Creating a ROC curve without using a library

For this worksheet, we'll see how to generate a ROC curve without using a library. I know we would never do this in practice, but I think this will help people fix their bugs. I've done this worksheet myself, but then I've removed pieces of the code for you to fill in. I've used question marks in places where I've removed code.

First, we need a dataset and a prediction task. For this, we will use the iris dataset. We are going to use a naive bayes for this task. Here is some basic code.

In [1]:
```
print(__doc__)

import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets

# import some data to play with
iris = datasets.load_iris()
X = iris.data[:, :2]  # we only take the first two features. We could
                      # avoid this ugly slicing by using a two-dim dataset
y = iris.target

from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
model = gnb.fit(X,y)
```

```
Automatically created module for IPython interactive environment
```

So now that we have that base code what are some things we must consider. First, there are three classes in this dataset, so we have some options for the ROC. I'm going to show you how you can generate three ROC curves from this data where each curve assumes one of the classes to be positive.

In [109]:
```
np.unique(y)
```

Out[109]:
```
array([0, 1, 2])
```

In [110]:
```
pos_label = 1 # We are going to pick this one to start as it more challenging
 than the 0 class.
binary_y = y
binary_y[y != pos_label] = -1 # Use -1 for the negative class since we know we
 aren't using it from above.
```

In [111]:
```
np.unique(binary_y)
```

Out[111]:
```
array([-1,  1])
```

In [112]:
```
print 'Number of positive examples: ',np.sum(binary_y == pos_label)
print 'Number of negative examples: ',np.sum(binary_y != pos_label)
```

```
Number of positive examples:  50
Number of negative examples:  100
```

Now let's create our classifier and then let's extract some predictions.

In [113]:
```
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
y_pred = gnb.fit(X,binary_y)
probs = gnb.predict_proba(X)
```

In [114]:
```
print probs
```

```
[[  9.66141243e-01   3.38587573e-02]
 [  8.26121704e-01   1.73878296e-01]
 [  9.52534146e-01   4.74658538e-02]
 [  9.49853876e-01   5.01461237e-02]
 [  9.85338221e-01   1.46617787e-02]
 [  9.95623089e-01   4.37691074e-03]
 [  9.87119421e-01   1.28805792e-02]
 [  9.54762943e-01   4.52370572e-02]
 [  9.54465969e-01   4.55340314e-02]
 [  8.73661783e-01   1.26338217e-01]
 [  9.81963568e-01   1.80364317e-02]
 [  9.74427121e-01   2.55728793e-02]
 [  8.66142042e-01   1.33857958e-01]
 [  9.78471598e-01   2.15284019e-02]
 [  9.96933689e-01   3.06631107e-03]
 [  9.99920073e-01   7.99269445e-05]
 [  9.95623089e-01   4.37691074e-03]
 [  9.66141243e-01   3.38587573e-02]
 [  9.86999869e-01   1.30001313e-02]
 [  9.94994286e-01   5.00571359e-03]
 [  8.99524391e-01   1.00475609e-01]
 [  9.90046855e-01   9.95314495e-03]
 [  9.95918760e-01   4.08124015e-03]
 [  9.07499149e-01   9.25008510e-02]
 [  9.74427121e-01   2.55728793e-02]
 [  7.81853212e-01   2.18146788e-01]
 [  9.54762943e-01   4.52370572e-02]
 [  9.57840280e-01   4.21597205e-02]
 [  9.28691281e-01   7.13087194e-02]
 [  9.52534146e-01   4.74658538e-02]
 [  9.04011522e-01   9.59884775e-02]
 [  8.99524391e-01   1.00475609e-01]
 [  9.99396892e-01   6.03107975e-04]
 [  9.99575639e-01   4.24361033e-04]
 [  8.73661783e-01   1.26338217e-01]
 [  8.88198054e-01   1.11801946e-01]
 [  9.30958002e-01   6.90419977e-02]
 [  8.73661783e-01   1.26338217e-01]
 [  9.66842216e-01   3.31577840e-02]
 [  9.42386271e-01   5.76137290e-02]
 [  9.73558019e-01   2.64419809e-02]
 [  8.31497680e-01   1.68502320e-01]
 [  9.84763757e-01   1.52362432e-02]
 [  9.73558019e-01   2.64419809e-02]
 [  9.94994286e-01   5.00571359e-03]
 [  8.66142042e-01   1.33857958e-01]
 [  9.94994286e-01   5.00571359e-03]
 [  9.66495265e-01   3.35047348e-02]
 [  9.84798846e-01   1.52011543e-02]
 [  9.26788096e-01   7.32119043e-02]
 [  8.91269509e-01   1.08730491e-01]
 [  7.23235823e-01   2.76764177e-01]
 [  8.06141207e-01   1.93858793e-01]
 [  2.52066719e-01   7.47933281e-01]
 [  4.18920783e-01   5.81079217e-01]
 [  3.64544589e-01   6.35455411e-01]
 [  7.90910242e-01   2.09089758e-01]
```

```
[  5.63940042e-01   4.36059958e-01]
[  5.26879810e-01   4.73120190e-01]
[  4.84841642e-01   5.15158358e-01]
[  5.03342555e-01   4.96657445e-01]
[  4.88579209e-01   5.11420791e-01]
[  1.93043583e-01   8.06956417e-01]
[  4.07906630e-01   5.92093370e-01]
[  4.55620411e-01   5.44379589e-01]
[  7.29101646e-01   2.70898354e-01]
[  5.37946659e-01   4.62053341e-01]
[  2.97104275e-01   7.02895725e-01]
[  2.02098214e-01   7.97901786e-01]
[  2.60096776e-01   7.39903224e-01]
[  6.79238800e-01   3.20761200e-01]
[  3.41320890e-01   6.58679110e-01]
[  2.44347190e-01   7.55652810e-01]
[  3.41320890e-01   6.58679110e-01]
[  4.58728314e-01   5.41271686e-01]
[  6.07708184e-01   3.92291816e-01]
[  5.50800107e-01   4.49199893e-01]
[  6.49017935e-01   3.50982065e-01]
[  4.04273690e-01   5.95726310e-01]
[  2.71024187e-01   7.28975813e-01]
[  2.63234424e-01   7.36765576e-01]
[  2.63234424e-01   7.36765576e-01]
[  2.97104275e-01   7.02895725e-01]
[  2.86593701e-01   7.13406299e-01]
[  6.03220843e-01   3.96779157e-01]
[  8.47538506e-01   1.52461494e-01]
[  7.29101646e-01   2.70898354e-01]
[  2.15673329e-01   7.84326671e-01]
[  5.37946659e-01   4.62053341e-01]
[  2.83828158e-01   7.16171842e-01]
[  3.15053320e-01   6.84946680e-01]
[  4.89361549e-01   5.10638451e-01]
[  2.58197952e-01   7.41802048e-01]
[  4.79233288e-01   5.20766712e-01]
[  3.31306238e-01   6.68693762e-01]
[  5.14785723e-01   4.85214277e-01]
[  4.32684767e-01   5.67315233e-01]
[  4.18102596e-01   5.81897404e-01]
[  4.56128613e-01   5.43871387e-01]
[  3.64544589e-01   6.35455411e-01]
[  7.90910242e-01   2.09089758e-01]
[  2.97104275e-01   7.02895725e-01]
[  8.31013358e-01   1.68986642e-01]
[  4.34991809e-01   5.65008191e-01]
[  5.71420213e-01   4.28579787e-01]
[  9.68411132e-01   3.15888683e-02]
[  5.89244197e-01   4.10755803e-01]
[  8.71443673e-01   1.28556327e-01]
[  3.58284093e-01   6.41715907e-01]
[  9.92124425e-01   7.87557460e-03]
[  7.47177457e-01   2.52822543e-01]
[  3.34086569e-01   6.65913431e-01]
[  6.93972458e-01   3.06027542e-01]
[  2.42617083e-01   7.57382917e-01]
```

```
[  3.49413906e-01   6.50586094e-01]
[  7.23235823e-01   2.76764177e-01]
[  5.71420213e-01   4.28579787e-01]
[  9.99708563e-01   2.91437360e-04]
[  9.43813446e-01   5.61865542e-02]
[  1.93043583e-01   8.06956417e-01]
[  8.63627519e-01   1.36372481e-01]
[  3.86327932e-01   6.13672068e-01]
[  9.62852025e-01   3.71479748e-02]
[  3.13069193e-01   6.86930807e-01]
[  8.67219336e-01   1.32780664e-01]
[  9.37074260e-01   6.29257403e-02]
[  3.50838693e-01   6.49161307e-01]
[  4.89361549e-01   5.10638451e-01]
[  3.89301187e-01   6.10698813e-01]
[  8.70438370e-01   1.29561630e-01]
[  8.80271304e-01   1.19728696e-01]
[  9.99890049e-01   1.09951188e-04]
[  3.89301187e-01   6.10698813e-01]
[  3.66723169e-01   6.33276831e-01]
[  2.51401250e-01   7.48598750e-01]
[  9.79564684e-01   2.04353163e-02]
[  8.63137878e-01   1.36862122e-01]
[  6.31800054e-01   3.68199946e-01]
[  4.85598126e-01   5.14401874e-01]
[  8.06141207e-01   1.93858793e-01]
[  7.29101646e-01   2.70898354e-01]
[  8.06141207e-01   1.93858793e-01]
[  2.97104275e-01   7.02895725e-01]
[  8.34066235e-01   1.65933765e-01]
[  8.67219336e-01   1.32780664e-01]
[  6.49017935e-01   3.50982065e-01]
[  2.44347190e-01   7.55652810e-01]
[  5.71420213e-01   4.28579787e-01]
[  8.54774025e-01   1.45225975e-01]
[  4.88579209e-01   5.11420791e-01]]
```

In [115]:
```
print gnb.classes_ # print the class labels as seen by the classifier
```

```
[-1  1]
```

In [116]:
```
# This means that the first column is the negative class. We can honestly use
 either column, but we need to know what column is
# what. I will use the positive column always.
pos_probs = np.squeeze(probs[:,1])
```

In [117]:
```
pos_probs
```

Out[117]:
```
array([  3.38587573e-02,   1.73878296e-01,   4.74658538e-02,
         5.01461237e-02,   1.46617787e-02,   4.37691074e-03,
         1.28805792e-02,   4.52370572e-02,   4.55340314e-02,
         1.26338217e-01,   1.80364317e-02,   2.55728793e-02,
         1.33857958e-01,   2.15284019e-02,   3.06631107e-03,
         7.99269445e-05,   4.37691074e-03,   3.38587573e-02,
         1.30001313e-02,   5.00571359e-03,   1.00475609e-01,
         9.95314495e-03,   4.08124015e-03,   9.25008510e-02,
         2.55728793e-02,   2.18146788e-01,   4.52370572e-02,
         4.21597205e-02,   7.13087194e-02,   4.74658538e-02,
         9.59884775e-02,   1.00475609e-01,   6.03107975e-04,
         4.24361033e-04,   1.26338217e-01,   1.11801946e-01,
         6.90419977e-02,   1.26338217e-01,   3.31577840e-02,
         5.76137290e-02,   2.64419809e-02,   1.68502320e-01,
         1.52362432e-02,   2.64419809e-02,   5.00571359e-03,
         1.33857958e-01,   5.00571359e-03,   3.35047348e-02,
         1.52011543e-02,   7.32119043e-02,   1.08730491e-01,
         2.76764177e-01,   1.93858793e-01,   7.47933281e-01,
         5.81079217e-01,   6.35455411e-01,   2.09089758e-01,
         4.36059958e-01,   4.73120190e-01,   5.15158358e-01,
         4.96657445e-01,   5.11420791e-01,   8.06956417e-01,
         5.92093370e-01,   5.44379589e-01,   2.70898354e-01,
         4.62053341e-01,   7.02895725e-01,   7.97901786e-01,
         7.39903224e-01,   3.20761200e-01,   6.58679110e-01,
         7.55652810e-01,   6.58679110e-01,   5.41271686e-01,
         3.92291816e-01,   4.49199893e-01,   3.50982065e-01,
         5.95726310e-01,   7.28975813e-01,   7.36765576e-01,
         7.36765576e-01,   7.02895725e-01,   7.13406299e-01,
         3.96779157e-01,   1.52461494e-01,   2.70898354e-01,
         7.84326671e-01,   4.62053341e-01,   7.16171842e-01,
         6.84946680e-01,   5.10638451e-01,   7.41802048e-01,
         5.20766712e-01,   6.68693762e-01,   4.85214277e-01,
         5.67315233e-01,   5.81897404e-01,   5.43871387e-01,
         6.35455411e-01,   2.09089758e-01,   7.02895725e-01,
         1.68986642e-01,   5.65008191e-01,   4.28579787e-01,
         3.15888683e-02,   4.10755803e-01,   1.28556327e-01,
         6.41715907e-01,   7.87557460e-03,   2.52822543e-01,
         6.65913431e-01,   3.06027542e-01,   7.57382917e-01,
         6.50586094e-01,   2.76764177e-01,   4.28579787e-01,
         2.91437360e-04,   5.61865542e-02,   8.06956417e-01,
         1.36372481e-01,   6.13672068e-01,   3.71479748e-02,
         6.86930807e-01,   1.32780664e-01,   6.29257403e-02,
         6.49161307e-01,   5.10638451e-01,   6.10698813e-01,
         1.29561630e-01,   1.19728696e-01,   1.09951188e-04,
         6.10698813e-01,   6.33276831e-01,   7.48598750e-01,
         2.04353163e-02,   1.36862122e-01,   3.68199946e-01,
         5.14401874e-01,   1.93858793e-01,   2.70898354e-01,
         1.93858793e-01,   7.02895725e-01,   1.65933765e-01,
         1.32780664e-01,   3.50982065e-01,   7.55652810e-01,
         4.28579787e-01,   1.45225975e-01,   5.11420791e-01])
```

Now we have to ask ourselves the following, how many different cutoffs are there for the probabilities? Answer: number of unique probabilities - 1.

In [118]:
```
np.unique(pos_probs).shape
```

Out[118]: (116,)

Now how do we go from this to our y and x axis. The y-axis is the true positive rate and the x-axis is the false positive rate.

In [119]:
```
# For starters, assume 0.5 is our cutoff
cutoff = 0.5
pos_inxs = np.where(binary_y == pos_label)
neg_inxs = np.where(binary_y != pos_label)
tpr = np.size(np.where(pos_probs[pos_inxs] >= cutoff))*1./np.size(pos_inxs) #
 true positive rate for the above cutoff
fpr = np.size(np.where(pos_probs[neg_inxs] < cutoff))*1./np.size(neg_inxs) # f
alse negative rate for the above cutoff
print tpr,fpr
```

```
0.64 0.81
```

In [120]:
```
# Let try another cutoff
cutoff = 0.1
pos_inxs = np.where(binary_y == pos_label)
neg_inxs = np.where(binary_y != pos_label)
tpr = np.size(np.where(pos_probs[pos_inxs] >= cutoff))*1.0/np.size(pos_inxs) #
 true positive rate for the above cutoff
fpr = np.size(np.where(pos_probs[neg_inxs] >= cutoff))*1.0/np.size(neg_inxs) #
 false negative rate for the above cutoff
print tpr,fpr
```

```
1.0 0.53
```

In [121]:
```
# You get the idea. We need to use a bunch of cutoffs.
tprs = []
fprs = []
# Here is the trickiest part, we want all the cutoffs that are relevant. How d
o you get that and
# We must include 0 and 1.0 for a full graph.
cutoffs = []
cutoffs = np.sort(pos_probs).tolist()
cutoffs.insert(0,0)
cutoffs.append(1.0)
for cutoff in cutoffs:
    pos_inxs = np.where(binary_y == pos_label)
    neg_inxs = np.where(binary_y != pos_label)
    tpr = np.size(np.where(pos_probs[pos_inxs] >= cutoff))*1.0/np.size(pos_inx
s) # true positive rate for the above cutoff
    fpr = np.size(np.where(pos_probs[neg_inxs] >= cutoff))*1.0/np.size(neg_inx
s) # false negative rate for the above cutoff
    tprs.append(tpr)
    fprs.append(fpr)
```

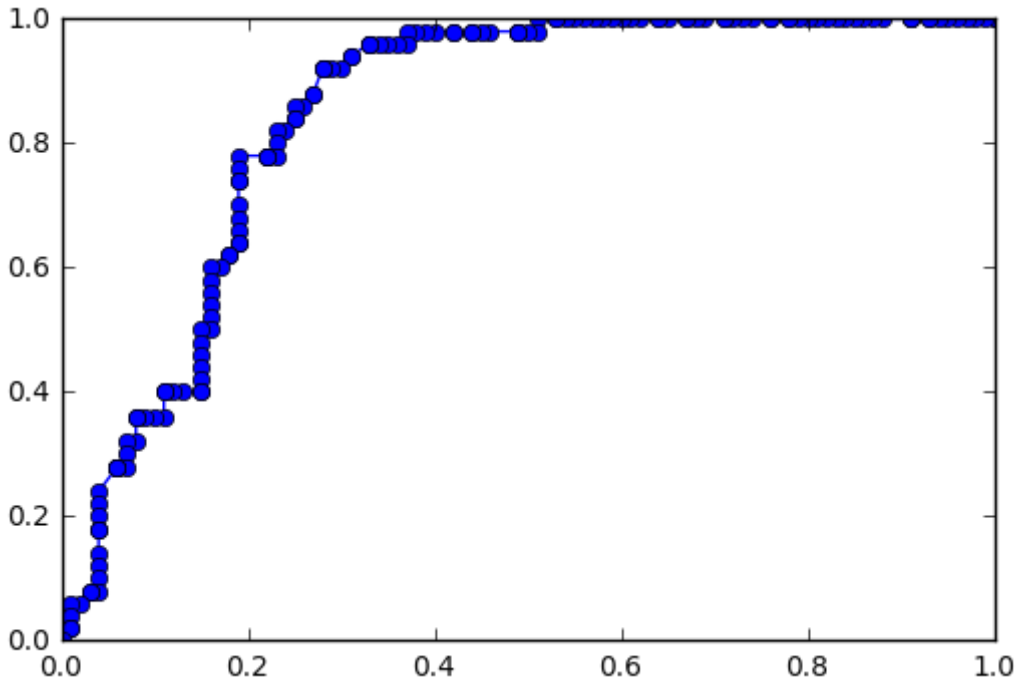
In [ ]:
```
```

In [122]:
```
%matplotlib inline

import matplotlib.pyplot as plt

plt.plot(fprs,tprs,marker='o')
```

Out[122]:
```
[<matplotlib.lines.Line2D at 0x7f01f79c1350>]
```



So you can see all the the different points, but why is this so awesome in terms of results? Answer: We haven't done any CV!!! To keep life as simple as possible, we are going to do leave one out. Specifically, we need to create the pos_probs array again but with leaving a sample out.

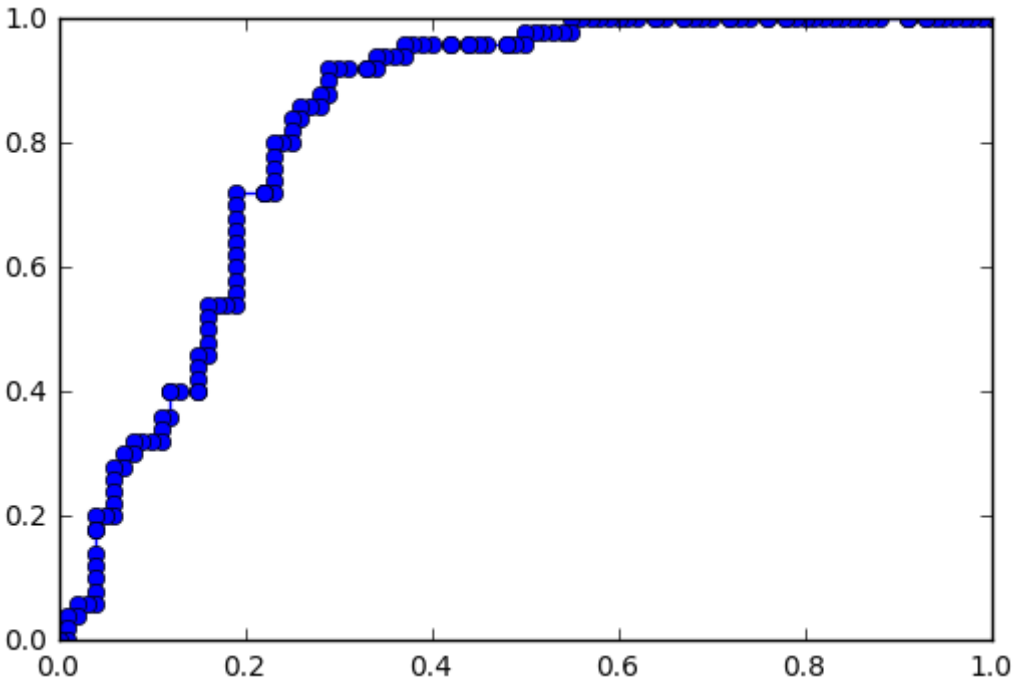
In [123]:
```
pos_probs = []
for i in range(np.size(binary_y)):
    gnb = GaussianNB()
    Xtrain = np.delete(X,i,axis=0)
    ytrain = np.delete(binary_y,i)
    Xtest = np.zeros((1,2))
    Xtest[0,:] = X[i,:]
    ytest = binary_y[i]
    gnb.fit(Xtrain,ytrain)
    probs = gnb.predict_proba(Xtest)
    pos_prob_test = np.squeeze(probs[:,1])
    pos_probs.append(pos_prob_test)
pos_probs = np.array(pos_probs)
```

In [124]:
```
# You get the idea. We need to use a bunch of cutoffs.
tprs = []
fprs = []
# Fill in your working code from above to now calculate tprs and fprs
cutoffs = []
cutoffs = np.sort(pos_probs).tolist()
cutoffs.insert(0,0)
cutoffs.append(1.0)
for cutoff in cutoffs:
    pos_inxs = np.where(binary_y == pos_label)
    neg_inxs = np.where(binary_y != pos_label)
    tpr = np.size(np.where(pos_probs[pos_inxs] >= cutoff))*1.0/np.size(pos_inx
s) # true positive rate for the above cutoff
    fpr = np.size(np.where(pos_probs[neg_inxs] >= cutoff))*1.0/np.size(neg_inx
s) # false negative rate for the above cutoff
    tprs.append(tpr)
    fprs.append(fpr)
```

In [125]:
```
import matplotlib.pyplot as plt

plt.plot(fprs,tprs,marker='o')
```

Out[125]:
```
[<matplotlib.lines.Line2D at 0x7f01f796f310>]
```



In [ ]: