Homework 1

Question 1

In [30]:
```
import numpy as np
import matplotlib.pyplot as plt

nych = np.genfromtxt("data/nyc_housing.txt",delimiter=None) # load the text
Y = nych[:,-1] # target value (NYC borough) is the last column
X = nych[:,0:-1] # features are the other columns
print(X.shape)
```

```
(300, 3)
```

1a. 300 data points, 3 features

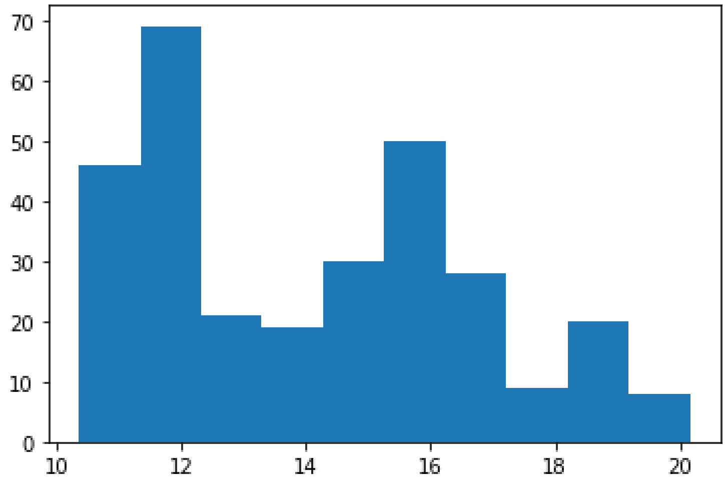
Feature 1

In [31]:
```
feature_1 = []
feature_2 = []
feature_3 = []
for x in X:
    for y in range(3):
        if y == 0:
            feature_1.append(x[0])
        elif y == 1:
            feature_2.append(x[1])
        elif y == 2:
            feature_3.append(x[2])
```

1b. Feature 1

In [32]:
```
plt.hist(feature_1)
```

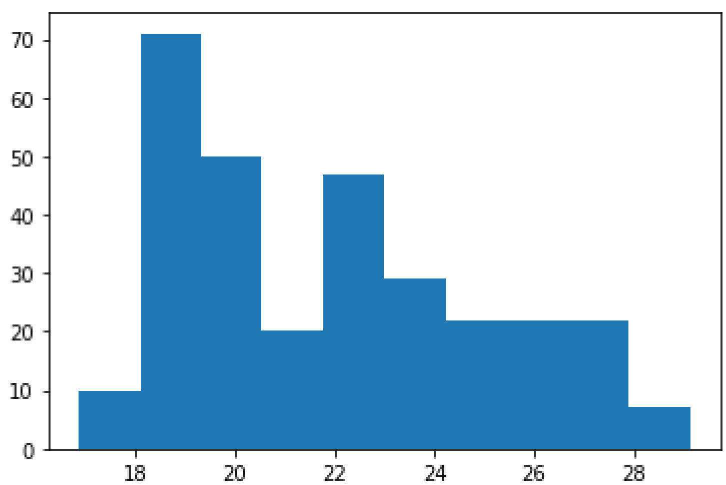
Out[32]:
```
(array([46., 69., 21., 19., 30., 50., 28.,  9., 20.,  8.]),
 array([10.36632221, 11.34496143, 12.32360065, 13.30223986, 14.28087908,
        15.2595183 , 16.23815751, 17.21679673, 18.19543594, 19.17407516,
        20.15271438]),
 <a list of 10 Patch objects>)
```




```
1b. Feature 2
```

In [33]:
```
plt.hist(feature_2)
```

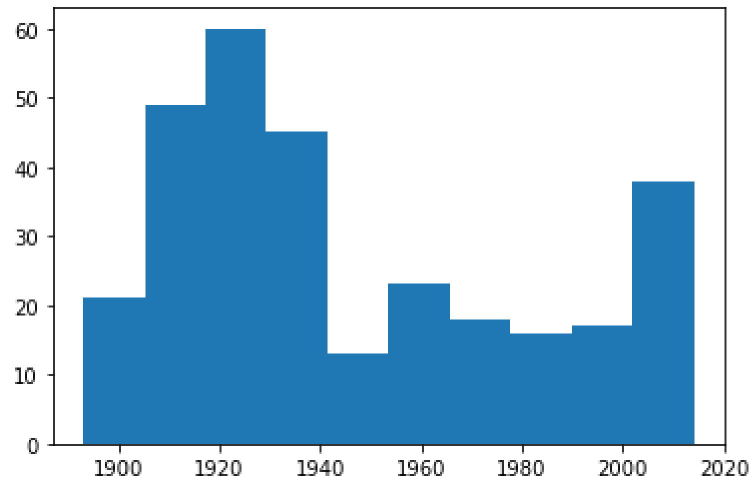
Out[33]:
```
(array([10., 71., 50., 20., 47., 29., 22., 22., 22.,  7.]),
 array([16.87267488, 18.09779353, 19.32291218, 20.54803083, 21.77314948,
        22.99826813, 24.22338678, 25.44850543, 26.67362408, 27.89874273,
        29.12386138]),
 <a list of 10 Patch objects>)
```




```
1b. Feature 3
```

In [34]:
```
plt.hist(feature_3)
```

Out[34]:
```
(array([21., 49., 60., 45., 13., 23., 18., 16., 17., 38.]),
 array([1893. , 1905.1, 1917.2, 1929.3, 1941.4, 1953.5, 1965.6, 1977.7,
        1989.8, 2001.9, 2014. ]),
 <a list of 10 Patch objects>)
```



1c. Feature 1 mean and standard deviation

In [35]:
```
np.mean(feature_1)
```

Out[35]: 14.118392438424483

In [36]:
```
np.std(feature_1)
```

Out[36]: 2.569090284260317

1c. Feature 2 mean and standard deviation

In [37]:
```
np.mean(feature_2)
```

Out[37]: 21.907116176170856

In [38]:
```
np.std(feature_2)
```

Out[38]: 2.9785784999947165

1c. Feature 3 mean and standard deviation

In [39]:
```
np.mean(feature_3)
```

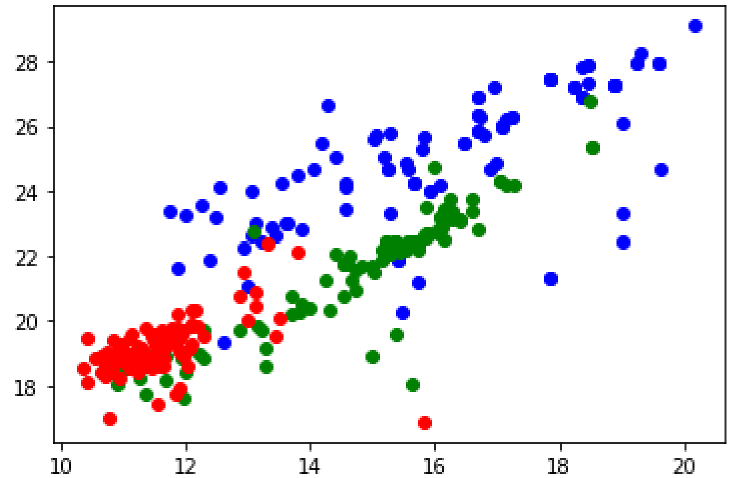
Out[39]: 1946.3533333333332

In [40]:
```
np.std(feature_3)
```

Out[40]: 35.39889577687731

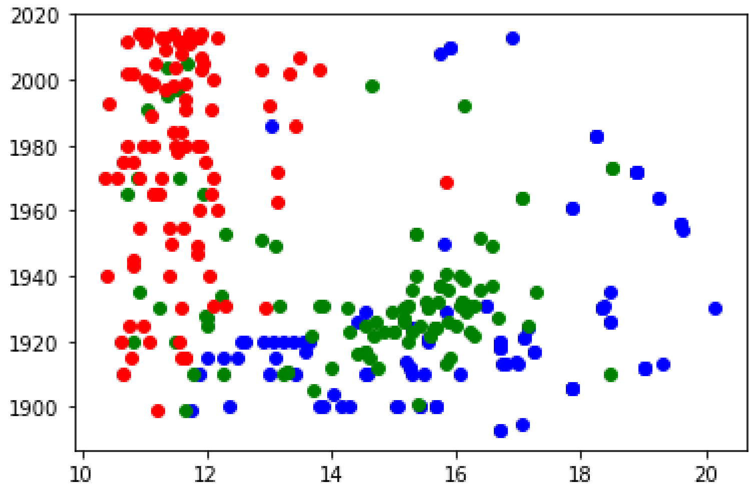
1d. (1, 2) scatter plot

In [41]:
```
colors= ['b','g','r']
for c in np.unique(Y):
    plt.plot( X[Y==c,0], X[Y==c,1], 'o',color=colors[int(c)] )
```



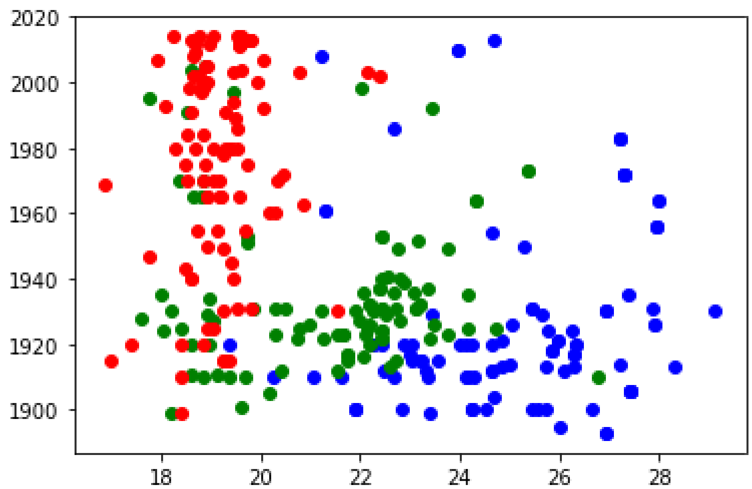
1d. (1, 3) scatter plot

In [42]:
```
for c in np.unique(Y):
    plt.plot( X[Y==c,0], X[Y==c,2], 'o',color=colors[int(c)] )
```



1d. (2, 3) scatter plot

In [43]:
```
for c in np.unique(Y):
    plt.plot( X[Y==c,1], X[Y==c,2], 'o',color=colors[int(c)] )
```

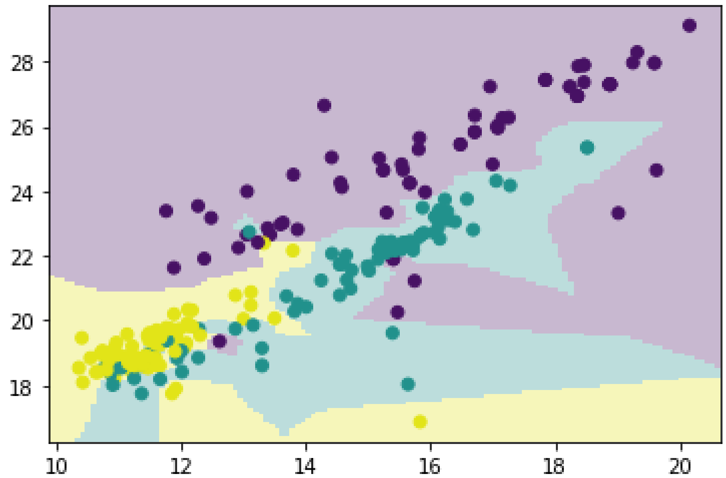


Question 2

In [44]:
```
nych = np.genfromtxt("data/nyc_housing.txt",delimiter=None) # load the data
Y = nych[:,-1]
X = nych[:,0:-1]
# Note: indexing with ":" indicates all values (in this case, all rows);
# indexing with a value ("0", "1", "-1", etc.) extracts only that value (he
# indexing rows/columns with a range ("1:-1") extracts any row/column in th

import mltools as ml
# We'll use some data manipulation routines in the provided class code
# Make sure the "mltools" directory is in a directory on your Python path,
# export PYTHONPATH=$\$${PYTHONPATH}:/path/to/parent/dir
# or add it to your path inside Python:
# import sys
# sys.path.append('/path/to/parent/dir/');

np.random.seed(0) # set the random number seed
X,Y = ml.shuffleData(X,Y) # shuffle data randomly
# (This is a good idea in case your data are ordered in some systematic way

Xtr,Xva,Ytr,Yva = ml.splitData(X,Y, 0.75) # split data into 75/25 train/val
```

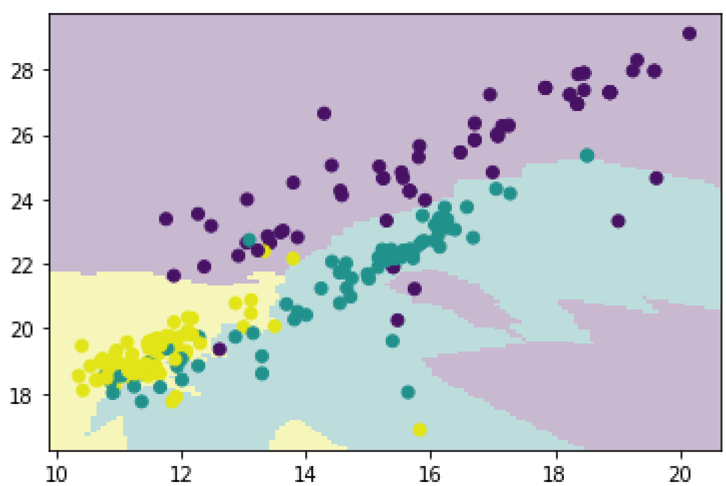
2a. K = 1

In [45]:
```
knn = ml.knn.knnClassify() # create the object and train it
knn.train(Xtr[:,0:2], Ytr, 1) # where K is an integer, e.g. 1 for nearest n
YvaHat = knn.predict(Xva[:,0:2]) # get estimates of y for each data point i

ml.plotClassify2D( knn, Xtr[:,0:2], Ytr ) # make 2D classification plot wit
```



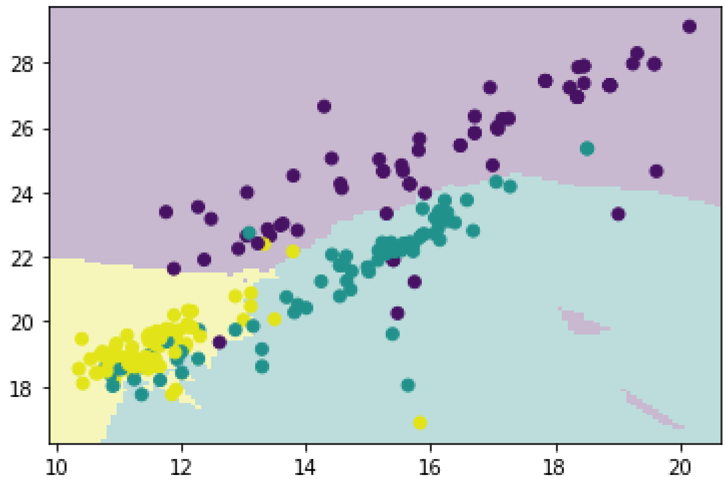
2a. K = 5

In [46]:
```
knn = ml.knn.knnClassify() # create the object and train it
knn.train(Xtr[:,0:2], Ytr, 5) # where K is an integer, e.g. 1 for nearest n
YvaHat = knn.predict(Xva[:,0:2]) # get estimates of y for each data point i

ml.plotClassify2D( knn, Xtr[:,0:2], Ytr ) # make 2D classification plot wit
```



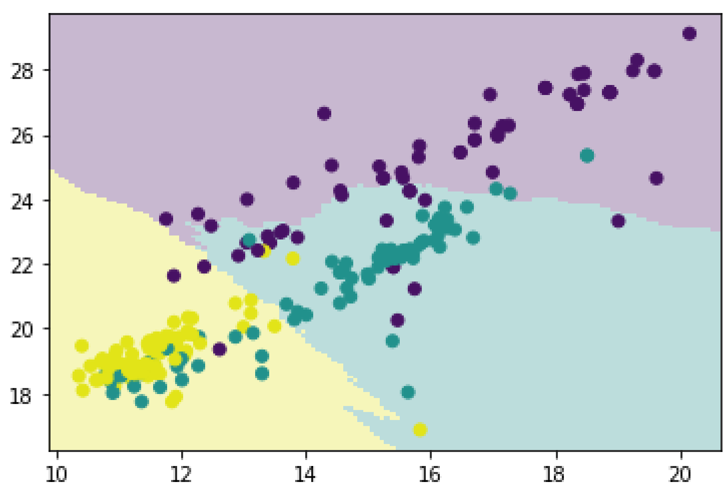
2a. K = 10

In [47]:
```
knn = ml.knn.knnClassify() # create the object and train it
knn.train(Xtr[:,0:2], Ytr, 10) # where K is an integer, e.g. 1 for nearest
YvaHat = knn.predict(Xva[:,0:2]) # get estimates of y for each data point i

ml.plotClassify2D( knn, Xtr[:,0:2], Ytr ) # make 2D classification plot wit
```



2a. K = 50

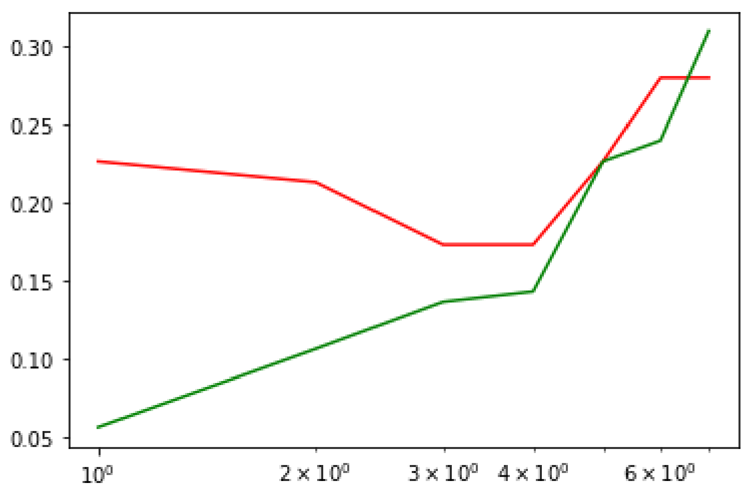
In [48]:
```
knn = ml.knn.knnClassify() # create the object and train it
knn.train(Xtr[:,0:2], Ytr, 50) # where K is an integer, e.g. 1 for nearest
YvaHat = knn.predict(Xva[:,0:2]) # get estimates of y for each data point i

ml.plotClassify2D( knn, Xtr[:,0:2], Ytr ) # make 2D classification plot wit
```



2b. For first two features: I would recommend k = 1.

In [49]:
```
K=[1,2,5,10,50,100,200];
errTrain = [None]*(len(K) + 1) # (preallocate storage for training error)
errValidation = [None]*(len(K) + 1)
for i,k in enumerate(K):
    learner = ml.knn.knnClassify(Xtr[:,0:2], Ytr, k) # TODO: complete code
    Yhat = learner.predict(Xva[:,0:2]) # TODO: predict results on training
    a = 0
    for j in range(len(Yhat)):
        if(Yhat[j] != Yva[j]):
            a += 1
    errTrain[i + 1] = a / len(Yhat)
    b = 0
    Yb = learner.predict(X[:,0:2]) # TODO: predict results on training data
    for l in range(len(Yb)):
        if(Yb[l] != Y[l]):
            b += 1
    errValidation[i + 1] = b / len(Yb)
# TODO: count what fraction of predictions are wrong
#TODO: repeat prediction / error evaluation for validation data
plt.semilogx(errTrain, color="r")
plt.semilogx(errValidation, color="g")
```

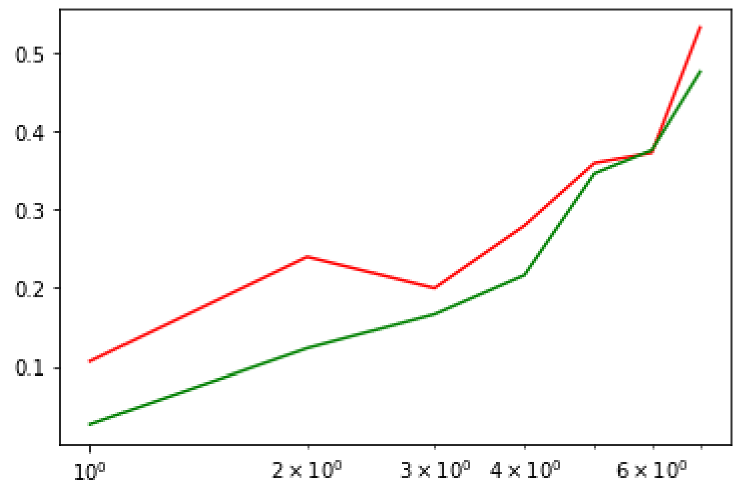
Out[49]:
```
[<matplotlib.lines.Line2D at 0x11c667280>]
```




```
2b. For all three features: I would recommend k = 1.
```

In [50]:
```
K=[1,2,5,10,50,100,200];
errTrain = [None]*(len(K) + 1) # (preallocate storage for training error)
errValidation = [None]*(len(K) + 1)
for i,k in enumerate(K):
    learner = ml.knn.knnClassify(Xtr[:,:], Ytr, k) # TODO: complete code to
    Yhat = learner.predict(Xva[:,:]) # TODO: predict results on training da
    a = 0
    for j in range(len(Yhat)):
        if(Yhat[j] != Yva[j]):
            a += 1
    errTrain[i + 1] = a / len(Yhat)
    b = 0
    Yb = learner.predict(X[:,:]) # TODO: predict results on training data
    for l in range(len(Yb)):
        if(Yb[l] != Y[l]):
            b += 1
    errValidation[i + 1] = b / len(Yb)
# TODO: count what fraction of predictions are wrong
#TODO: repeat prediction / error evaluation for validation data
plt.semilogx(errTrain, color="r")
plt.semilogx(errValidation, color="g")
```

Out[50]: [<matplotlib.lines.Line2D at 0x11c517700>]



3a.

```
p(y = 1) = 4/10 = 2/5
p(y = -1) = 6/10 = 3/5

p(x1 = 1| y = 1) = 3/4
p(x1 = 0| y = 1) = 1/4
```

```
p(x1 = 1| y = -1) = 3/6 = 1/2
p(x1 = 0| y = -1) = 3/6 = 1/2

p(x2 = 1| y = 1) = 0
p(x2 = 0| y = 1) = 1
p(x2 = 1| y = -1) = 5/6
p(x2 = 0| y = -1) = 1/6

p(x3 = 1| y = 1) = 3/4
p(x3 = 0| y = 1) = 1/4
p(x3 = 1| y = -1) = 4/6 = 2/3
p(x3 = 0| y = -1) = 2/6 = 1/2

p(x4 = 1| y = 1) = 2/4 = 1/2
p(x4 = 0| y = 1) = 2/4 = 1/2
p(x4 = 1| y = -1) = 4/6 = 5/6
p(x4 = 0| y = -1) = 2/6 = 1/6

p(x5 = 1| y = 1) = 1/4
p(x5 = 0| y = 1) = 3/4
p(x5 = 1| y = -1) = 2/6 = 1/3
p(x5 = 0| y = -1) = 4/6 = 2/3
```

3b. For x = 00000, y = 1 would be the expected class

p(x1 = 0, x2 = 0, x3 = 0, x4 = 0, x5 = 0 | y = 1) = p(x1 = 0|y = 1) * p(x2 = 0|y = 1) * p(x3 = 0|y = 1) * p(x4 = 0|y = 1) * p(x5 = 0|y = 1) = 1/4 * 1 * 1/4 * 1/2 * 3/4 = 3/128
p(x1 = 0, x2 = 0, x3 = 0, x4 = 0, x5 = 0 | y = -1) = p(x1 = 1|y = -1) * p(x2 = 1|y = -1) * p(x3 = 1|y = -1) * p(x4 = 1|y = -1) * p(x5 = 1|y = -1) = 1/2 * 1/6 * 1/2 * 1/6 * 2/3 = 1/216

3b. For x = 11010, y = -1 would be the expected class

p(x1 = 1, x2 = 1, x3 = 0, x4 = 1, x5 = 0 | y = 1) = p(x1 = 1|y = 1) * p(x2 = 1|y = 1) * p(x3 = 0|y = 1) * p(x4 = 1|y = 1) * p(x5 = 0|y = 1) = 3/4 * 0 * 1/4 * 1/2 * 3/4 = 0
p(x1 = 1, x2 = 1, x3 = 0, x4 = 1, x5 = 0 | y = -1) = p(x1 = 1|y = -1) * p(x2 = 1|y = -1) * p(x3 = 0|y = -1) * p(x4 = 1|y = -1) * p(x5 = 0|y = -1) = 1/2 * 5/6 * 1/2 * 5/6 * 2/3 = 50/432 = 25/216

3c. Posterior probability for x = 00000 and y = +1

p(x1 = 0, x2 = 0, x3 = 0, x4 = 0, x5 = 0 | y = 1) = p(x1 = 0|y = 1) * p(x2 = 0|y = 1) * p(x3 = 0|y = 1) * p(x4 = 0|y = 1) * p(x5 = 0|y = 1) = 1/4 * 1 * 1/4 * 1/2 * 3/4 = 3/128

3c. Posterior probability for x = 11010 and y = +1

p(x1 = 1, x2 = 1, x3 = 0, x4 = 1, x5 = 0 | y = 1) = p(x1 = 1|y = 1) * p(x2 = 1|y = 1) * p(x3 = 0|y = 1) * p(x4 = 1|y = 1) * p(x5 = 0|y = 1) = 3/4 * 0 * 1/4 * 1/2 * 3/4 = 0

3d. Disjoint bayes would have incomplete data since there would be no way to calculate the probabilities of other combinations without assuming conditional inpendence among the variable. Some combinations would all be zero by default due to not appearing in the data set.

3e. Since the variables are assumed to be conditionally independent in Naive Bayes, x1 would not have any effect on the probabilities of other variables. All that would happen is that x1 would not be used in the Naive Bayes Classifier. One would still calculate probabilities in the same way.

4a.

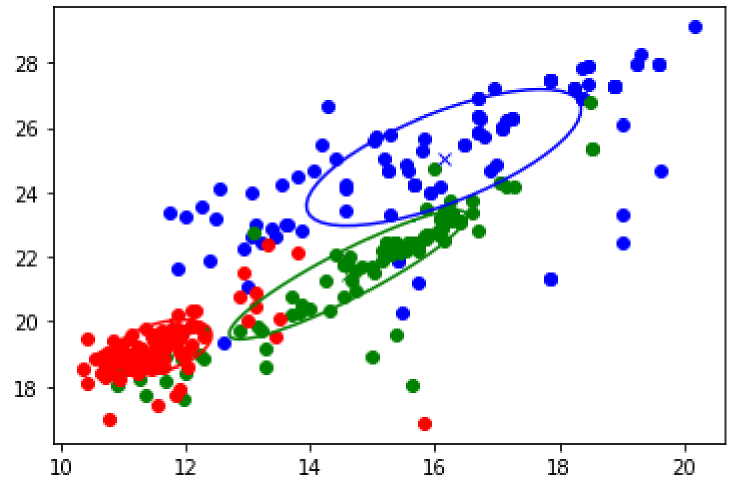
In [51]:
```
import numpy as np
nych = np.genfromtxt("data/nyc_housing.txt",delimiter=None) # load the data
Y = nych[:,-1] # target value (NYC borough) is the last column
X = nych[:,0:2]
class0 = []
class1 = []
class2 = []
for i in range(len(Y)):
    if(Y[i] == 0):
        class0.append(list(X[i, :]))
    elif(Y[i] == 1):
        class1.append(X[i, :])
    elif(Y[i] == 2):
        class2.append(X[i, :])

class0_feature0 = []
class0_feature1 = []
for x in class0:
    class0_feature0.append(x[0])
    class0_feature1.append(x[1])

class1_feature0 = []
class1_feature1 = []
for x in class1:
    class1_feature0.append(x[0])
    class1_feature1.append(x[1])

class2_feature0 = []
class2_feature1 = []
for x in class2:
    class2_feature0.append(x[0])
    class2_feature1.append(x[1])

mean0 = [np.mean(class0_feature0), np.mean(class0_feature1)]
mean1 = [np.mean(class1_feature0), np.mean(class1_feature1)]
mean2 = [np.mean(class2_feature0), np.mean(class2_feature1)]

print("Mean for class 0")
print(mean0)

print()

print("Mean for class 1")
print(mean1)

print()

print("Mean for class 2")
print(mean2)

print()

cov0 = np.cov(class0_feature0, class0_feature1)
cov1 = np.cov(class1_feature0, class1_feature1)
cov2 = np.cov(class2_feature0, class2_feature1)
```

```
print("Covariance matrix for class 0")
print(cov0)

print()

print("Covariance matrix for class 1")
print(cov1)

print()

print("Covariance matrix for class 2")
print(cov2)
```

```
Mean for class 0
[16.148986258099516, 25.07251957472242]

Mean for class 1
[14.608377678629202, 21.444688570537902]

Mean for class 2
[11.597813378544734, 19.204140383252252]

Covariance matrix for class 0
[[4.86296998 3.28966781]
 [3.28966781 4.44658827]]

Covariance matrix for class 1
[[3.63413268 3.47891043]
 [3.47891043 4.0026351 ]]

Covariance matrix for class 2
[[0.67861357 0.30523308]
 [0.30523308 0.71854788]]
```

4b.

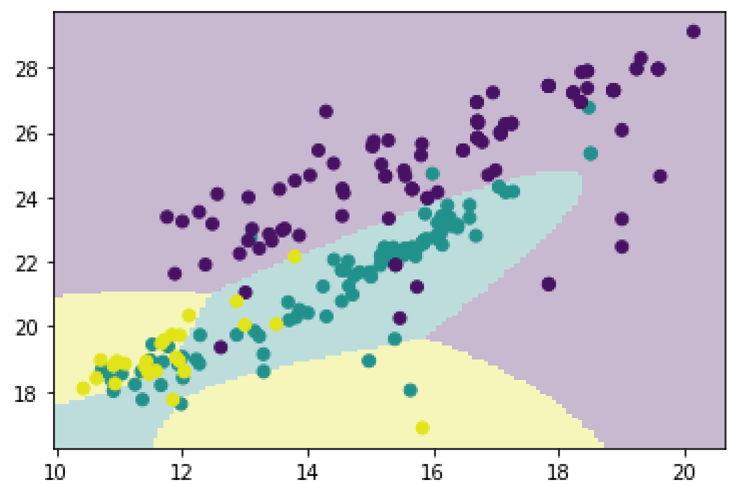
In [55]:
```
import matplotlib.pyplot as plt
import mltools as ml
colors= ['b','g','r']
for c in np.unique(Y):
    plt.plot( X[Y==c,0], X[Y==c,1], 'o',color=colors[int(c)] )
ml.plotGauss2D(mean0, cov0, color = "blue")
ml.plotGauss2D(mean1, cov1, color = "green")
ml.plotGauss2D(mean2, cov2, color = "red")
```



```
print(Xva)
```

4c.

In [58]:
```
Xtr,Xva,Ytr,Yva = ml.splitData(X,Y, 0.75) # split data into 75/25 train/val
bc = ml.bayes.gaussClassify( Xtr, Ytr );
ml.plotClassify2D(bc, Xtr, Ytr);
```



In [59]:
```
import mltools as ml
learner = ml.bayes.gaussClassify(Xtr, Ytr) # TODO: complete code to train m

Yhat = learner.predict(Xva) # TODO: predict results on training data
a = 0
for j in range(len(Yhat)):
    if(Yhat[j] != Yva[j]):
        a += 1
train = a / len(Yhat)

b = 0
Yb = learner.predict(X) # TODO: predict results on training data
for l in range(len(Yb)):
    if(Yb[l] != Y[l]):
        b += 1
validate = b / len(Yb)

print("training data error:", train)
print("validation data error:", validate)
```

```
training data error: 0.18666666666666668
validation data error: 0.15666666666666668
```

Statement of Collaboration: Discussed structure of mean and covariance vectors in question 4 with Luke Yi.