In [33]:
```
import math
import pandas as pd
import numpy as np
import random
from statistics import mean, stdev, median, mode
from sklearn.model_selection import train_test_split,cross_val_score, cross_va
l_predict
from sklearn.decomposition import KernelPCA
import matplotlib.pyplot as plt
import matplotlib.pyplot as mp
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
```

In [34]:
```
data1 = np.loadtxt('data.csv')
#data1
```

In [35]:
```
def minimum(x,y):
    min = np.argmin(y)

    return x[min]
# x=[1,2,3,4]
# y=[5,6,7,8]
# z = minimum(x,y)
# z
```

In [36]:
```
def minimum3(x,y,z):
    min = np.argmin(z)

    return x[min], y[min], z[min]
```

In [37]:
```
def decisionRegion(clf, X, Y):


# 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(clf.predict([[xP,yP]])=="1.0"):
                cPred.append("b")
            else:
                cPred.append("r")

## Visualize Results
#plot the points
    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")
```

In [38]:
```
#shuffle the data and select training and test data
np.random.seed(100)
np.random.shuffle(data1)


features = []
digits = []


for row in data1:
    #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:]
```

In [39]:
```
# Q)1)
```

In [40]:
```
KPCA = KernelPCA(n_components = 2, kernel = 'poly', degree = 1)
data = KPCA.fit_transform(trainFeatures)
```

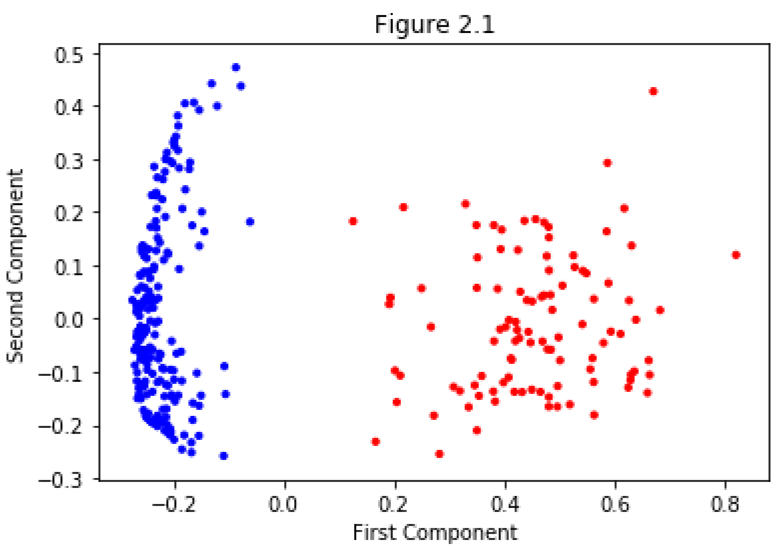
In [41]:
```
train = pd.DataFrame(data = data)
train.head()
```

Out[41]:

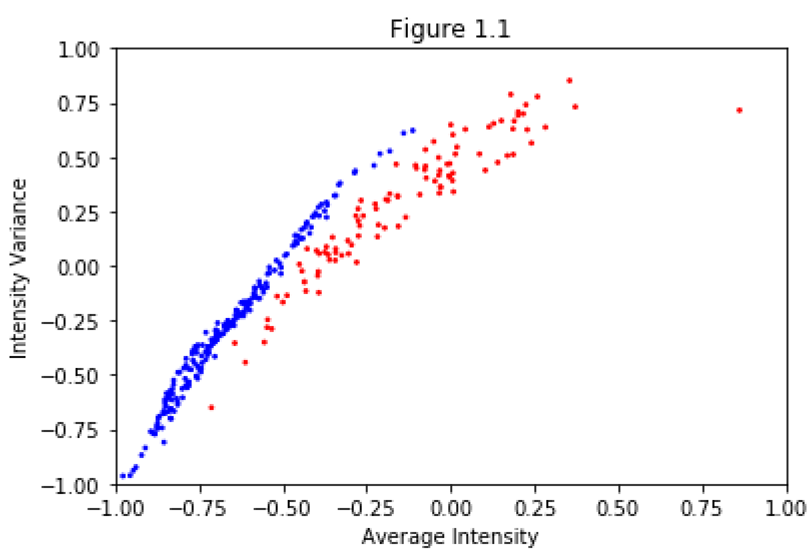
	0	1
0	0.662562	-0.078380
1	-0.249190	-0.186351
2	0.387871	0.056345
3	0.636381	-0.098758
4	-0.171666	0.280755

In [42]:
```
#Colors will be passed to the graphing library to color the points.
#1's are blue: "b" and 5's are red: "r"
colors = []
for index in range(len(trainFeatures)):
    if(trainDigits[index]=="1.0"):
        colors.append("b")
    else:
        colors.append("r")

#plot the data points



plt.scatter(train[[0]],train[[1]], s=10,c=colors)
plt.xlabel('First Component')
plt.ylabel('Second Component')
plt.title("Figure 2.1")
```

Out[42]:
```
Text(0.5, 1.0, 'Figure 2.1')
```



In [43]:
```
#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 = []
for index in range(len(trainFeatures)):
    #produce the 2D dataset for graphing/training and scale the data so it is
 in the [-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 st
ring "1.0" or "5.0"
    if(trainDigits[index]=="1.0"):
        colors.append("b")
    else:
        colors.append("r")

#plot the data points
### https://matplotlib.org/api/_as_gen/matplotlib.pyplot.scatter.html
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")
mp.title("Figure 1.1")
#display the current graph
mp.show()
```



In [44]:
```
# 1)a) Yes. The components of KPCA seem to seperate the data better than mean
 and variance components.
#      It is evident in the figure 2.1 and 1.1 as you can see that there are l
ess number of points near the decision region with KPCA.
```

In [45]:
```
KPCA = KernelPCA(n_components = 2, kernel = 'poly', degree = 3)
data3 = KPCA.fit_transform(trainFeatures)
```

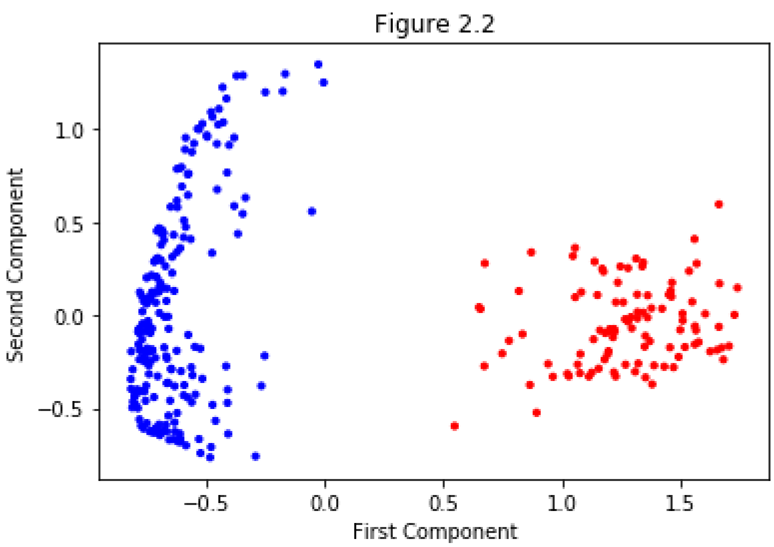
In [46]:
```
train3 = pd.DataFrame(data = data3)
train3.head()
```

Out[46]:

	0	1
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

In [47]:
```
#Colors will be passed to the graphing library to color the points.
#1's are blue: "b" and 5's are red: "r"
colors = []
for index in range(len(trainFeatures)):
    if(trainDigits[index]=="1.0"):
        colors.append("b")
    else:
        colors.append("r")

#plot the data points



plt.scatter(train3[[0]],train3[[1]], s=10,c=colors)
plt.xlabel('First Component')
plt.ylabel('Second Component')
plt.title("Figure 2.2")
```

Out[47]:
```
Text(0.5, 1.0, 'Figure 2.2')
```

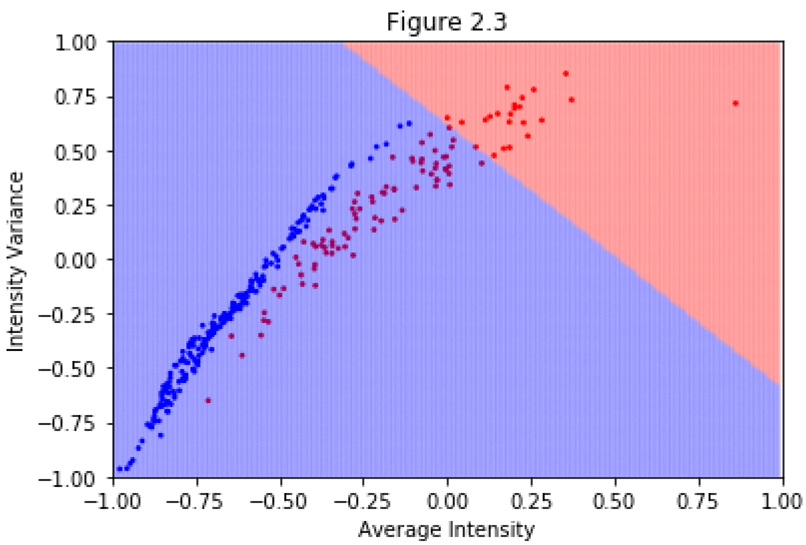


In [48]:
```
# 1)b) Yes. This kpca (degree 3) seperates the data better than the above one
 (degree 1) because there are less data points near the
# decision boundary when compared to the no of data points near the decision b
oundary for kpca of degree 1
```

In [49]:
```
# Q)2)
```

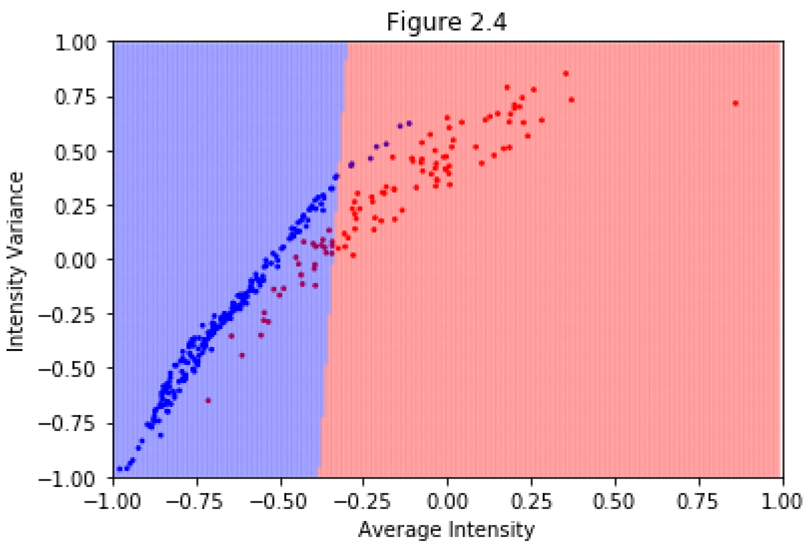
In [50]:
```
LR = LogisticRegression(penalty = 'l2',C = 0.01)
clf = LR.fit(simpleTrain, trainDigits)

decisionRegion(clf, X, Y)
mp.title("Figure 2.3")
mp.show()
```

```
C:\Users\kalya\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:4
32: FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify
a solver to silence this warning.
  FutureWarning)
```



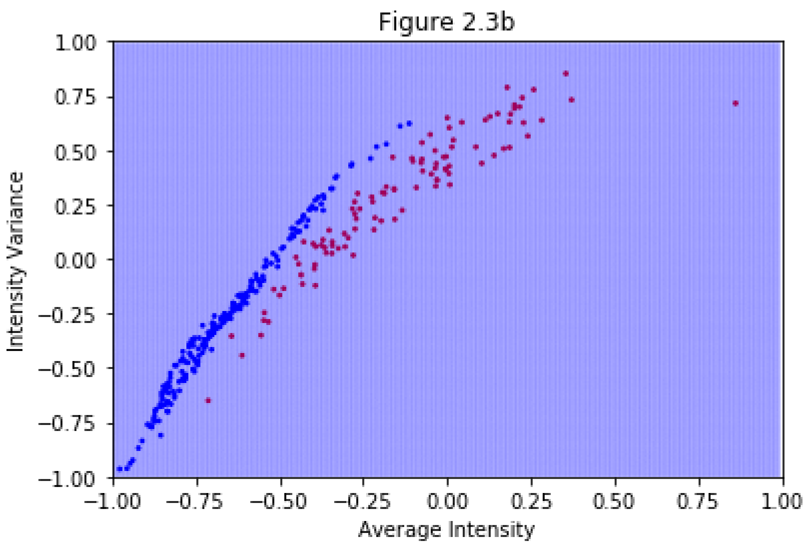
In [51]:
```
LR = LogisticRegression(penalty = 'l2',C = 2.0)
clf = LR.fit(simpleTrain, trainDigits)

decisionRegion(clf, X, Y)
mp.title("Figure 2.4")
mp.show()
```

```
C:\Users\kalya\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:4
32: FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify
a solver to silence this warning.
  FutureWarning)
```



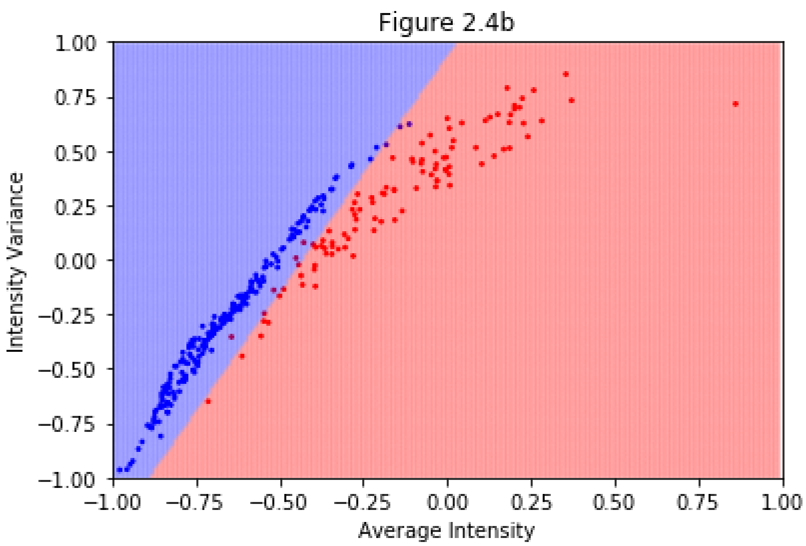
In [52]:
```
LR = LogisticRegression(penalty = 'l1',C = 0.01)
clf = LR.fit(simpleTrain, trainDigits)

decisionRegion(clf, X, Y)
mp.title("Figure 2.3b")
mp.show()
```

```
C:\Users\kalya\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:4
32: FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify
a solver to silence this warning.
  FutureWarning)
```



In [53]:
```
LR = LogisticRegression(penalty = 'l1',C = 2.0)
clf = LR.fit(simpleTrain, trainDigits)


decisionRegion(clf, X, Y)
mp.title("Figure 2.4b")
mp.show()
```

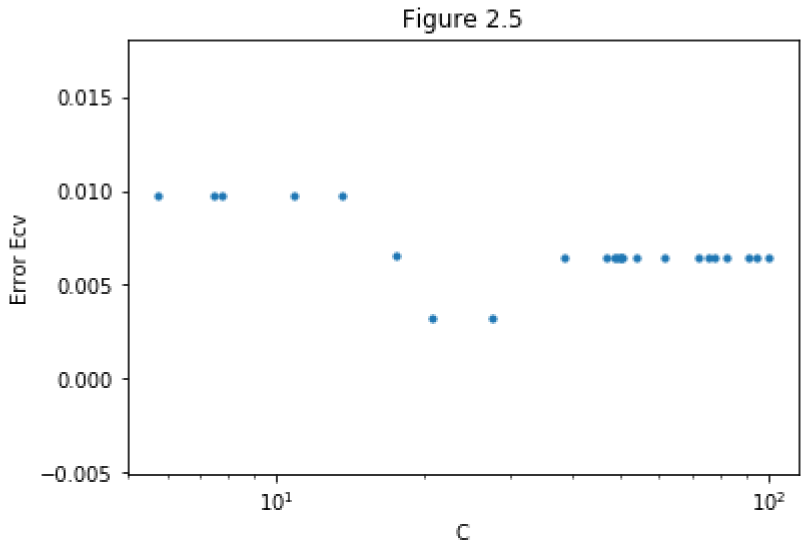
```
C:\Users\kalya\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:4
32: FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify
a solver to silence this warning.
  FutureWarning)
```

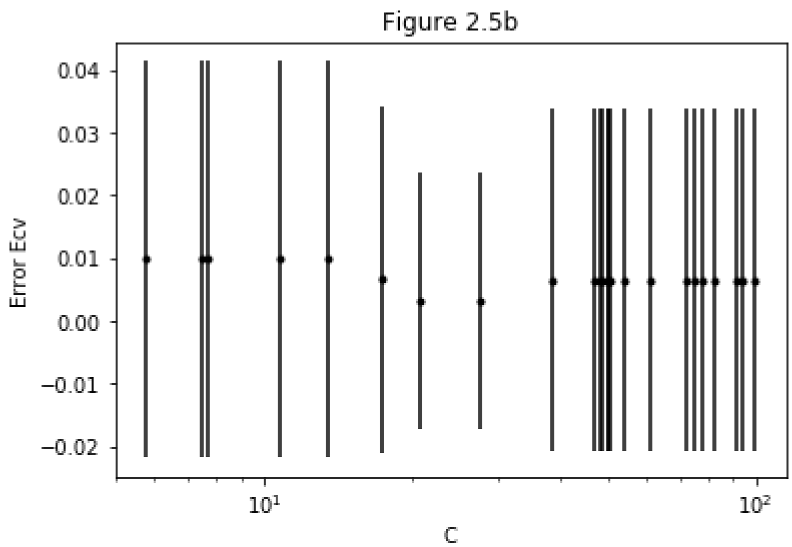


In [54]:
```
#Q) 2) Graduate student question:

#  From the figures 2.3 and 2.3b, for C = 0.01, both l1 and l2 underfit the da
ta but l1 completely underfit the data
#  For C = 2.0, the regularization techniques behaved conversely, l1 has less
 number of training errors compared to l2
#  It looks like the regularizaton parameter C has more role to play than regu
larization techniques in making a model overfit or underfit.
```

In [55]:
```
# Q) 3)
```

In [56]:
```
# USING 2D dimensional data
x = []
y = []
z = []
p =[]
for i in range(1,25):
    c = np.random.uniform(0.01,100)
    model2  = SVC(C= c, gamma= 'scale')
    #model2.predict(testFeatures)
    cvs = cross_val_score(model2, simpleTrain, trainDigits, cv = 10, scoring=
'accuracy')
    err = 1-cvs
    evsm = 1-(cvs.mean())
    p.append(err)
    x.append(c)
    y.append(evsm)
    z.append([x,evsm])

# print(len(x))
# print(len(y))
# print(count)
mp.scatter(x,y, s=10)
mp.xlabel("C")
mp.xscale('log')
mp.ylabel("Error Ecv")
mp.title("Figure 2.5")
mp.show()

m=[]
std =[]
#print(y)
for i in range(len(p)):
    m.append(mean(p[i]))
    std.append(2*stdev(p[i]))


mp.errorbar(x, m, yerr=std, fmt='.k');

mp.xlabel("C")
mp.xscale('log')
mp.ylabel("Error Ecv")
mp.title("Figure 2.5b")
mp.show()
```

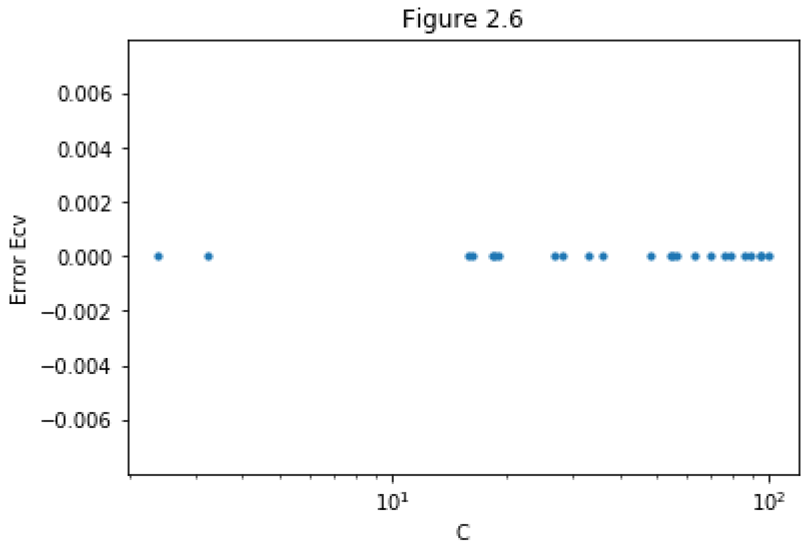


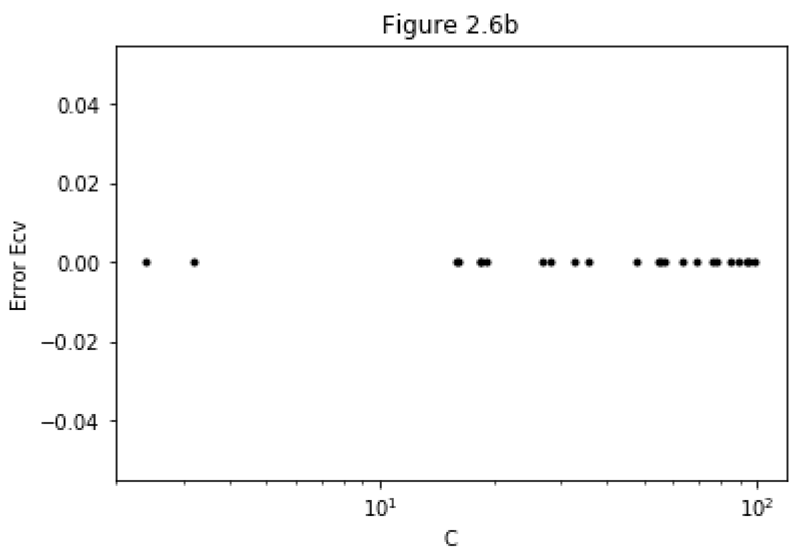


In [57]:
```
# from Figure 2.5, the error is low for x = 3
c_opt = minimum(x,y)
c_opt
```

Out[57]: 27.404470110232598

In [58]:
```
# USING 256D dimensional data
x256 = []
y256 = []
p256 =[]

for i in range(1,25):
    c256 = np.random.uniform(0.01,100)
    model2  = SVC(C= c256, gamma= 'scale')
    #model2.predict(testFeatures)
    cvs256 = cross_val_score(model2, trainFeatures, trainDigits, cv = 10, scor
ing='accuracy')
    err256 = 1-cvs256
    evsm256 = 1-(cvs256.mean())
    p256.append(err256)
    x256.append(c256)
    y256.append(evsm256)

# print(len(x))
# print(len(y))
# print(count)
mp.scatter(x256,y256, s=10)
mp.xlabel("C")
mp.xscale('log')
mp.ylabel("Error Ecv")
mp.title("Figure 2.6")
mp.show()

m256=[]
std256 =[]
#print(y)
for i in range(len(p256)):
    m256.append(mean(p256[i]))
    std256.append(2*stdev(p256[i]))


mp.errorbar(x256, m256, yerr=std256, fmt='.k');

mp.xlabel("C")
mp.xscale('log')
mp.ylabel("Error Ecv")
mp.title("Figure 2.6b")
mp.show()

print(np.sort(x256))
```

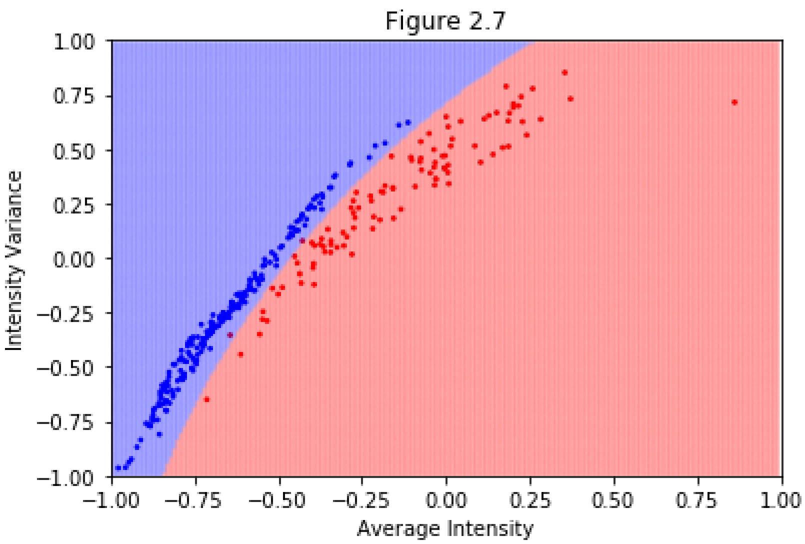





```
[ 2.38892019  3.22532004 15.85696226 16.20335476 18.43898396 18.48234987
 19.08740535 26.97663363 28.27696536 32.96013892 35.85365496 48.01854316
 54.69987977 55.4738074  56.65342317 63.14267019 69.49737684 75.77161746
 78.54534258 85.53950334 89.36686788 94.14115424 94.5088163  98.84508942]
```

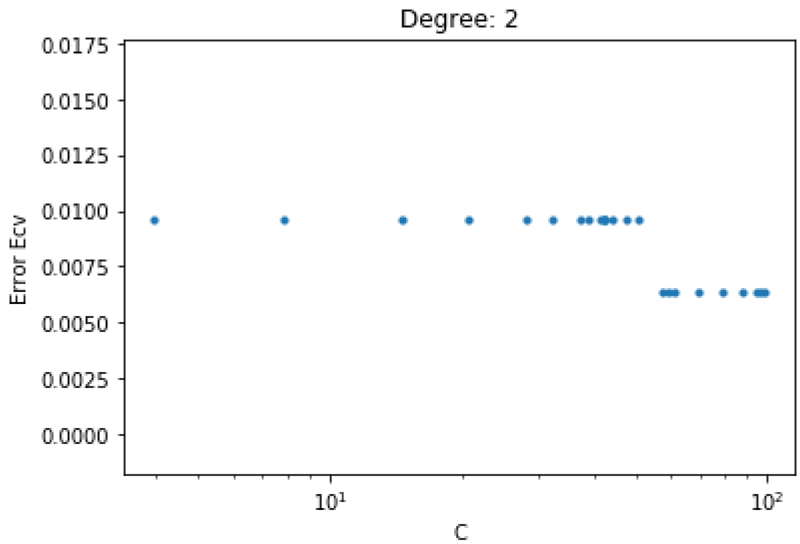
In [59]:
```
# Since the 256 dimensional data is linearly seperable, we got zero training e
rrors for all C values.
# But if you want me to pick a value for C, I would pick 44 because a small va
lue of C would underfit the test data set and
#  a large value of C would overfit the test data
```

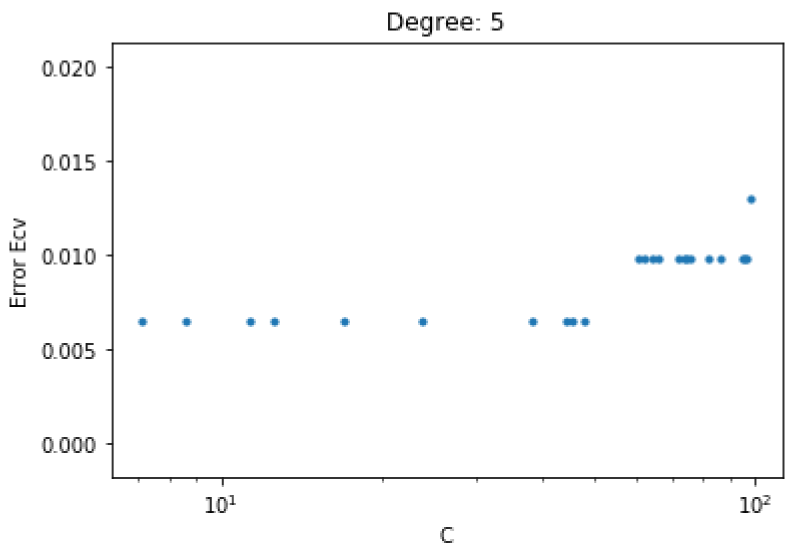
In [60]:
```
svm = SVC(C = c_opt, gamma= 'auto')
clf = svm.fit(simpleTrain, trainDigits)

decisionRegion(clf, X, Y)
mp.title("Figure 2.7")
mp.show()
```

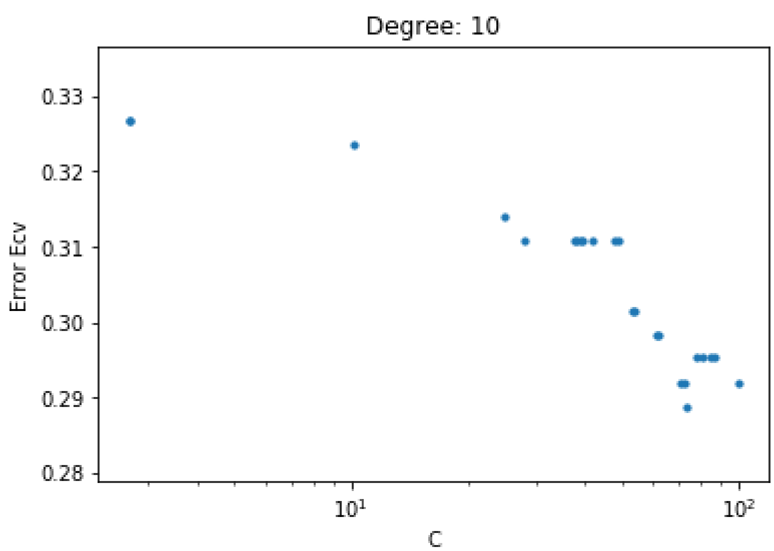


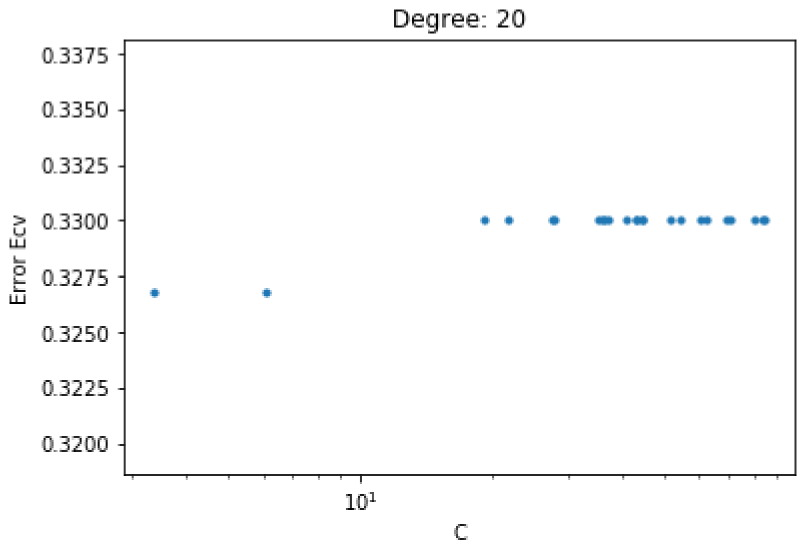
In [74]:
```
# Gamma vs Degree
# It is taking a lot of time for the following cell to run for degree of 10 an
d 20 with Gamma = 'scale'.
# So, I used Gamma = 1 for those two degree

x = []
y = []
z = []
p =[]
c_opt4_1 = []
degree1 = [2,5]
degree2 = [10,20]
r1= []
for i in degree1:
    x1 = []
    y1 = []
    z1 = []
    p1 =[]
    for j in range(1,25):
        c1 = np.random.uniform(0.01,100)
        svm = SVC(kernel = 'poly', C = c1, degree = i, gamma= 'scale')
        cvs = cross_val_score(svm, simpleTrain, trainDigits, cv = 10, scoring=
'accuracy', n_jobs = -1)
        err = 1-cvs
        evsm = 1-(cvs.mean())
        p.append(err)
        x.append(c1)
        y.append(evsm)
        z.append(i)
        p1.append(err)
        x1.append(c1)
        y1.append(evsm)
        z1.append(i)
    coptimal = minimum(x1,y1)
    c_opt4_1.append(coptimal)
    r1.append(y[np.argmin(y1)])
    mp.scatter(x1,y1, s=10)
    mp.xlabel("C")
    mp.xscale('log')
    mp.ylabel("Error Ecv")
    mp.title("Degree: " + str(i))
    mp.show()
#     svm = SVC(kernel = 'poly', C = coptimal, degree = i, gamma= 'scale')
#     svm.fit(simpleTrain, trainDigits)
#     decisionRegion(svm, X, Y)
#     mp.title("Degree: " + str(i))
#     mp.show()

c_opt4_2 = []
r2 =[]
for i in degree2:
    x2 = []
    y2 = []
    z2 = []
    p2 =[]
    for j in range(1,25):
```

```
        c2 = np.random.uniform(0.01,100)
        svm = SVC(kernel = 'poly', C = c2, degree = i, gamma= 1)
        cvs = cross_val_score(svm, simpleTrain, trainDigits, cv = 10, scoring=
'accuracy', n_jobs = -1)
        err = 1-cvs
        evsm = 1-(cvs.mean())
        p2.append(err)
        x2.append(c2)
        y2.append(evsm)
        z2.append(i)
        p1.append(err)
        x1.append(c2)
        y1.append(evsm)
        z1.append(i)
    coptimal = minimum(x2,y2)
    c_opt4_2.append(coptimal)
    r2.append(y[np.argmin(y2)])
    mp.scatter(x2,y2, s=10)
    mp.xlabel("C")
    mp.xscale('log')
    mp.ylabel("Error Ecv")
    mp.title("Degree: " + str(i))
    mp.show()
#     svm = SVC(kernel = 'poly', C = coptimal, degree = i, gamma= 1)
#     svm.fit(simpleTrain, trainDigits)
#     decisionRegion(svm, X, Y)
#     mp.title("Degree: " + str(i))
#     mp.show()

c_opt = c_opt4_1 + c_opt4_2
r = r1 + r2
degree = degree1 + degree2

print('\n')
print("Optimal C Values for Degrees " + str(degree) +" are: " + str(c_opt))

d_opt, c_optf, Min_Ecv = minimum3(z,x,y)
print('\n')
print("Min Ecv Value is: " + str(Min_Ecv))
print("Optimal degree is: " + str(d_opt))
print("Optimal C Value is: " + str(c_optf))


plt.scatter(degree,r)
plt.xlabel('Degree')
plt.ylabel('Min Ecv')
plt.title('Gamma vs Ecv')
plt.show()

plt.scatter(degree,c_opt)
plt.xlabel('Degree')
plt.ylabel('C value of Min Ecv')
plt.title('Degree vs C')
plt.show()
```

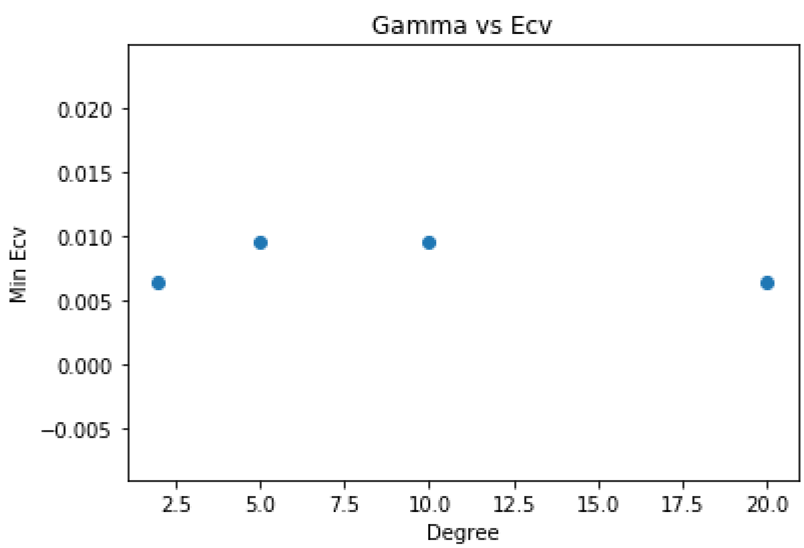

Degree: 2
0.0175
0.0150
0.0125
0.0100
0.0075
0.0050
0.0025
0.0000
Error Ecv
10^1
10^2
C

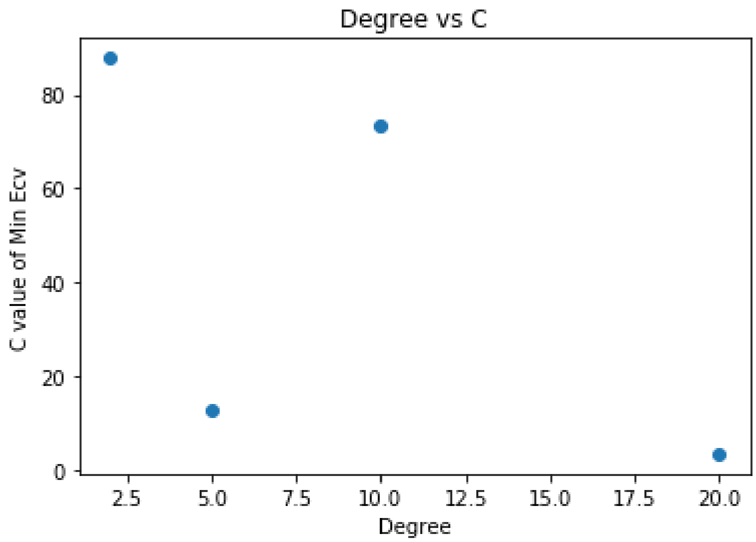

Degree: 5
0.020
0.015
0.010
0.005
0.000
Error Ecv
10^1
10^2
C


Degree: 10
0.33
0.32
0.31
0.30
0.29
0.28
Error Ecv
10^1
10^2
C



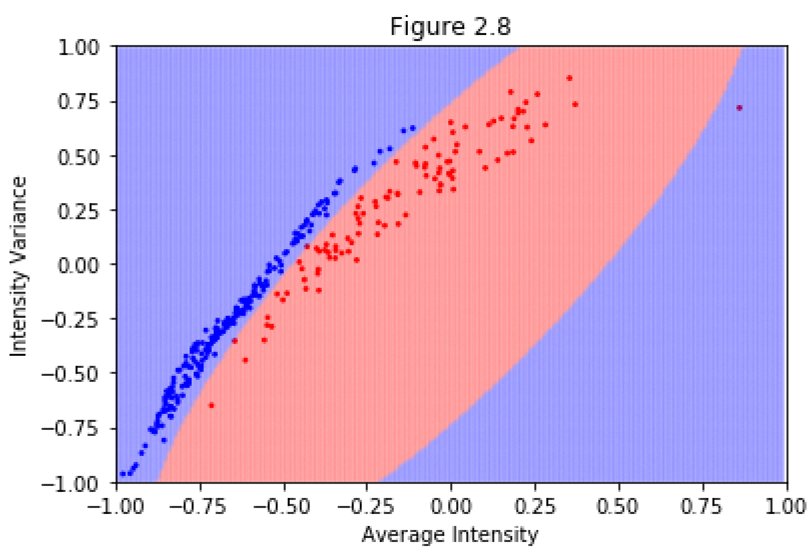

```
Optimal C Values for Degrees [2, 5, 10, 20] are: [87.83078117113321, 12.60714
9031715023, 73.48612969804714, 3.3801631824841496]


Min Ecv Value is: 0.006350806451612789
Optimal degree is: 2
Optimal C Value is: 87.83078117113321
```





In [75]:
```
# Q) 3)
# With smaller degree, there is higher chance of underfitting so, to compensat
e that we need a greater C value which gave me lesser Ecv.
#   And in case of higher degree which causes overfitting, we need smaller C v
alue to get less cross validation error Ecv
#   It is evident in the above observations.
```

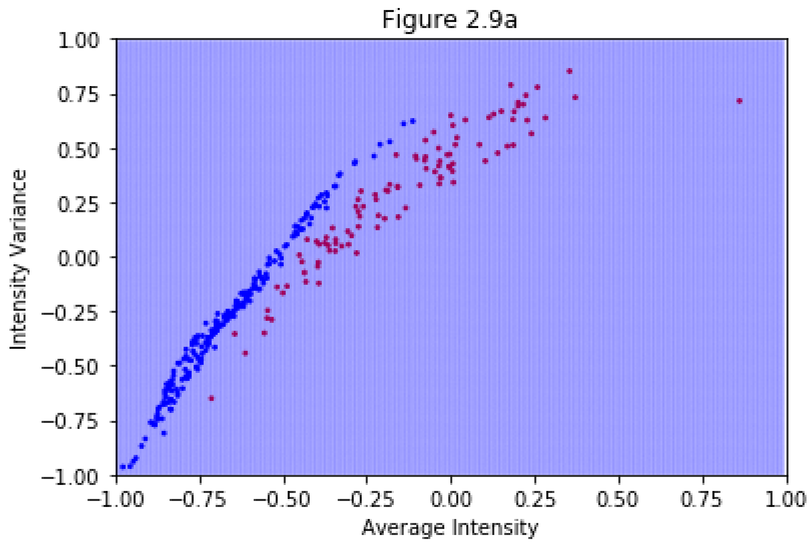
In [76]:
```
svm = SVC(C = c_optf, kernel = 'poly', degree = d_opt, gamma= 'scale')
clf = svm.fit(simpleTrain, trainDigits)

decisionRegion(clf, X, Y)
mp.title("Figure 2.8")
mp.show()
```



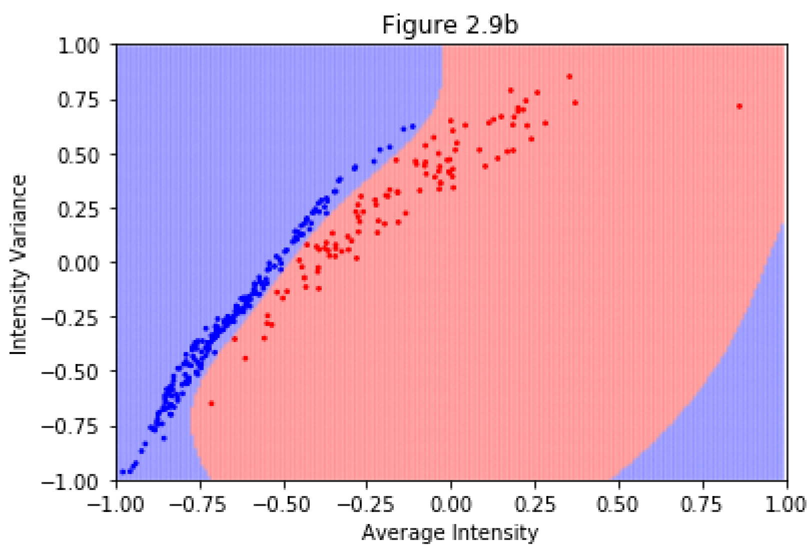
In [77]:
```
# Q)3) From the above plot, it is evident that this SVM model is best in terms
of seperating the training data.
```

In [78]:
```
# Graduate Student Question
```

In [79]:
```
svm = SVC(C = 0.000001, gamma = 'scale')
clf = svm.fit(simpleTrain, trainDigits)


decisionRegion(clf, X, Y)
mp.title("Figure 2.9a")
mp.show()
```



In [80]:
```
svm = SVC(C = 10000, gamma = 'scale')
clf = svm.fit(simpleTrain, trainDigits)

decisionRegion(clf, X, Y)
mp.title("Figure 2.9b")
mp.show()
```

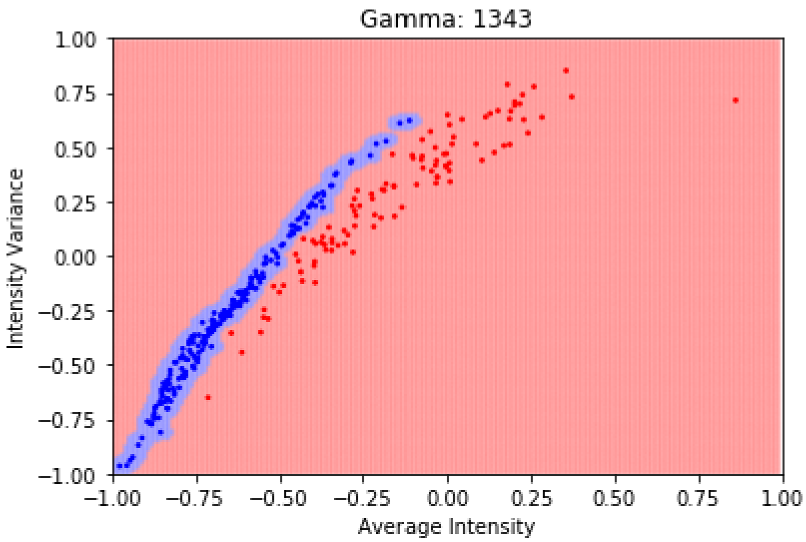


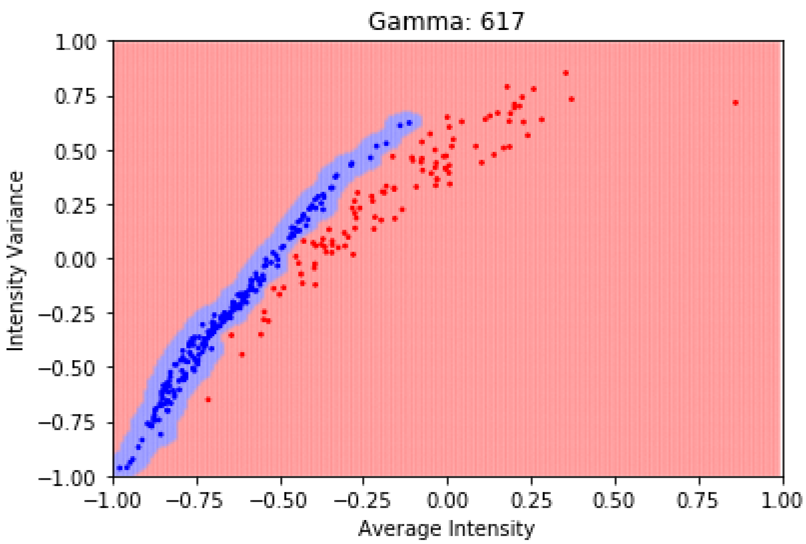
In [81]:
```
# Q)3) Graduate Student Question
# When C is low, the hinge loss would be high and svm model produces high marg
in and the model underfits. It is evident in the figure 2.9a
# When C is high, the hinge loss would be low and SVM model tries to overfit t
he data. Check the figure 2.9b
```

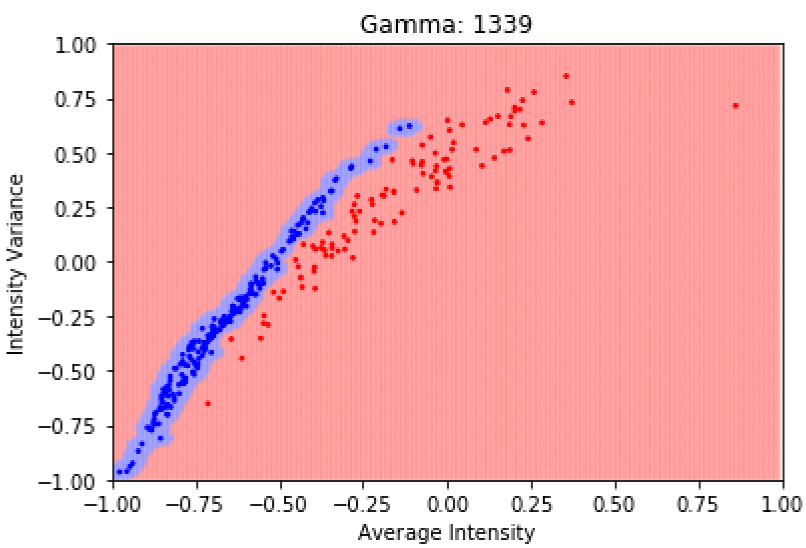
In [82]:
```
# Extra Credit
```

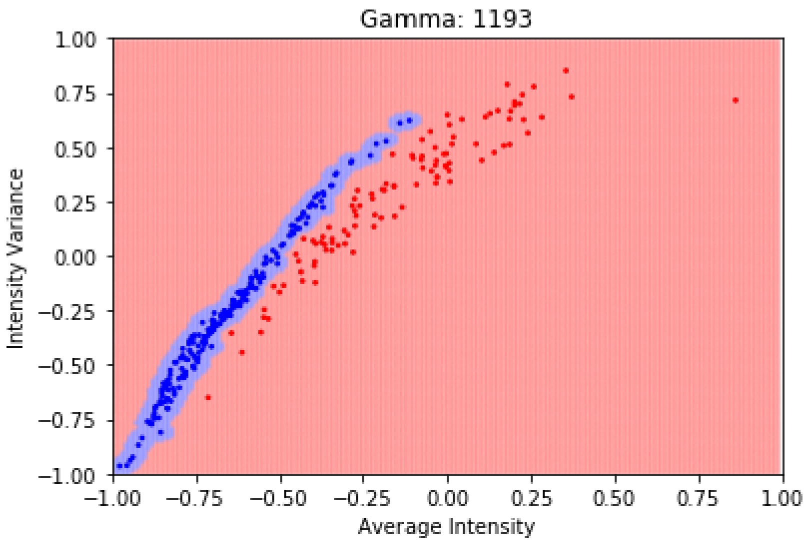
In [ ]:
```
```

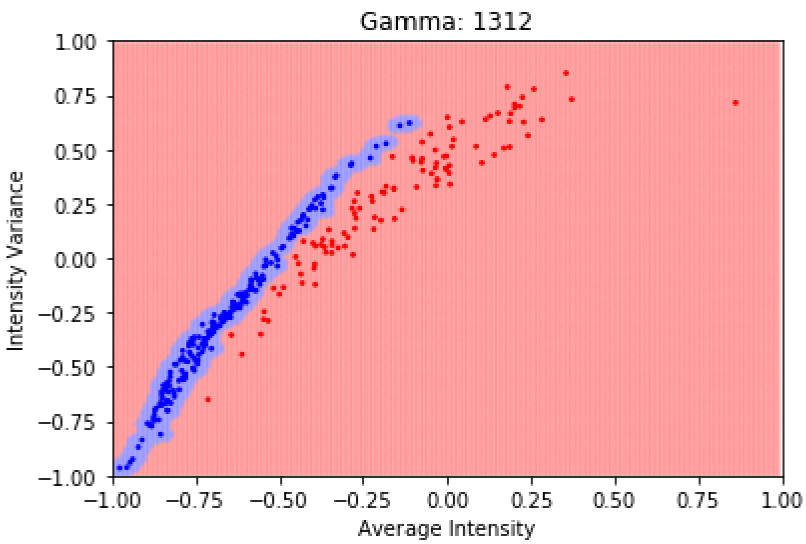
In [83]:
```
# Varying only C. Keeping C and Gamma constant
x = []
y = []
z = []
p =[]
degree = [2,5,10,20]

for i in range(1,10):
    gamma = np.random.randint(1,1500)
    svm = SVC(kernel = 'rbf', gamma= gamma)
    cvs = cross_val_score(svm, simpleTrain, trainDigits, cv = 10, scoring='acc
uracy')
    err = 1-cvs
    evsm = 1-(cvs.mean())
    p.append(err)
    x.append(gamma)
    y.append(evsm)
    z.append([x,evsm])
    svm.fit(simpleTrain, trainDigits)
    decisionRegion(svm,X, Y)
    mp.title("Gamma: " + str(gamma))
    mp.show()

z = minimum(x,y)
print("Optimal Value of Gamma = " + str(z))
mp.scatter(x,y)
mp.xlabel('Gamma')
mp.ylabel('Ecv')
mp.title("Gamma vs Ecv")
mp.show()
```

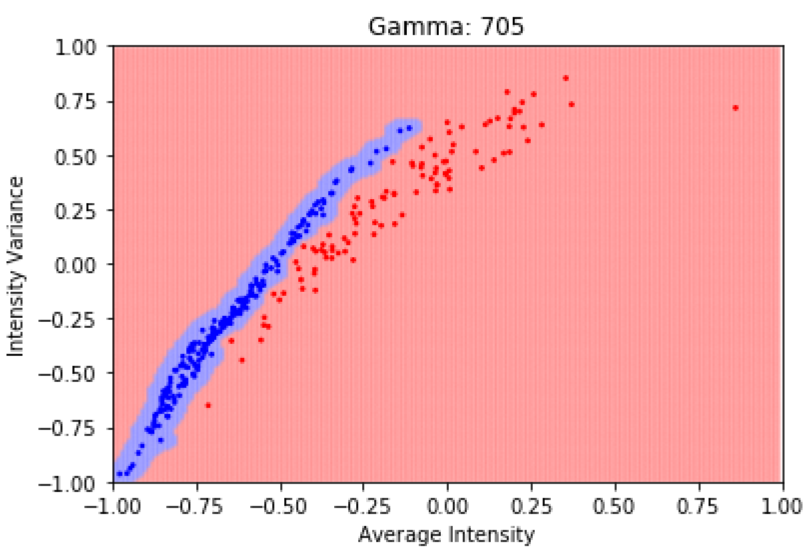

Gamma: 1343
Intensity Variance
Average Intensity
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.00 −0.75 −0.50 −0.25 0.00 0.25 0.50 0.75 1.00

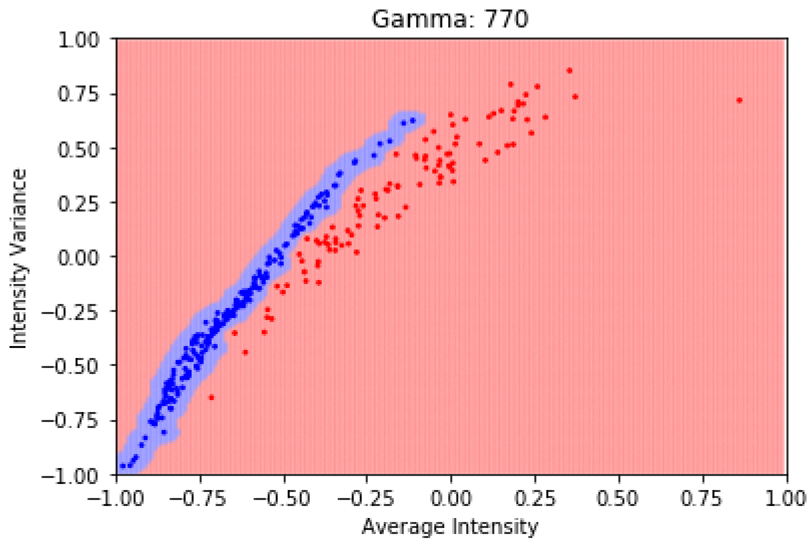

Gamma: 617
Intensity Variance
Average Intensity
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.00 −0.75 −0.50 −0.25 0.00 0.25 0.50 0.75 1.00

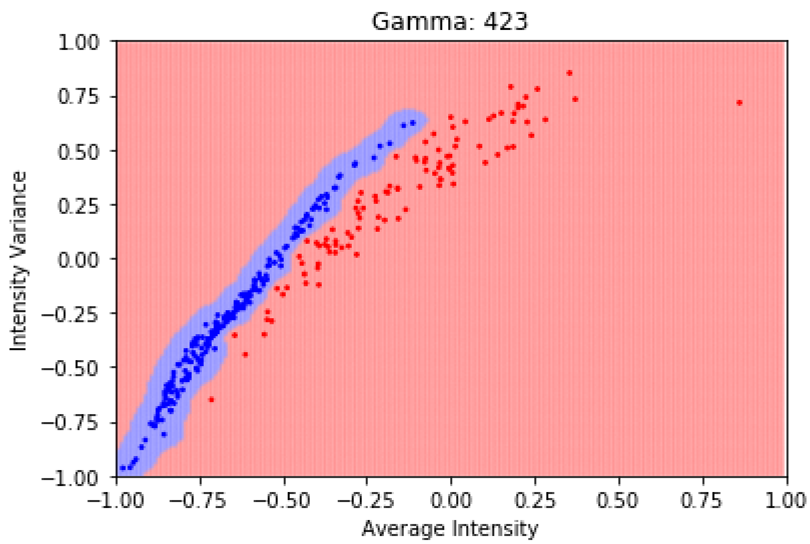

Gamma: 1339
Intensity Variance
Average Intensity
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.00 −0.75 −0.50 −0.25 0.00 0.25 0.50 0.75 1.00

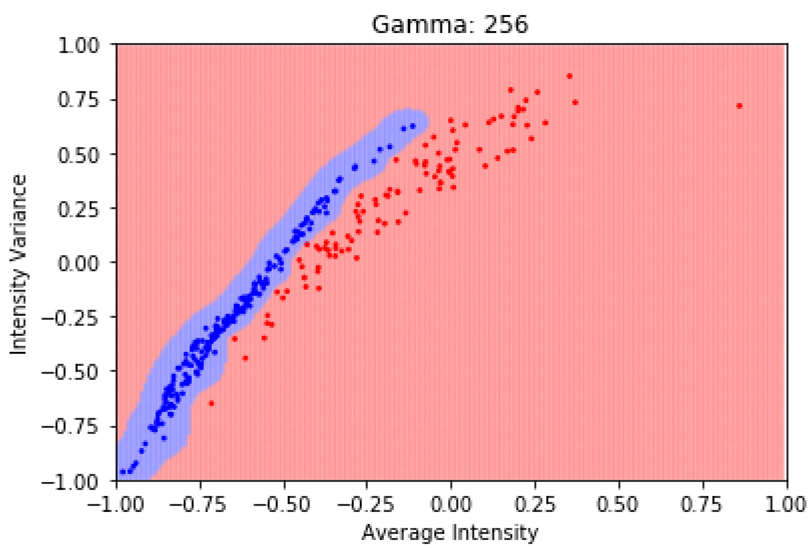

Gamma: 1193
Intensity Variance
Average Intensity
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.00 −0.75 −0.50 −0.25 0.00 0.25 0.50 0.75 1.00


Gamma: 1312
Intensity Variance
Average Intensity
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.00 −0.75 −0.50 −0.25 0.00 0.25 0.50 0.75 1.00

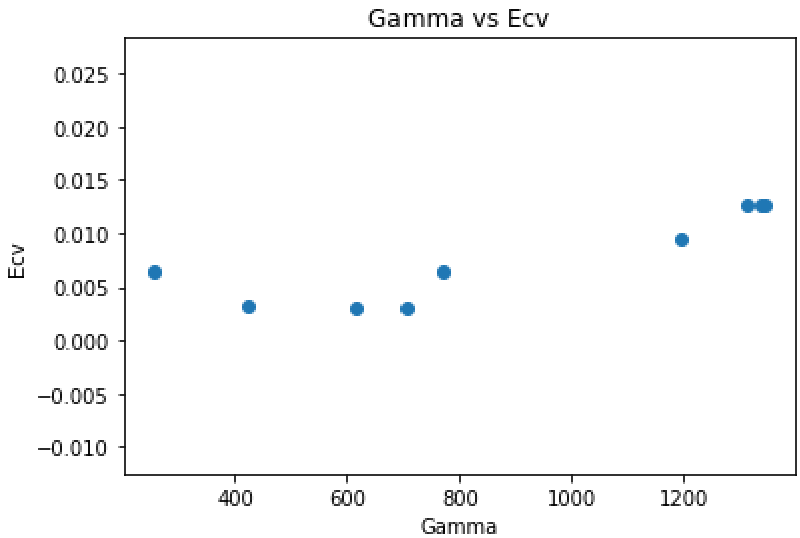

Gamma: 705
Intensity Variance
Average Intensity
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.00 −0.75 −0.50 −0.25 0.00 0.25 0.50 0.75 1.00







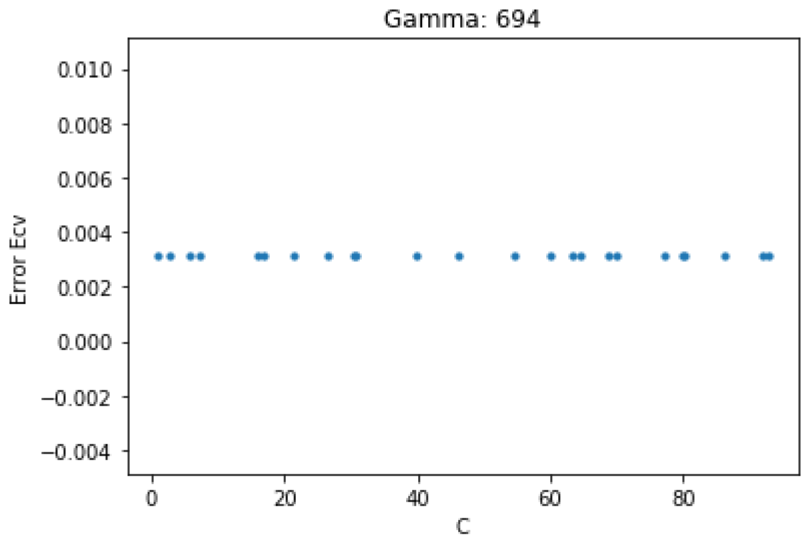

```
Optimal Value of Gamma = 617
```

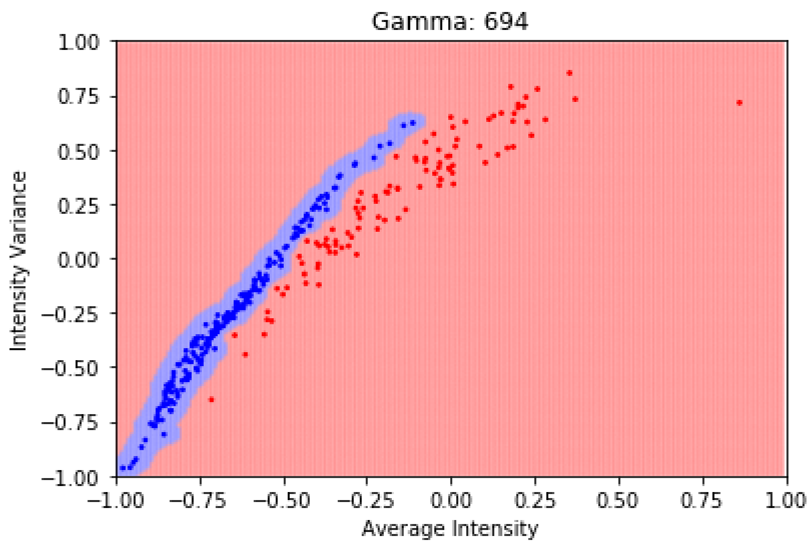


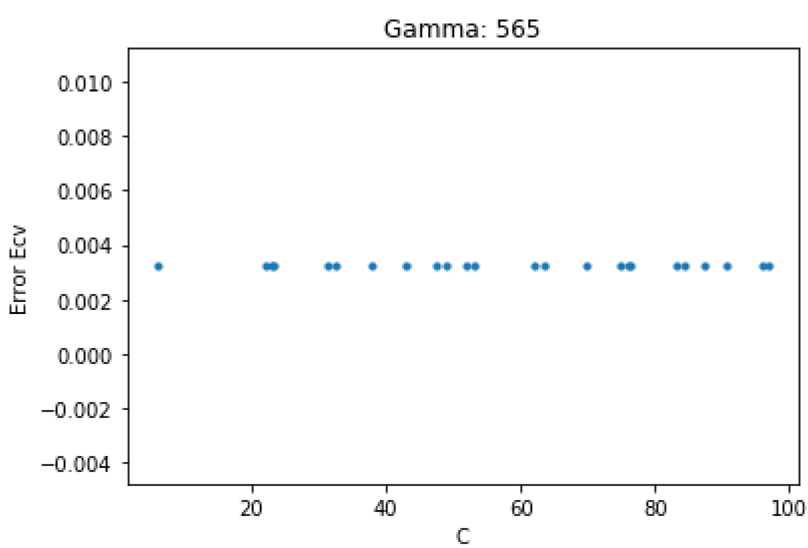
In [84]:
```
# From the above plots, we could say that the models with smaller values of ga
mma underfit the data.
# And the models with greater values of gamma formed smaller islands i.e. over
fit the data.
```

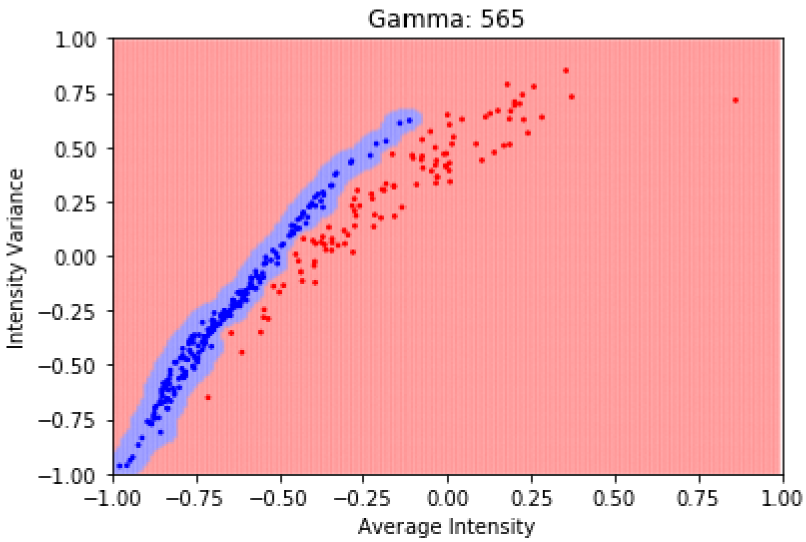
In [92]:
```
# Varying both C and Gamma
x1 = []
y1 = []
z1 = []
p1 =[]
c_opt4 = []
g =[]
r =[]

for i in range(1,5):
    x = []
    y = []
    z = []
    p =[]
    gamma = np.random.randint(1,1500)
    g.append(gamma)
    for j in range(1,25):
        c = np.random.uniform(0.01,100)
        svm = SVC(kernel = 'rbf', C = c, gamma= gamma)
        cvs = cross_val_score(svm, simpleTrain, trainDigits, cv = 10, scoring=
'accuracy', n_jobs = -1)
        err = 1-cvs
        evsm = 1-(cvs.mean())
        p.append(err)
        x.append(c)
        y.append(evsm)
        z.append(gamma)
        p1.append(err)
        x1.append(c)
        y1.append(evsm)
        z1.append(gamma)
    coptimal = minimum(x,y)
    c_opt4.append(coptimal)
    r.append(y[np.argmin(y)])
    mp.scatter(x,y, s=10)
    mp.xlabel("C")
    mp.ylabel("Error Ecv")
    mp.title("Gamma: " + str(gamma))
    mp.show()
    svm = SVC(kernel = 'rbf', C = coptimal, gamma= gamma)
    svm.fit(simpleTrain, trainDigits)
    decisionRegion(svm, X, Y)
    mp.title("Gamma: " + str(gamma))
    mp.show()

print('\n')
print("Optimal C Values for Gamma " + str(g) +" are: " + str(c_opt4))

g_opt, c_optf, Min_Ecv = minimum3(z1,x1,y1)
print('\n')
print("Min Ecv Value is: " + str(Min_Ecv))
print("Optimal Gamma is: " + str(g_opt))
print("Optimal C Value is: " + str(c_optf))

plt.scatter(g,r)
plt.xlabel('Gamma')
```

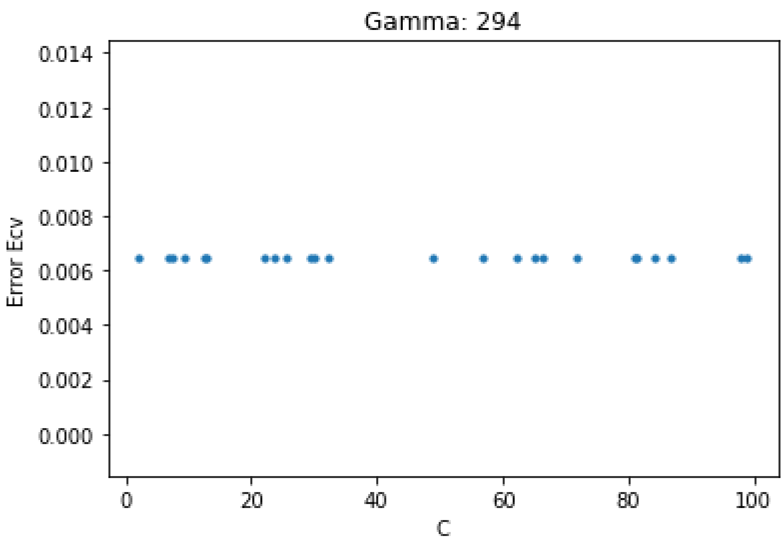
```
plt.ylabel('Min Ecv')
plt.title('Gamma vs Ecv')
plt.show()

plt.scatter(g,c_opt4)
plt.xlabel('Gamma')
plt.ylabel('C value of Min Ecv')
plt.title('Gamma vs C')
plt.show()
```

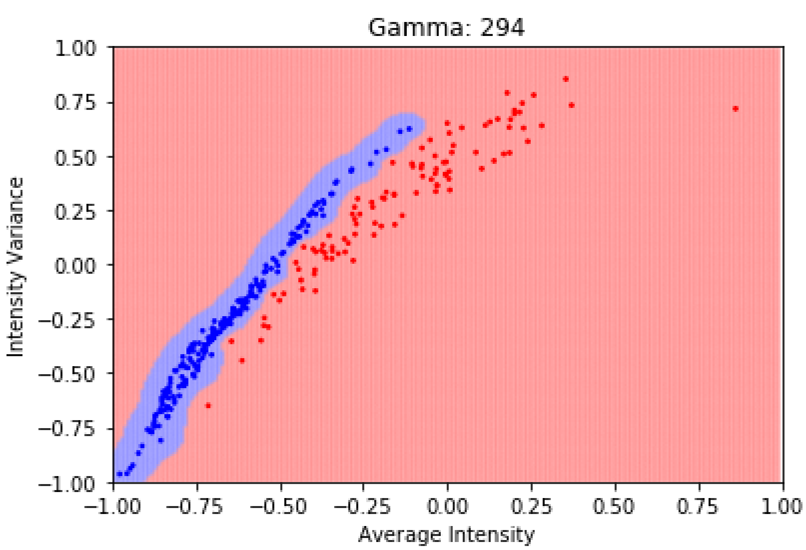

Gamma: 694
Error Ecv
C
0.010
0.008
0.006
0.004
0.002
0.000
−0.002
−0.004
0
20
40
60
80

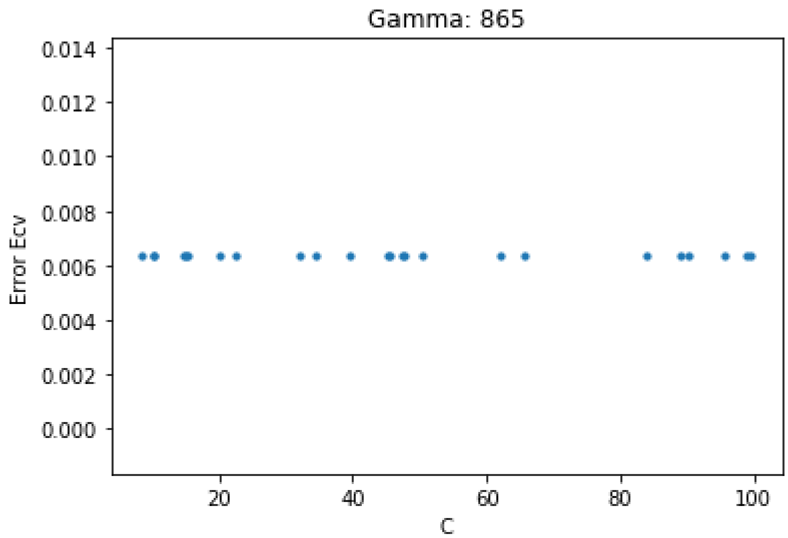

Gamma: 694
Intensity Variance
Average Intensity
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

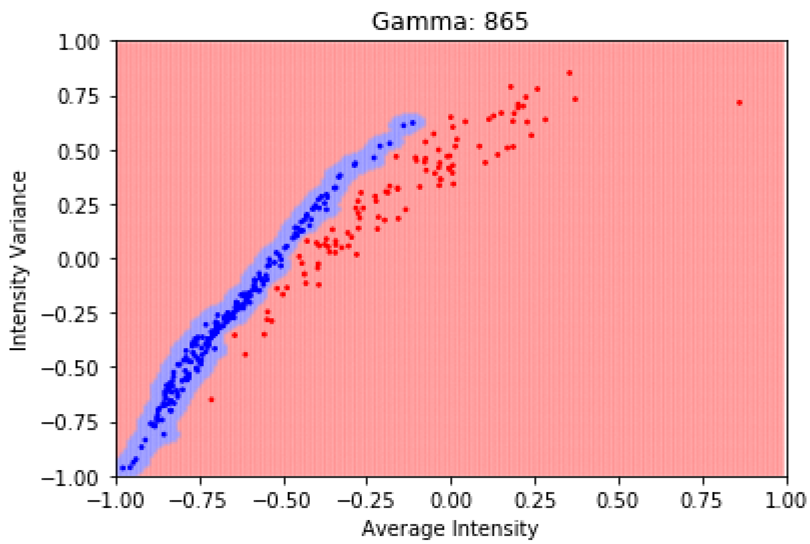

Gamma: 565
Error Ecv
C
0.010
0.008
0.006
0.004
0.002
0.000
−0.002
−0.004
20
40
60
80
100


Gamma: 565
Intensity Variance
Average Intensity
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.00 −0.75 −0.50 −0.25 0.00 0.25 0.50 0.75 1.00

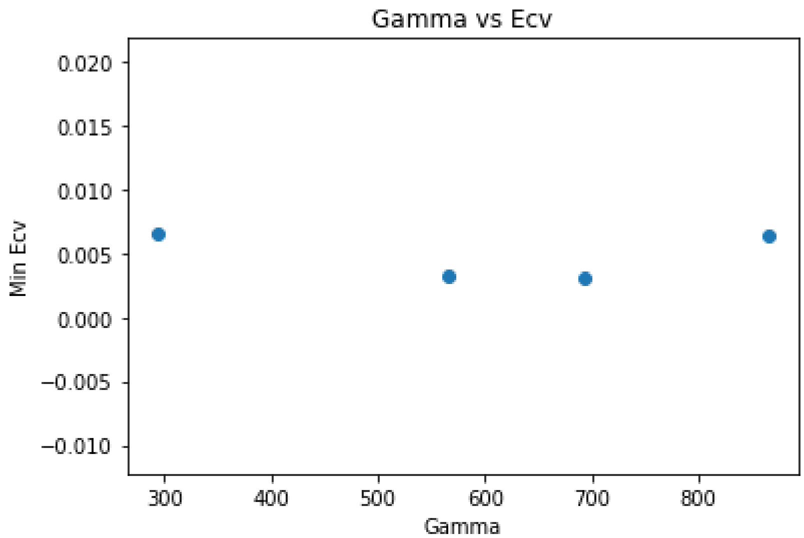

Gamma: 294
Error Ecv
C
0.014
0.012
0.010
0.008
0.006
0.004
0.002
0.000
0 20 40 60 80 100

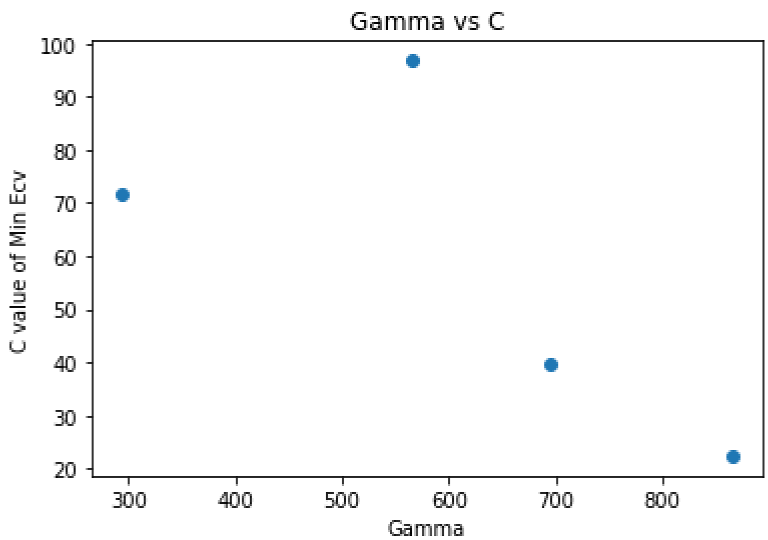

Gamma: 294
Intensity Variance
Average Intensity
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.00 −0.75 −0.50 −0.25 0.00 0.25 0.50 0.75 1.00





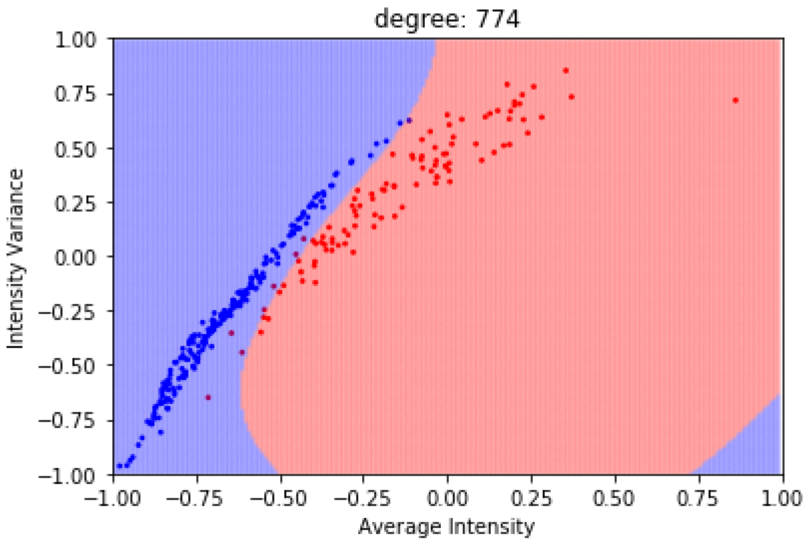

```
Optimal C Values for Gamma [694, 565, 294, 865] are: [39.87402383420889, 96.7
9593140378634, 71.77385236560717, 22.442276346482515]


Min Ecv Value is: 0.0031250000000000444
Optimal Gamma is: 694
Optimal C Value is: 39.87402383420889
```

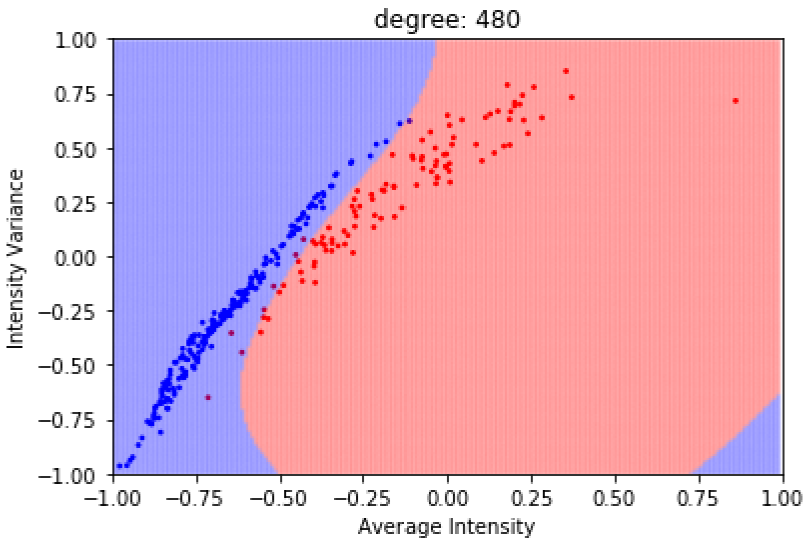


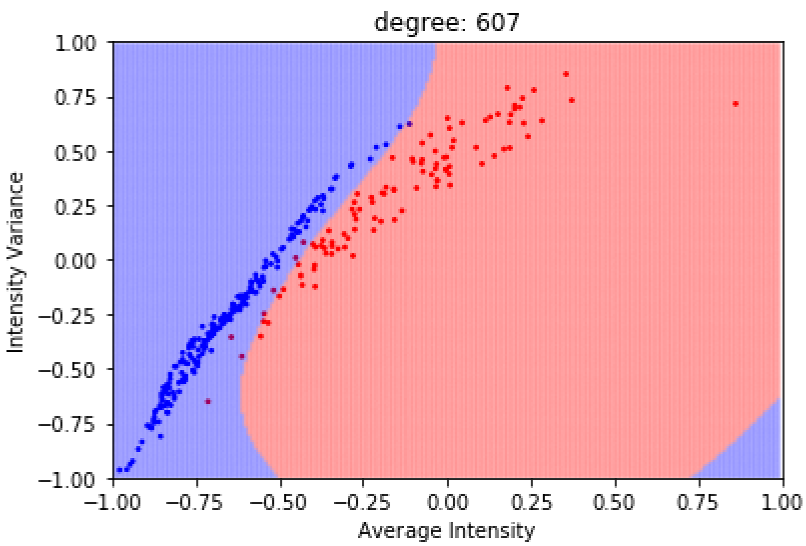


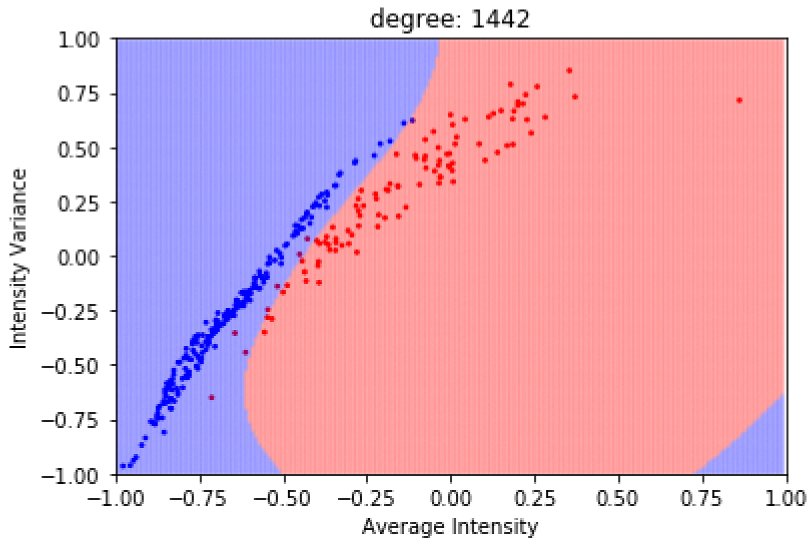
In [93]:
```
# As models with higher values of Gamma tend to overfit the data, we need smal
ler value of C to get minimum cross validation error.
# Similarly, we need higher value of C for models with smaller value of Gamma
 as these models tend to underfit the data
# It makes sense from the above Gamma vs C plot
```

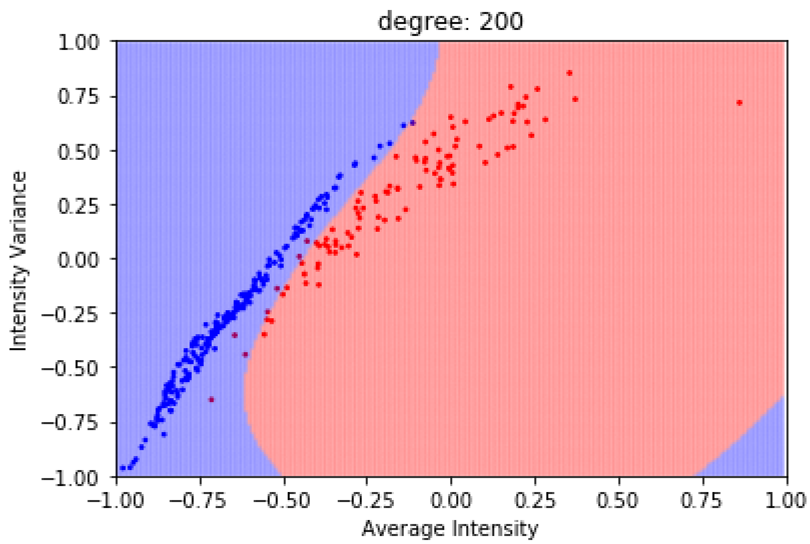
In [90]:
```
# Varying only degree. Keeping C and Gamma constant
x = []
y = []
z = []
p =[]
degree = [2,5,10,20]

for i in range(1,10):
    degree = np.random.randint(1,1500)
    svm = SVC(kernel = 'rbf', gamma= 'scale', degree = degree)
    cvs = cross_val_score(svm, simpleTrain, trainDigits, cv = 10, scoring='acc
uracy')
    err = 1-cvs
    evsm = 1-(cvs.mean())
    p.append(err)
    x.append(gamma)
    y.append(evsm)
    z.append([x,evsm])
    svm.fit(simpleTrain, trainDigits)
    decisionRegion(svm,X, Y)
    mp.title("degree: " + str(degree))
    mp.show()
```

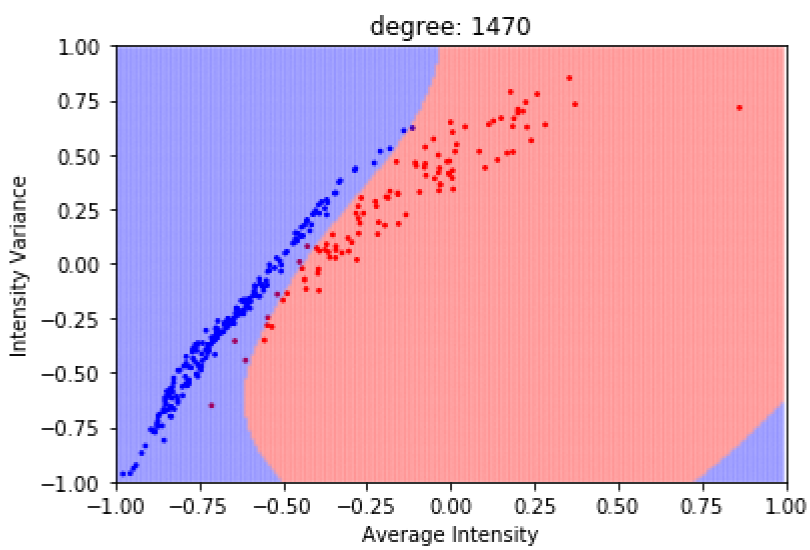

degree: 774
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
Intensity Variance
−1.00 −0.75 −0.50 −0.25 0.00 0.25 0.50 0.75 1.00
Average Intensity

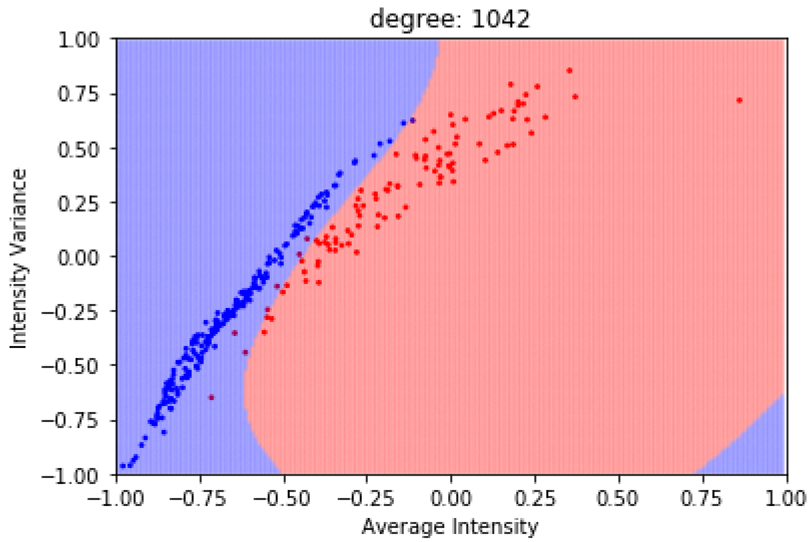

degree: 480
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
Intensity Variance
−1.00 −0.75 −0.50 −0.25 0.00 0.25 0.50 0.75 1.00
Average Intensity

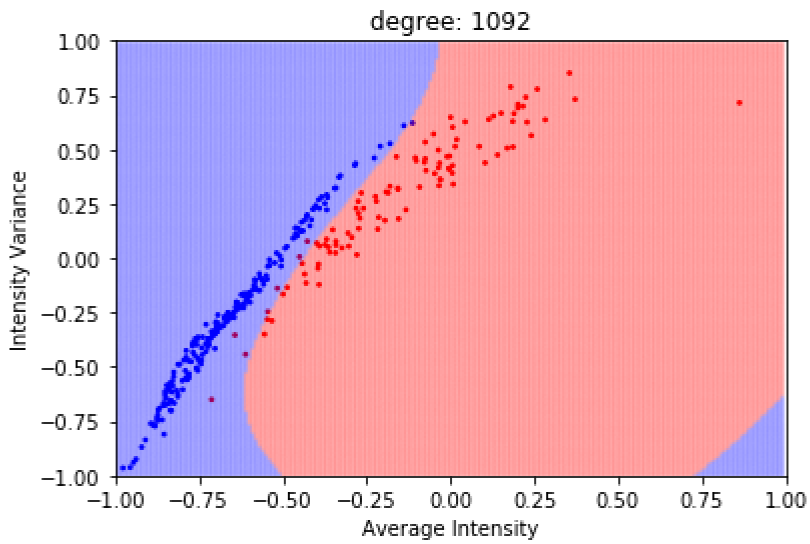

degree: 607
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
Intensity Variance
−1.00 −0.75 −0.50 −0.25 0.00 0.25 0.50 0.75 1.00
Average Intensity

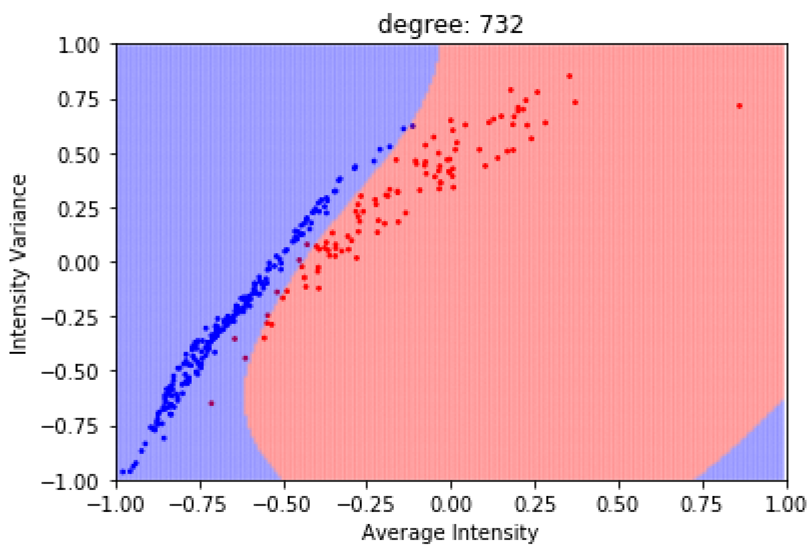

degree: 1442
Intensity Variance
Average Intensity
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.00 −0.75 −0.50 −0.25 0.00 0.25 0.50 0.75 1.00


degree: 200
Intensity Variance
Average Intensity
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.00 −0.75 −0.50 −0.25 0.00 0.25 0.50 0.75 1.00


degree: 1470
Intensity Variance
Average Intensity
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.00 −0.75 −0.50 −0.25 0.00 0.25 0.50 0.75 1.00







In [91]:
```
# As you can see in above plots, the parameter degree has no effect on the mod
el performance.
# This makes sense as degree parameter is only valid for the poly kernel
```

In [ ]:

In [ ]: