale
c_vals = []
err_vals = []
svc_model_d20 = SVC(C=0.01, gamma="scale", kernel='rbf', degree=5)
c_vals.append(0.01)
cv_score = cross_val_score(svc_model_d20, simpleTrain, trainDigits, cv=10)
print('cv_scores mean:{}'.format(np.mean(cv_score)))
error = 1-np.mean(cv_score)
print('Cross validation error:', error)
err_vals.append(error)
```

```
cv_scores mean:0.6699932795698925
Cross validation error: 0.3300067204301075
```

In [30]:
```
model_min_error = 1
model_cv_score = 0
model_c_value = 0
for i in range(1,100):
    svc_model_d5 = SVC(C=i, gamma="scale", kernel='rbf', degree=5)
    c_vals.append(i)
    cv_score = cross_val_score(svc_model_d5, simpleTrain, trainDigits, cv=10)
    cv_score = np.mean(cv_score)
    error = 1-np.mean(cv_score)
    err_vals.append(error)
    #print('Cross validation error:', error)
    if (error < model_min_error):
        model_min_error = error
        model_cv_score = cv_score
        model_c_value = i
print("Optimum cv_score:", model_cv_score)
print("Minimum cv_error:", model_min_error)
print("Value of c:", model_c_value)
```

```
Optimum cv_score: 0.9967741935483871
Minimum cv_error: 0.003225806451612856
Value of c: 19
```

In [31]:
```
#plot the data points
### https://matplotlib.org/api/_as_gen/matplotlib.pyplot.scatter.html
fig = mp.figure()
mp.scatter(c_vals,err_vals,s=10)
mp.plot(c_vals,err_vals)

#specify the axes
mp.xlabel("C values in SVC")
mp.ylabel("CV_Error")

#Labeling the plot
#mp.legend(['1'])
#mp.legend(legends)
fig.suptitle('Degree 5, gamma=scale', fontsize=15)

#display the current graph
show()
```



Kernel = rbf, gamma = auto, Degree = 2, different c values

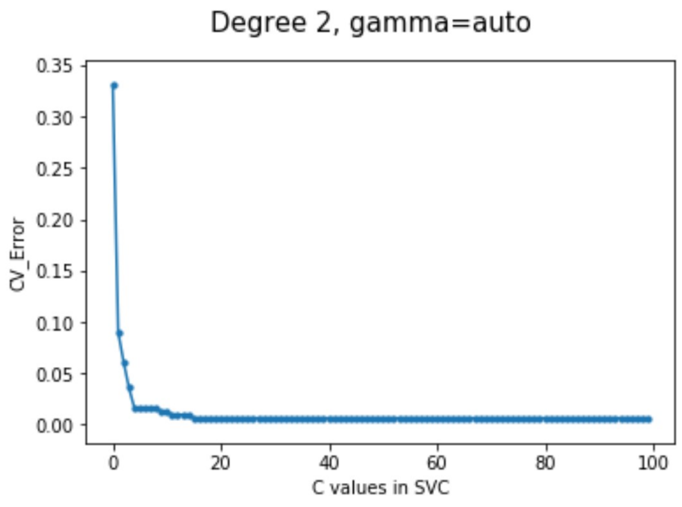
In [32]:
```
#RBF Kernel degree 2 auto
c_vals = []
err_vals = []
svc_model_d20 = SVC(C=0.01, gamma="auto", kernel='rbf', degree=2)
c_vals.append(0.01)
cv_score = cross_val_score(svc_model_d20, simpleTrain, trainDigits, cv=10)
print('cv_scores mean:{}'.format(np.mean(cv_score)))
error = 1-np.mean(cv_score)
print('Cross validation error:', error)
err_vals.append(error)
```

```
cv_scores mean:0.6699932795698925
Cross validation error: 0.3300067204301075
```

In [33]:
```
#Polynomial Kernel degree 5 (smallest cross validatio error)
model_min_error = 1
model_cv_score = 0
model_c_value = 0
for i in range(1,100):
    svc_model_d5 = SVC(C=i, gamma="auto", kernel='rbf', degree=2)
    c_vals.append(i)
    cv_score = cross_val_score(svc_model_d5, simpleTrain, trainDigits, cv=10)
    cv_score = np.mean(cv_score)
    error = 1-np.mean(cv_score)
    err_vals.append(error)
    #print('Cross validation error:', error)
    if (error < model_min_error):
        model_min_error = error
        model_cv_score = cv_score
        model_c_value = i
print("Optimum cv_score:", model_cv_score)
print("Minimum cv_error:", model_min_error)
print("Value of c:", model_c_value)
```

```
Optimum cv_score: 0.9935483870967742
Minimum cv_error: 0.006451612903225823
Value of c: 15
```

```
In [34]: #plot the data points
         ### https://matplotlib.org/api/_as_gen/matplotlib.pyplot.scatter.html
         fig = mp.figure()
         mp.scatter(c_vals,err_vals,s=10)
         mp.plot(c_vals,err_vals)

         #specify the axes
         mp.xlabel("C values in SVC")
         mp.ylabel("CV_Error")

         #Labeling the plot
         #mp.legend(['1'])
         #mp.legend(legends)
         fig.suptitle('Degree 2, gamma=auto', fontsize=15)

         #display the current graph
         show()
```



Kernel = rbf, gamma = auto, Degree = 5, different c values

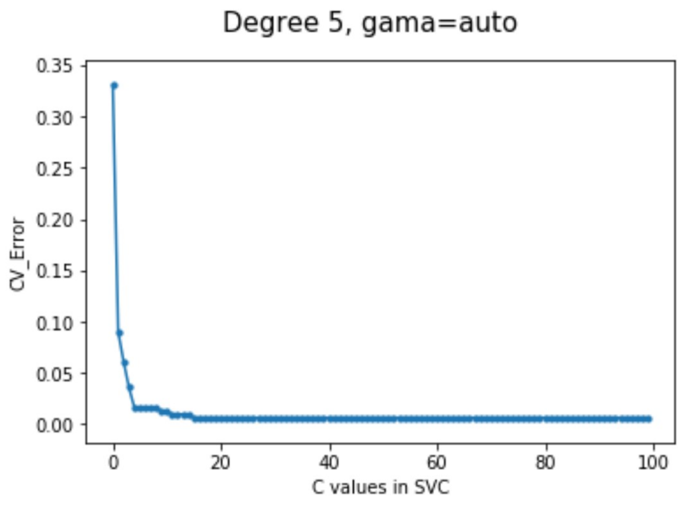
```
In [35]: #RBF Kernel degree 5 auto
         c_vals = []
         err_vals = []
         svc_model_d20 = SVC(C=0.01, gamma="auto", kernel='rbf', degree=5)
         c_vals.append(0.01)
         cv_score = cross_val_score(svc_model_d20, simpleTrain, trainDigits, cv=10)
         print('cv_scores mean:{}'.format(np.mean(cv_score)))
         error = 1-np.mean(cv_score)
         print('Cross validation error:', error)
         err_vals.append(error)
```

```
cv_scores mean:0.6699932795698925
Cross validation error: 0.3300067204301075
```

In [36]:
```python
#Polynomial Kernel degree 5 (smallest cross validatio error)
model_min_error = 1
model_cv_score = 0
model_c_value = 0
for i in range(1,100):
    svc_model_d5 = SVC(C=i, gamma="auto", kernel='rbf', degree=5)
    c_vals.append(i)
    cv_score = cross_val_score(svc_model_d5, simpleTrain, trainDigits, cv=10)
    cv_score = np.mean(cv_score)
    error = 1-np.mean(cv_score)
    err_vals.append(error)
    #print('Cross validation error:', error)
    if (error < model_min_error):
        model_min_error = error
        model_cv_score = cv_score
        model_c_value = i
print("Optimum cv_score:", model_cv_score)
print("Minimum cv_error:", model_min_error)
print("Value of c:", model_c_value)
```

```
Optimum cv_score: 0.9935483870967742
Minimum cv_error: 0.006451612903225823
Value of c: 15
```

In [37]:
```python
#plot the data points
### https://matplotlib.org/api/_as_gen/matplotlib.pyplot.scatter.html
fig = mp.figure()
mp.scatter(c_vals,err_vals,s=10)
mp.plot(c_vals,err_vals)

#specify the axes
mp.xlabel("C values in SVC")
mp.ylabel("CV_Error")

#Labeling the plot
#mp.legend(['1'])
#mp.legend(legends)
fig.suptitle('Degree 5, gama=auto', fontsize=15)

#display the current graph
show()
```



Kernel = rbf, gamma = 1, Degree = 2, different c values

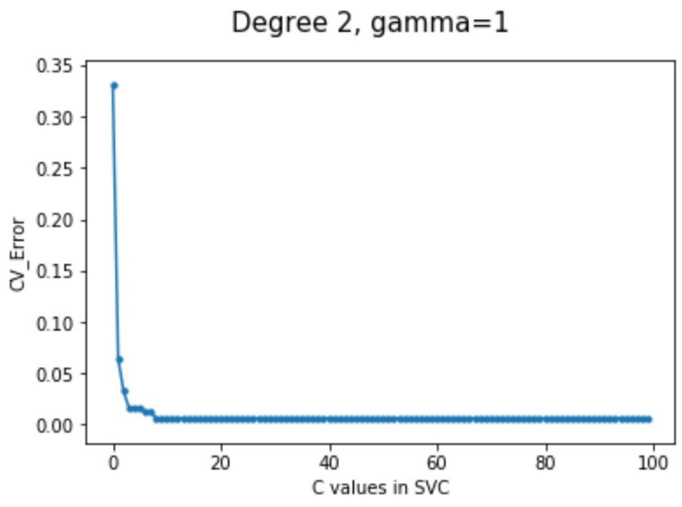
In [38]:
```
#RBF Kernel degree 2 gamma=1
c_vals = []
err_vals = []
svc_model_d20 = SVC(C=0.01, gamma=1, kernel='rbf', degree=2)
c_vals.append(0.01)
cv_score = cross_val_score(svc_model_d20, simpleTrain, trainDigits, cv=10)
print('cv_scores mean:{}'.format(np.mean(cv_score)))
error = 1-np.mean(cv_score)
print('Cross validation error:', error)
err_vals.append(error)
```

```
cv_scores mean:0.6699932795698925
Cross validation error: 0.3300067204301075
```

In [39]:
```
#Polynomial Kernel degree 5 (smallest cross validatio error)
model_min_error = 1
model_cv_score = 0
model_c_value = 0
for i in range(1,100):
    svc_model_d5 = SVC(C=i, gamma=1, kernel='rbf', degree=2)
    c_vals.append(i)
    cv_score = cross_val_score(svc_model_d5, simpleTrain, trainDigits, cv=10)
    cv_score = np.mean(cv_score)
    error = 1-np.mean(cv_score)
    err_vals.append(error)
    #print('Cross validation error:', error)
    if (error < model_min_error):
        model_min_error = error
        model_cv_score = cv_score
        model_c_value = i
print("Optimum cv_score:", model_cv_score)
print("Minimum cv_error:", model_min_error)
print("Value of c:", model_c_value)
```

```
Optimum cv_score: 0.9935483870967742
Minimum cv_error: 0.006451612903225823
Value of c: 8
```

In [40]:
```
#plot the data points
### https://matplotlib.org/api/_as_gen/matplotlib.pyplot.scatter.html
fig = mp.figure()
mp.scatter(c_vals,err_vals,s=10)
mp.plot(c_vals,err_vals)

#specify the axes
mp.xlabel("C values in SVC")
mp.ylabel("CV_Error")

#Labeling the plot
#mp.legend(['1'])
#mp.legend(legends)
fig.suptitle('Degree 2, gamma=1', fontsize=15)

#display the current graph
show()
```



Kernel = rbf, gamma = 1, Degree = 5, different c values

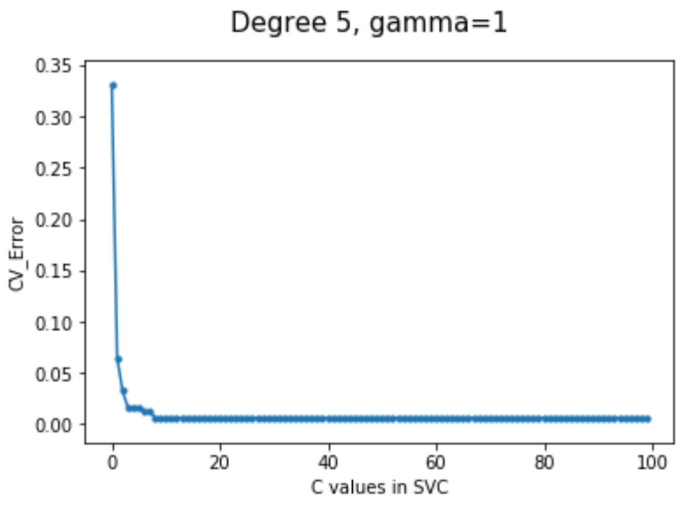
In [41]:
```
#RBF Kernel degree 2 gamma=1
c_vals = []
err_vals = []
svc_model_d20 = SVC(C=0.01, gamma=1, kernel='rbf', degree=5)
c_vals.append(0.01)
cv_score = cross_val_score(svc_model_d20, simpleTrain, trainDigits, cv=10)
print('cv_scores mean:{}'.format(np.mean(cv_score)))
error = 1-np.mean(cv_score)
print('Cross validation error:', error)
err_vals.append(error)
```

```
cv_scores mean:0.6699932795698925
Cross validation error: 0.3300067204301075
```

In [42]:
```
#Polynomial Kernel degree 5 (smallest cross validatio error)
model_min_error = 1
model_cv_score = 0
model_c_value = 0
for i in range(1,100):
    svc_model_d5 = SVC(C=i, gamma=1, kernel='rbf', degree=5)
    c_vals.append(i)
    cv_score = cross_val_score(svc_model_d5, simpleTrain, trainDigits, cv=10)
    cv_score = np.mean(cv_score)
    error = 1-np.mean(cv_score)
    err_vals.append(error)
    #print('Cross validation error:', error)
    if (error < model_min_error):
        model_min_error = error
        model_cv_score = cv_score
        model_c_value = i
print("Optimum cv_score:", model_cv_score)
print("Minimum cv_error:", model_min_error)
print("Value of c:", model_c_value)
```

```
Optimum cv_score: 0.9935483870967742
Minimum cv_error: 0.006451612903225823
Value of c: 8
```

In [43]:
```
#plot the data points
### https://matplotlib.org/api/_as_gen/matplotlib.pyplot.scatter.html
fig = mp.figure()
mp.scatter(c_vals,err_vals,s=10)
mp.plot(c_vals,err_vals)

#specify the axes
mp.xlabel("C values in SVC")
mp.ylabel("CV_Error")

#Labeling the plot
#mp.legend(['1'])
#mp.legend(legends)
fig.suptitle('Degree 5, gamma=1', fontsize=15)

#display the current graph
show()
```



Explanation:There is no much difference in values of degree when gamma is same.

For degree=2, gamma=scale & degree=5, gamma=scale the optimal c in both cases is 19.

For degree=2, gamma=auto & degree=5, gamma=auto the optimal c in both cases is 15.

For degree=2, gamma=1 & degree=5, gamma=1 the optimal c in both cases is 8.

Note: I couldn't experimen with high degrees as it is taking huge time to run.

```
In [ ]:
```