In [146]:

```
import numpy as np
import matplotlib.pyplot as plt
iris = np.genfromtxt("iris.txt",delimiter=None) # load the text file
Y = iris[:,-1] # target value (iris species) is the last column
X = iris[:,0:-1]             # features are the other columns
```

Problem 1.1:

In [148]:

```
print(X.shape)
```

```
(148, 4)
```

148 is the number of data points and 4 is the number of features.

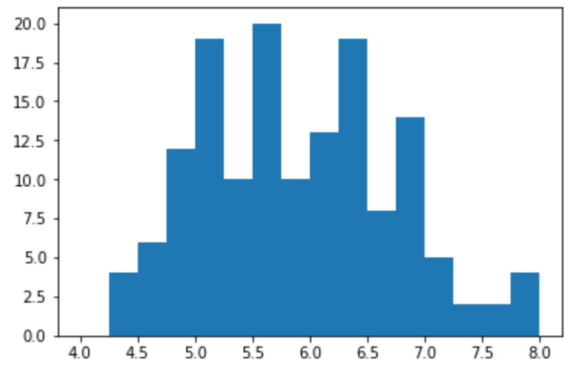
Problem 1.2:

In [153]:

```
X1 = X[:,0] # extract first feature
Bins = np.linspace(4,8,17) # use explicit bin locations
plt.hist( X1, bins=Bins ) # generate the plot"
```

Out[153]:

```
(array([ 0.,   4.,   6., 12., 19., 10., 20., 10., 13., 19.,   8., 14.,   5.,
          2.,   2.,   4.]),
 array([4.  , 4.25, 4.5 , 4.75, 5.  , 5.25, 5.5 , 5.75, 6.  , 6.25, 6.5 ,
        6.75, 7.  , 7.25, 7.5 , 7.75, 8.  ]),
 <a list of 16 Patch objects>)
```

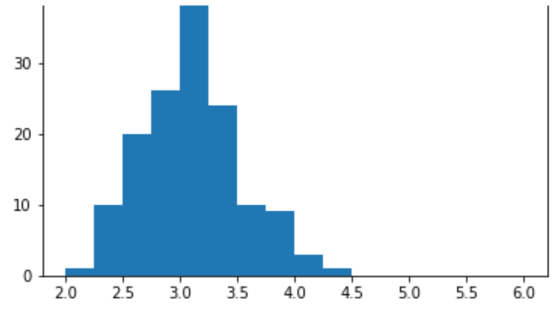


In [28]:

```
X1 = X[:,1] # extract first feature
Bins = np.linspace(2,6,17) # use explicit bin locations
plt.hist( X1, bins=Bins ) # generate the plot"
```

Out[28]:

```
(array([ 1., 10., 20., 26., 44., 24., 10.,  9.,  3.,  1.,  0.,  0.,  0.,
          0.,  0.,  0.]),
 array([2.  , 2.25, 2.5 , 2.75, 3.  , 3.25, 3.5 , 3.75, 4.  , 4.25, 4.5 ,
        4.75, 5.  , 5.25, 5.5 , 5.75, 6.  ]),
 <a list of 16 Patch objects>)
```

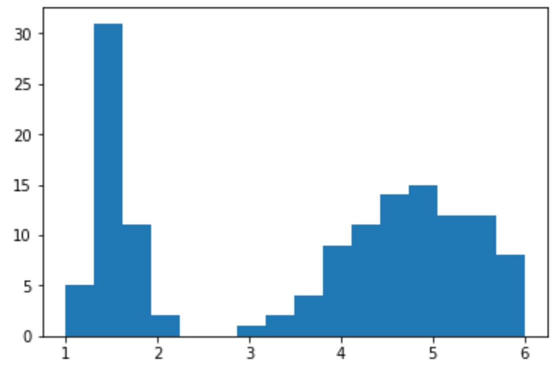


In [17]:

```
X1 = X[:,2] # extract first feature
Bins = np.linspace(1,6,17) # use explicit bin locations
plt.hist( X1, bins=Bins ) # generate the plot"
```

Out[17]:

```
(array([ 5., 31., 11.,  2.,  0.,  0.,  1.,  2.,  4.,  9., 11., 14., 15.,
        12., 12.,  8.]),
 array([1.    , 1.3125, 1.625 , 1.9375, 2.25  , 2.5625, 2.875 , 3.1875,
        3.5   , 3.8125, 4.125 , 4.4375, 4.75  , 5.0625, 5.375 , 5.6875,
        6.    ]),
 <a list of 16 Patch objects>)
```

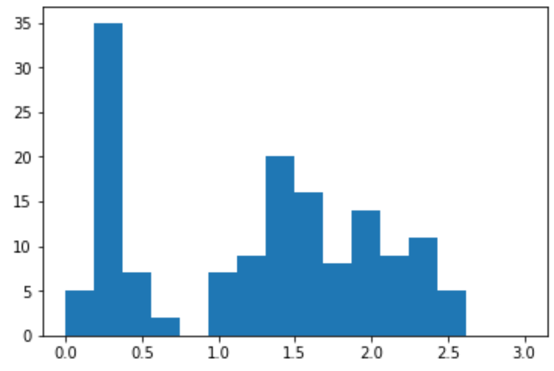


In [29]:

```
X1 = X[:,3] # extract first feature
Bins = np.linspace(0,3,17) # use explicit bin locations
plt.hist( X1, bins=Bins ) # generate the plot"
```

Out[29]:

```
(array([ 5., 35.,  7.,  2.,  0.,  7.,  9., 20., 16.,  8., 14.,  9., 11.,
         5.,  0.,  0.]),
 array([0.    , 0.1875, 0.375 , 0.5625, 0.75  , 0.9375, 1.125 , 1.3125,
        1.5   , 1.6875, 1.875 , 2.0625, 2.25  , 2.4375, 2.625 , 2.8125,
        3.    ]),
 <a list of 16 Patch objects>)
```



Problem 1.3:

In [225]:

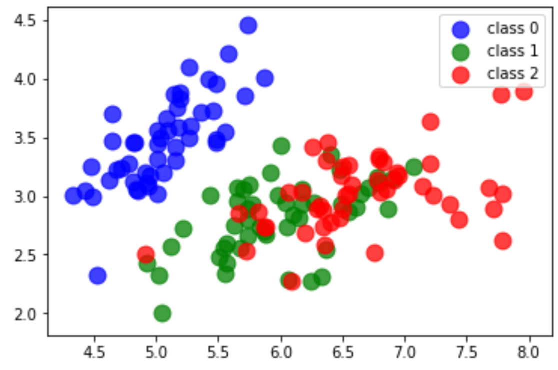
```
print("mean:               " + str(np.mean(X, axis = 0)))
print("standard deviation: " + str(np.std(X, axis = 0)))
```

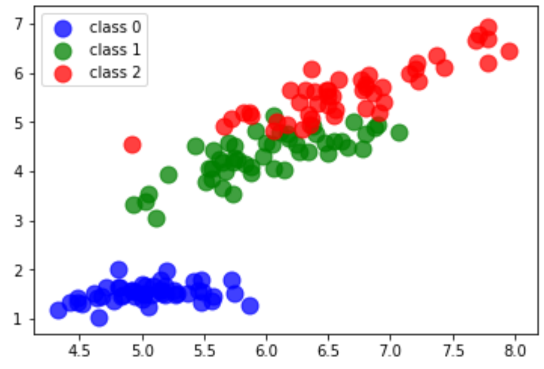
```
mean:               [5.90010376 3.09893092 3.81955484 1.25255548]
standard deviation: [0.83340207 0.43629184 1.75405711 0.75877246]
```

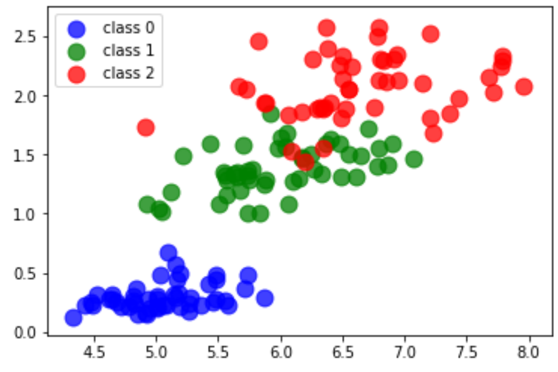
Problem 1.4:

In [226]:

```
colors = ['blue', 'green', 'red']
for x in range(1, 4):
    for i, c in enumerate(np.unique(iris[:, -1])):
        mask = np.where(iris[:, -1] == c)[0]    # Finding the right points
        plt.scatter(iris[mask, 0], iris[mask, x], s=120, c=colors[i], alpha=0.75, label='class %d'
% i)
    plt.legend()
    plt.show()
```







In [102]:

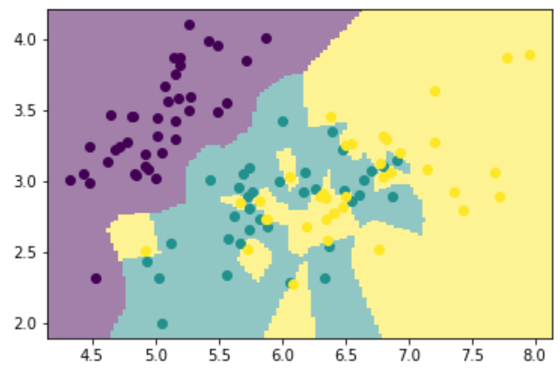
```
iris = np.genfromtxt("iris.txt",delimiter=None) # load the data
Y = iris[:,-1]
X = iris[:,0:2]
# Note: indexing with ":" indicates all values (in this case, all rows);
# indexing with a value ("0", "1", "-1", etc.) extracts only that value (here, columns); # indexin
g rows/columns with a range ("1:-1") extracts any row/column in that range.
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, e.g.,
# 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/validation
```

Problem 2.1

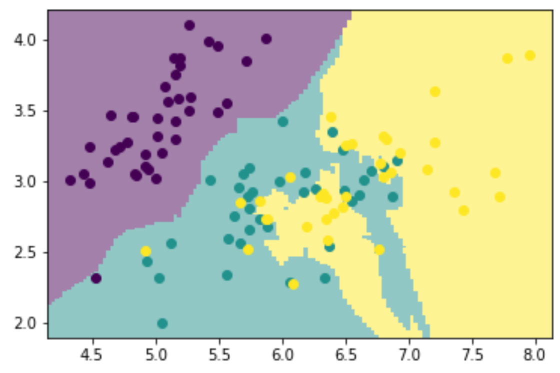
In [130]:

```
u = [1, 5, 10, 50]
for i in range(4):
    print("k = " +  str(u[i]))
    knn = ml.knn.knnClassify() # create the object and train it
    knn.train(Xtr, Ytr, u[i]) # where K is an integer, e.g. 1 for nearest neighbor prediction
   # YvaHat = knn.predict(Xva) # get estimates of y for each data point in Xva
    # Alternatively, the constructor provides a shortcut to "train":
    knn = ml.knn.knnClassify( Xtr, Ytr, u[i])
   # YvaHat = predict( knn, Xva );
    ml.plotClassify2D( knn, Xtr, Ytr ) # make 2D classification plot with data (Xtr,Ytr)
    plt.show()
```

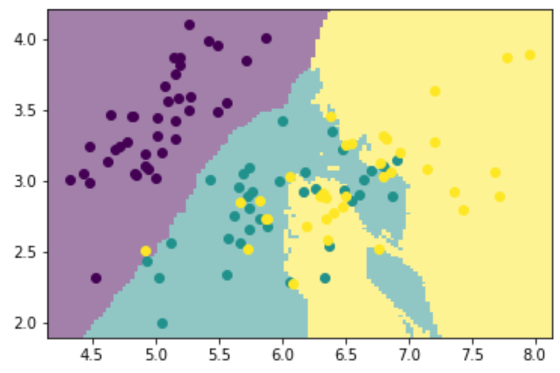
```
k = 1
```



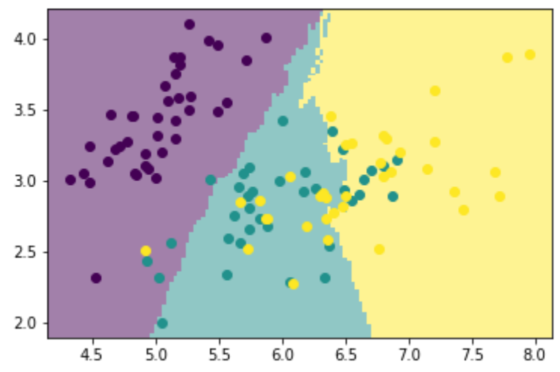
```
k = 5
```



```
k = 10
```



```
k = 50
```

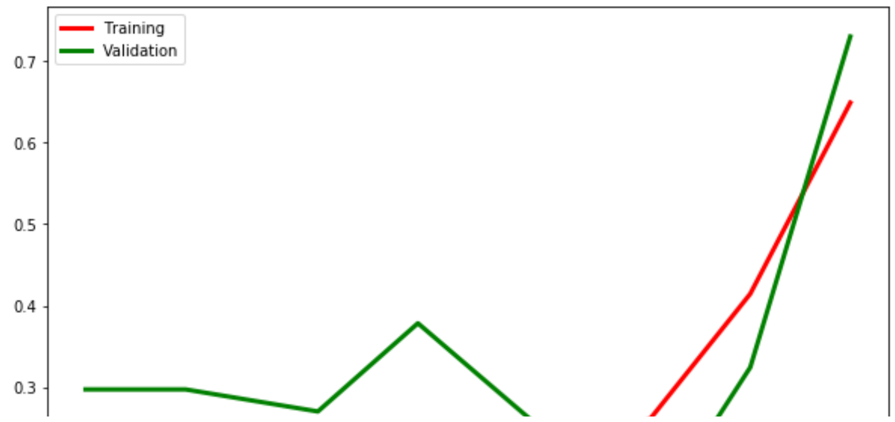


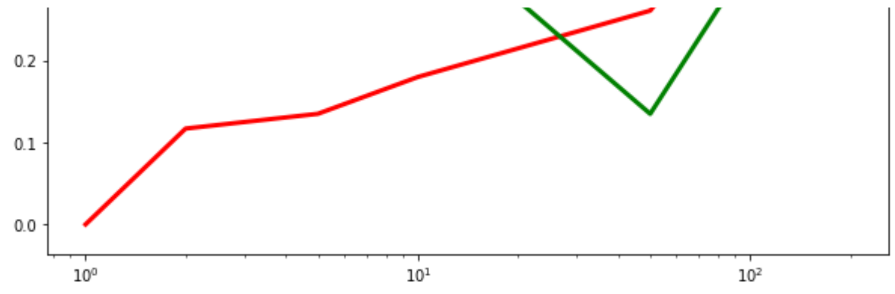
Problem 2.2:

In [184]:

```
K=[1,2,5,10,50,100,200];
errTrain = [None]*len(K) # (preallocate storage for training error)
errVal = [None]*len(K)
fig, ax = plt.subplots(1, 1, figsize=(10, 8))
for i,k in enumerate(K):
    learner = ml.knn.knnClassify(Xtr, Ytr) # TODO: complete code to train model
    Yhat = learner.predict(Xtr) # TODO: predict results on training data
    learner.train(Xtr, Ytr, k)
    errVal[i] = learner.err(Xva, Yva)
    errTrain[i] = learner.err(Xtr, Ytr)  # TODO: count what fraction of predictions are wrong

ax.semilogx(K, errTrain, 'r-', lw=3, label='Training')
ax.semilogx(K, errVal, 'g-', lw=3, label='Validation')
ax.legend()
#TODO: repeat prediction / error evaluation for validation data
plt.show()
```



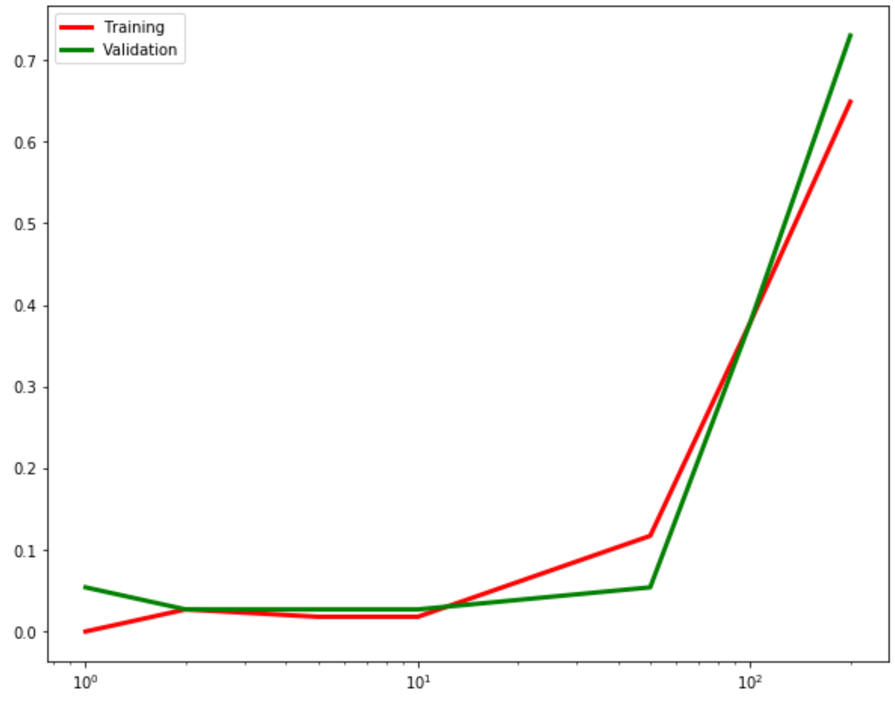


I think the K value I would recommend is 50 because it has the lowest validation error.

In [185]:

```
iris = np.genfromtxt("iris.txt",delimiter=None) # load the data
Y = iris[:,-1]
X = iris[:,:-1]
np.random.seed(0)
X,Y = ml.shuffleData(X,Y)
Xtr,Xva,Ytr,Yva = ml.splitData(X,Y, 0.75);
K=[1,2,5,10,50,100,200];
errTrain = [None]*len(K) # (preallocate storage for training error)
errVal = [None]*len(K)
fig, ax = plt.subplots(1, 1, figsize=(10, 8))
for i,k in enumerate(K):
    learner = ml.knn.knnClassify(Xtr, Ytr) # TODO: complete code to train model
    Yhat = learner.predict(Xtr) # TODO: predict results on training data
    learner.train(Xtr, Ytr, k)
    errVal[i] = learner.err(Xva, Yva)
    errTrain[i] = learner.err(Xtr, Ytr)  # TODO: count what fraction of predictions are wrong

ax.semilogx(K, errTrain, 'r-', lw=3, label='Training')
ax.semilogx(K, errVal, 'g-', lw=3, label='Validation')
ax.legend()
#TODO: repeat prediction / error evaluation for validation data
plt.show()
```



The plots do seem very different but they do both go high towards the end of the graph. It seems like the validation error is lowest at K = 2 so I think that may be the best K value, meaning my K value is different.

Problem 3.1:

In [207]:

```
#Not reading (-1):
x1y = 3/6
x2y = 5/6
x3y = 4/6
x4y = 5/6
x5y = 2/6

#Reading (+1):
x1y1 = 3/4
x2y1 = 0/4
x3y1 = 3/4
x4y1 = 2/4
x5y1 = 1/4

#p(y)
py = 4/10
```

Problem 3.2:

Class prediction for x = (00000):

In [212]:

```
zerosy0 = (1-x1y) * (1-x2y) * (1-x3y) * (1-x4y) * (1-x5y)
zerosy0 *= (1 - py)
print("y0 00000: 0.00185") #calculated by outside calculator because python wouldn't give the corr
ect result
zerosy1 = (1-x1y1) * (1-x2y1) * (1-x3y1) * (1-x4y1) * (1-x5y1)
zerosy1 *= (py)
print("y1 00000: 0.00938")
print("y1 > y0")
print("")
numsy0 = (x1y) * (x2y) * (1-x3y) * (x4y) * (1-x5y)
numsy0 *= (1 - py)
print("y0 11010: 0.046296")
numsy1 = (x1y1) * (x2y1) * (1-x3y1) * (x4y1) * (1-x5y1)
numsy1 *= (py)
print("y1 11010: 0")
print("y0 > y1")
```

```
y0 00000: 0.00185
y1 00000: 0.00938
y1 > y0

y0 11010: 0.046296
y1 11010: 0
y0 > y1
```

For 00000, I predict +1 (read). For 11010, I predict -1 (discard).

Problem 3.3:

In [221]:

```
#p(y1 | 11010) =   # calculations for this portion are done with a calculator and not python
print("p(y1 | 11010): ")
print("numsy1 / (numsy1 + numsy0) = 0")
print("")
#p(y1 | 00000) =
print("p(y1 | 00000): ")
print("zerosy1 / (zerosy1 + zerosy0) = 0.83526")
```

```
p(y1 | 11010):
numsy1 / (numsy1 + numsy0) = 0

p(y1 | 00000):
zerosy1 / (zerosy1 + zerosy0) = 0.83526
```

Problem 3.4:

We have a finite amount of data points and some of the data points may have no data associated with them. For example, x2y1 has a 0 probability. It would be an overfitting affect. There would be many co-observed variables.

Problem 3.5:

With the Naïve Bayes Model, covarient features are independent, it doesn't catch dependencies. Since each feature is taken into account independentaly, you wouldn't need to retrain the model because they weren't dependent on the feature that was taken out. If there are no longer a sufficient amount of parameters than the data set may not be enough to construct good probabilities.

Problem 4:

Statement of Collaboration: I did not discuss these topics with anyone but the TA who I asked a question about if I was using Jupyter properly and how I would need to save my work as a pdf and turn it in. I also checked Piazza for updates from the TA or professor on certain problems for clarification.