ROC Curve

Alper.Halbutogullari@gmail.com (mailto:Alper.Halbutogullari@gmail.com)

This exercise will introduce you to the ROC (Receiver Operating Characteristic) Curve and the AUC (Area Under the (ROC) Curve). We will first use the existing packages to generate them and later construct it ourselves from scratch.

Here are some good visual introductions to the meaning of the curve:

https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5 (https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5)

https://paulvanderlaken.com/2019/08/16/roc-auc-precision-and-recall-visually-explained/ (https://paulvanderlaken.com/2019/08/16/roc-auc-precision-and-recall-visually-explained/)

In [1]:
```
from IPython.display import IFrame

url = "https://paulvanderlaken.com/2019/08/16/roc-auc-precision-and-rec
IFrame(url, width=700, height=350)
```

Out[1]:

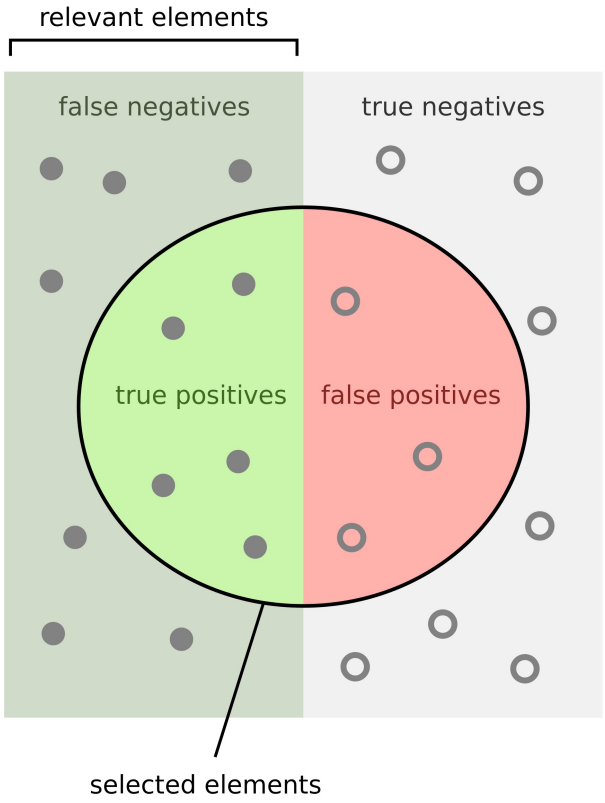
The ROC curve is plotted with TPR (True Positive Rate) against the FPR (False Positive Rate) where TPR is on y-axis and FPR is on the x-axis. Here are the definitions (TP: True Positives, FP: False Positives, TN: True Negatives, FN: False Negatives):

$$\text{TPR /Recall / Sensitivity} = \frac{TP}{TP + FN}$$

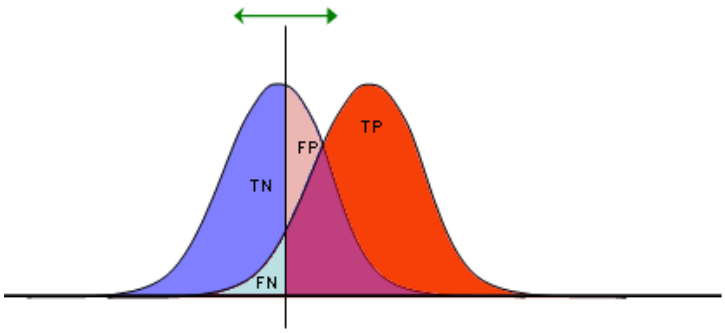
$$\text{Specificity} = \frac{TN}{TN + FP}$$

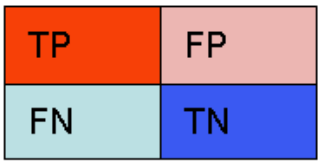
$$\text{FPR} = 1 - \text{Specificity} = \frac{FP}{TN + FP}$$

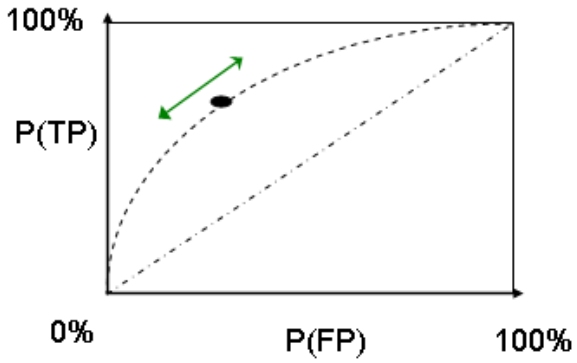
Here is a good picture to understand TP, FP, TN and FN:



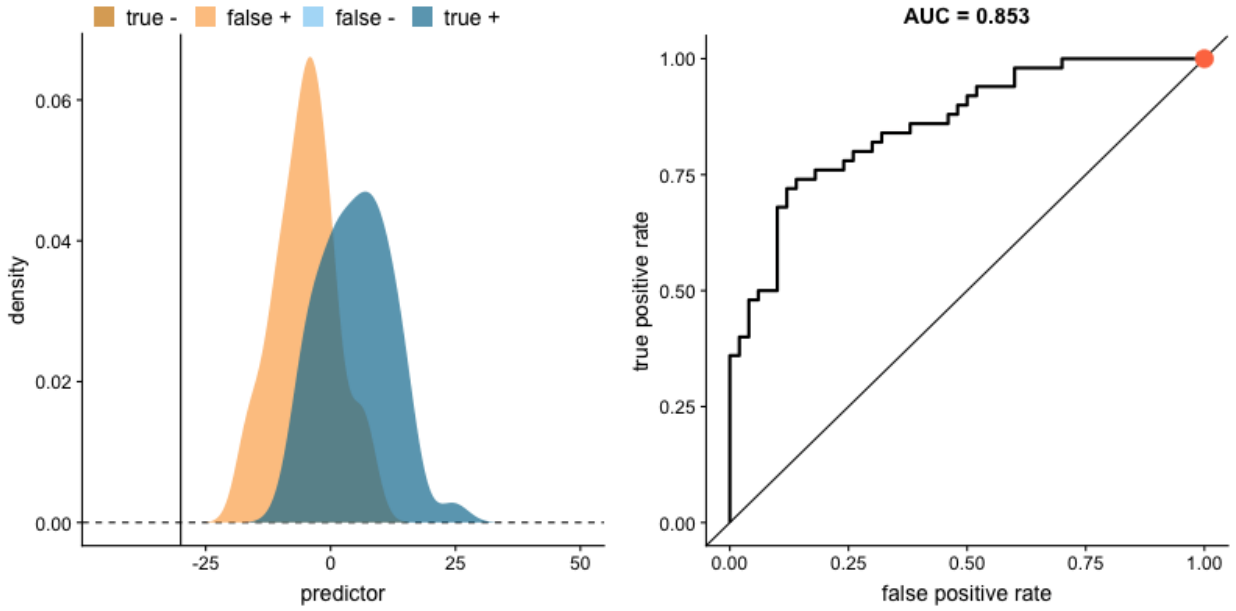
For a given predictor, the ROC curve is generated using a subset of all possible cutoffs as follows (each point corresponds to a cutoff point):



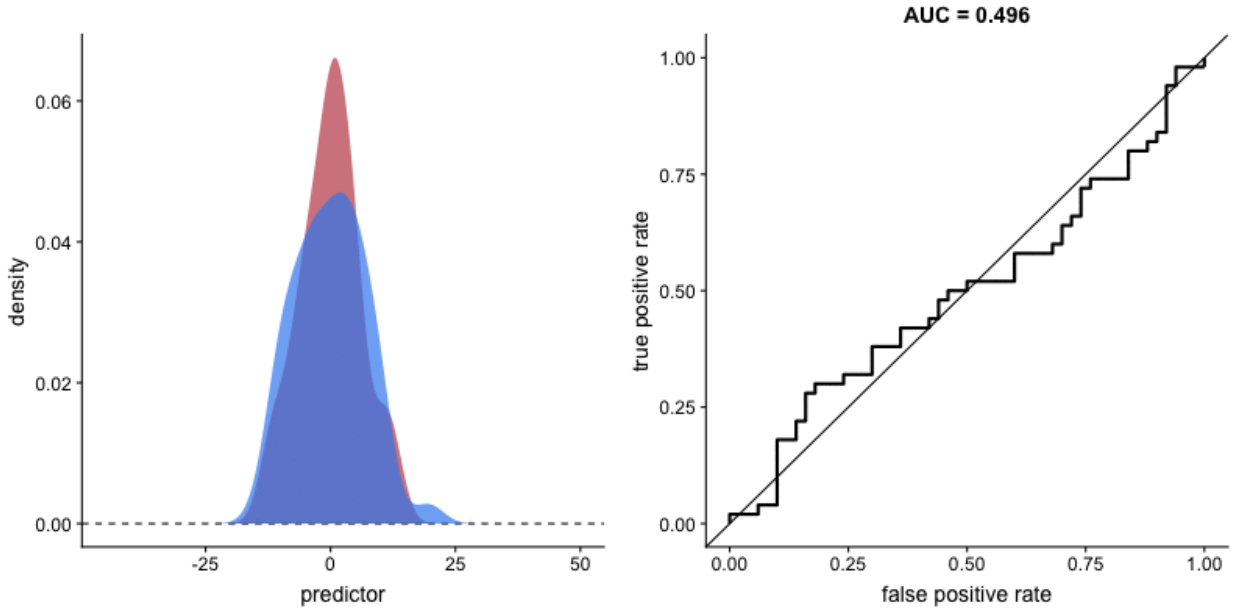




Here is an animated picture of the same thing:



But the trick is to find good predictors so that the distributions of the two cases are separated as much as possible:



As you can see the AUC measure of a ROC curve for nicely separated distributions gets close to one.

Let's see it for a specific data shown below. In reality, there are 10 positive and 10 negative cases in this dataset (e.g. cancer and non-cancer patients). A model assigned a probability score for each input record. As an example, 0.87 means that a specific input record is 87 percent likely to be positive. The probabilities are relative scores, so one has to come up with a cut-off point for the decision boundary (this also depends on the relative importance of making mistakes on positive and negative cases).

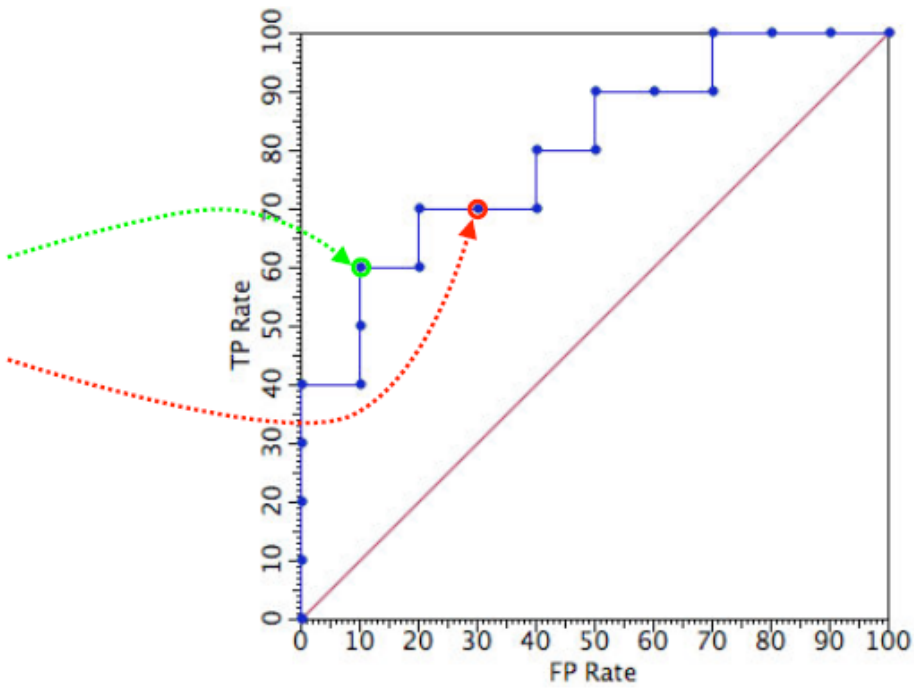
First, you sort the probabilities generated by the model and then select the cut-off point for the decision boundary. Everything above the threshold (i.e. higher probabilities) is assigned to the positive class and everything below is assigned to the negative class.

In the following example, if you choose the cut-off point to be the green line (any value between 0.62 and 0.67 works, but let's say we pick the cut-off to be 0.65), then you predict that 7 cases above this line are positive and 13 cases below it are negative. 6 of the above are predicted correctly as (true) positives (TP) and one of them was incorrectly predicted as negative (FP). Similarly, by looking at the observations below the green cut-off line, you predict 9 of them correctly as (true) negative (TN), but also 4 actual positives are incorrectly predicted as negative (FN).

In summary: TP=6, FP=1, TN = 9, FN=4 for this specific cut-off point. This corresponds to TPR=60% and FPR=10%, which is shown on the ROC curve as the green dot.

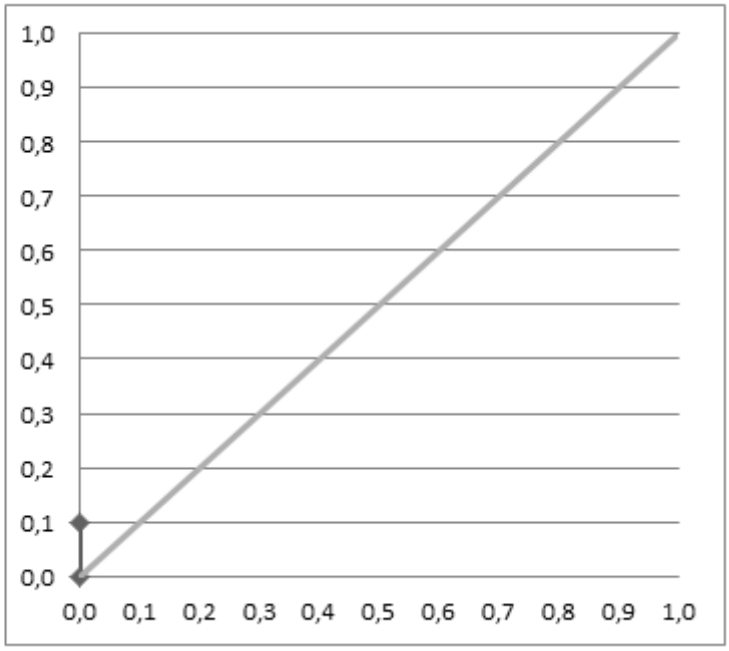
If you choose the cut-off point to be the red line (let's say the cut-off is 0.50), then TPR=70% and FPR=30%, which is shown on the ROC curve as the red dot. As you see, each cut-off point corresponds to a point on the ROC curve.

Class	Score
+	0.98
+	0.93
+	0.87
+	0.84
–	0.79
+	0.73
+	0.67
–	0.62
+	0.57
–	0.54
–	0.48
+	0.43
–	0.37
+	0.34
–	0.28
–	0.24
+	0.18
–	0.12
–	0.09
–	0.03



Let's assume we move the cut-off point, i.e. our decision boundary downwards. When the decision boundary moves below a data point, we start to assign the positive class to that point (which was assigned to the negative class before), so we change the decision for a single data point. If that point originally belonged to the positive class, then we are doing the right thing and we increased the accuracy of our model; if not, then we started to make one more mistake. This corresponds to moving up (i.e. increasing the TPR, True Positive Rate), or moving left, (i.e. increasing the FPR, False Positive Rate) on the ROC curve. So, the ROC curve is generated by trying all cut-off points between the data points (trying extra points between two data points would be redundant):

#	C	Score
1	P	0,9
2	P	0,8
3	N	0,7
4	P	0,6
5	P	0,55
6	P	0,54
7	N	0,53
8	N	0,52
9	P	0,51
10	N	0,505
11	P	0,4
12	N	0,39
13	P	0,38
14	N	0,37
15	N	0,36
16	N	0,35
17	P	0,34
18	N	0,33
19	P	0,3
20	N	0,1



Source: http://homepage.stat.uiowa.edu/~rdecook/stat6220/ROC_animated2.html (http://homepage.stat.uiowa.edu/~rdecook/stat6220/ROC_animated2.html)

As you can see from the above animation, the first cut-off point classifies everything as negative and so both TPR and FPR is zero, i.e. all positive cases are misclassified and negative cases are correctly classified as negative. As we move up, both the TPR and FPR increases. At the very end, both of them are one, i.e. now all positive cases are correctly classified as positve and all negative cases are misclassified.

When the number of data points are small, the ROC curve looks like a stair case, and as the number of data points increase, it becomes a smoother-curve.

Data

We will keep it simple and use the Iris dataset from scikit-learn. It may be a good idea to go through the lab called **EDA and Visualizations for Iris Dataset** first, to get get yourself familiar with the this dataset (if you haven't done so yet).

In [2]:
```
from operator import itemgetter

import numpy as np
import pandas as pd

from sklearn import datasets
from sklearn.metrics import roc_curve, auc
from sklearn.preprocessing import label_binarize
from sklearn.multiclass import OneVsRestClassifier

from matplotlib import pyplot as plt

%matplotlib inline

DEBUG=False

random_state = 0
```

In [3]:
```
iris = datasets.load_iris()
iris
```

```
       [4.8, 3. , 1.4, 0.1],
       [4.3, 3. , 1.1, 0.1],
       [5.8, 4. , 1.2, 0.2],
       [5.7, 4.4, 1.5, 0.4],
       [5.4, 3.9, 1.3, 0.4],
       [5.1, 3.5, 1.4, 0.3],
       [5.7, 3.8, 1.7, 0.3],
       [5.1, 3.8, 1.5, 0.3],
       [5.4, 3.4, 1.7, 0.2],
       [5.1, 3.7, 1.5, 0.4],
       [4.6, 3.6, 1. , 0.2],
       [5.1, 3.3, 1.7, 0.5],
       [4.8, 3.4, 1.9, 0.2],
       [5. , 3. , 1.6, 0.2],
       [5. , 3.4, 1.6, 0.4],
       [5.2, 3.5, 1.5, 0.2],
       [5.2, 3.4, 1.4, 0.2],
       [4.7, 3.2, 1.6, 0.2],
       [4.8, 3.1, 1.6, 0.2],
       [5.4, 3.4, 1.5, 0.4],
```

In [4]:
```
X = iris.data
Y = labels = iris.target

feature_names = iris.feature_names
Y_names = iris.target_names

n_labels = len(Y_names)

n_samples, n_features = X.shape

print("n_labels=%d \t n_samples=%d \t n_features=%d" % (n_labels, n_sam
```

```
n_labels=3       n_samples=150   n_features=4
```

We will binarize the labels so that we can analyze each case separately:

In [5]:
```
Y = label_binarize(Y, classes=range(n_labels))
n_classes = Y.shape[1]
Y
```

```
       [0, 1, 0],
       [0, 1, 0],
       [0, 1, 0],
       [0, 1, 0],
       [0, 1, 0],
       [0, 1, 0],
       [0, 1, 0],
       [0, 1, 0],
       [0, 1, 0],
       [0, 1, 0],
       [0, 1, 0],
       [0, 1, 0],
       [0, 1, 0],
       [0, 1, 0],
       [0, 1, 0],
       [0, 1, 0],
       [0, 1, 0],
       [0, 1, 0],
       [0, 1, 0],
       [0, 1, 0],
```

Generating the Model Probabilities

Our goal is just to work on the ROC curve, so we won't split the data into train and test sets. Let's use *OneVsRestClassifier* to make predictions on the dataset:

In [6]:
```
from sklearn.linear_model import LogisticRegression

classifier = OneVsRestClassifier(LogisticRegression(solver='lbfgs',
                                    random_state=random_state))

classifier.fit(X, Y)
```

Out[6]:
```
OneVsRestClassifier(estimator=LogisticRegression(random_state=0))
```

Here are the predictions:

In [7]:
```
Y_pred = classifier.predict(X)
Y_pred
```

```
[1, 1, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[1, 0, 0],
[0, 0, 0],
[0, 0, 0],
[0, 0, 0],
[0, 1, 0],
[0, 0, 0],
[0, 1, 0],
[0, 0, 0],
[0, 1, 0],
[0, 0, 0],
[0, 0, 0],
[0, 1, 0],
```

But for the ROC curve we will use the probabilities classifier computed for each class:

In [8]:
```
Y_pred_prob = classifier.predict_proba(X)
Y_pred_prob
```

```
[3.63313724e-05, 1.18146542e-01, 9.83596929e-01],
[1.52053593e-04, 3.95369076e-01, 9.65692457e-01],
[1.00332288e-03, 5.34237129e-01, 5.54660229e-01],
[3.16189337e-04, 7.45463048e-01, 8.35456562e-01],
[2.89833567e-05, 2.96765672e-01, 9.88216216e-01],
[2.24520210e-04, 1.25754325e-01, 9.72140871e-01],
[3.70831992e-04, 3.49066830e-01, 8.72533869e-01],
[2.05605604e-03, 3.04444223e-01, 5.20251762e-01],
[2.79681017e-04, 2.37780111e-01, 8.96448239e-01],
[1.43458181e-04, 2.02273905e-01, 9.72038401e-01],
[4.62125744e-04, 1.63988612e-01, 8.53525760e-01],

[7.76210894e-04, 4.82444396e-01, 8.07666095e-01],
[8.24248388e-05, 2.19863713e-01, 9.83622553e-01],
[1.23514967e-04, 1.35206127e-01, 9.81669572e-01],
[3.65981539e-04, 2.12410319e-01, 8.98970473e-01],
[6.54842124e-04, 5.48887656e-01, 7.40250811e-01],
[5.35635248e-04, 2.90664023e-01, 8.23406178e-01],
[4.11715811e-04, 1.26423970e-01, 9.40693579e-01],
[1.07166706e-03, 3.54504045e-01, 7.31011143e-01]])
```

So, the model used the maximum of each row to predict the final class.

Using the Metrics Module

We will use the *metrics* package to compute the ROC curve and AUC for each class:

In [9]:
```
fpr = dict()
tpr = dict()
roc_auc = dict()
thresholds = dict()
for n_class in range(n_classes):
    fpr[n_class], tpr[n_class], thresholds[n_class] = roc_curve(Y[:, n_
    roc_auc[n_class] = auc(fpr[n_class], tpr[n_class])
```

Let's plot the ROC curve for each class. Here are the colors we will use:

In [10]:
```
colors = ["blue", "red", "green"]
for name, color in zip(Y_names, colors):
    print("{0} <-> {1}".format(name, color))
```

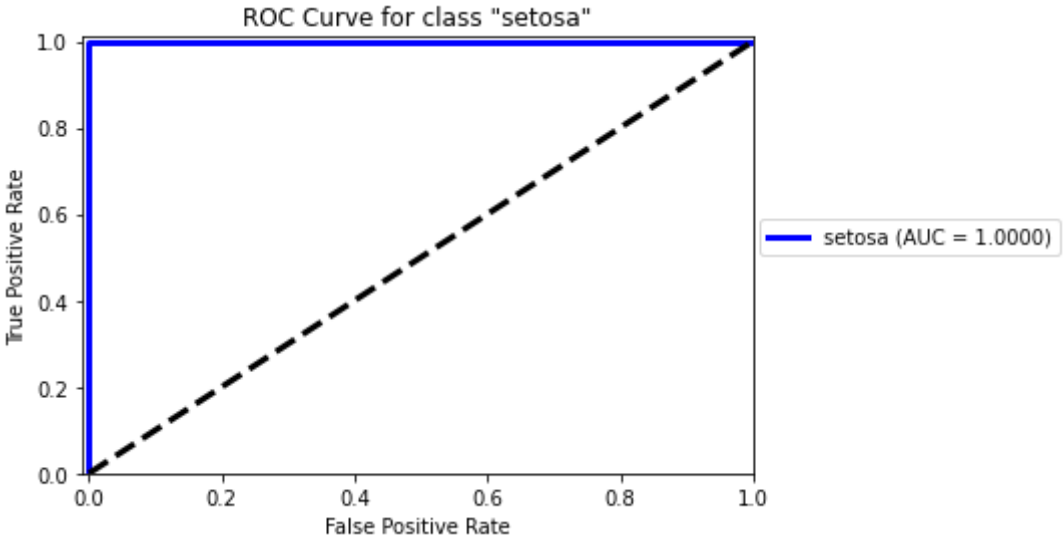
```
setosa <-> blue
versicolor <-> red
virginica <-> green
```

We will use the same code repeatedly, so let's write a function for it:

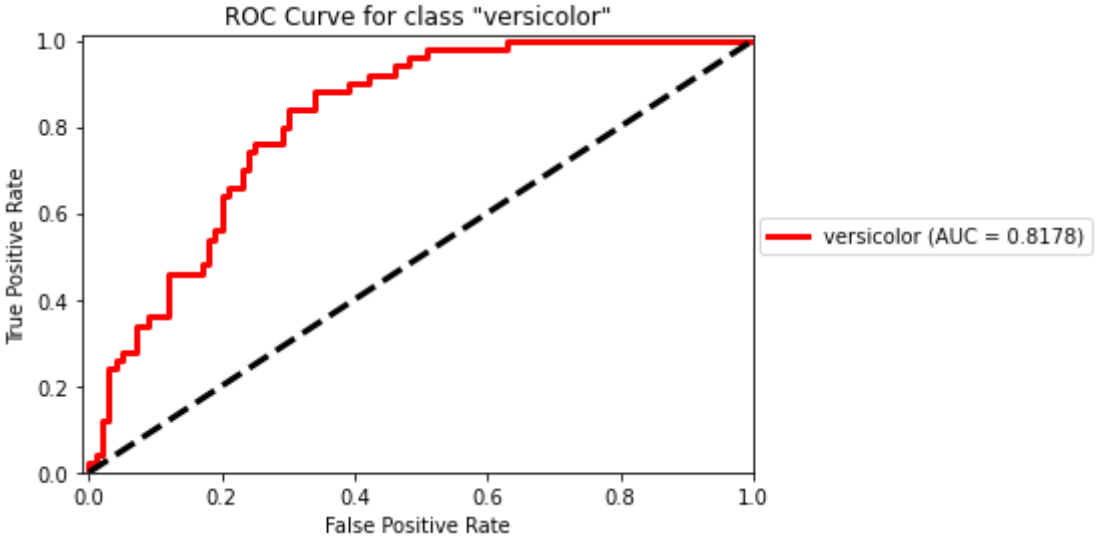
In [11]:
```
def plot_ROC_curve(n_class, fpr, tpr, roc_auc):

    buffer = 0.01 # to be able to clearly visualize the graph close to
    lw = 3  # line width

    plt.figure()
    plt.plot(fpr[n_class], tpr[n_class], color=colors[n_class],
             lw=lw, label="{} (AUC = {:0.4f})".format(Y_names[n_class],

    plt.plot([0, 1], [0, 1], color='black', lw=lw, linestyle='--')
    plt.xlim([-buffer, 1.0])
    plt.ylim([0.0, 1.0+buffer])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('ROC Curve for class \"{}\"'.format(Y_names[n_class]))
    plt.legend(loc=(1.01, 0.5))
    plt.show()
```

Here is the plot of the ROC curve for the each class:

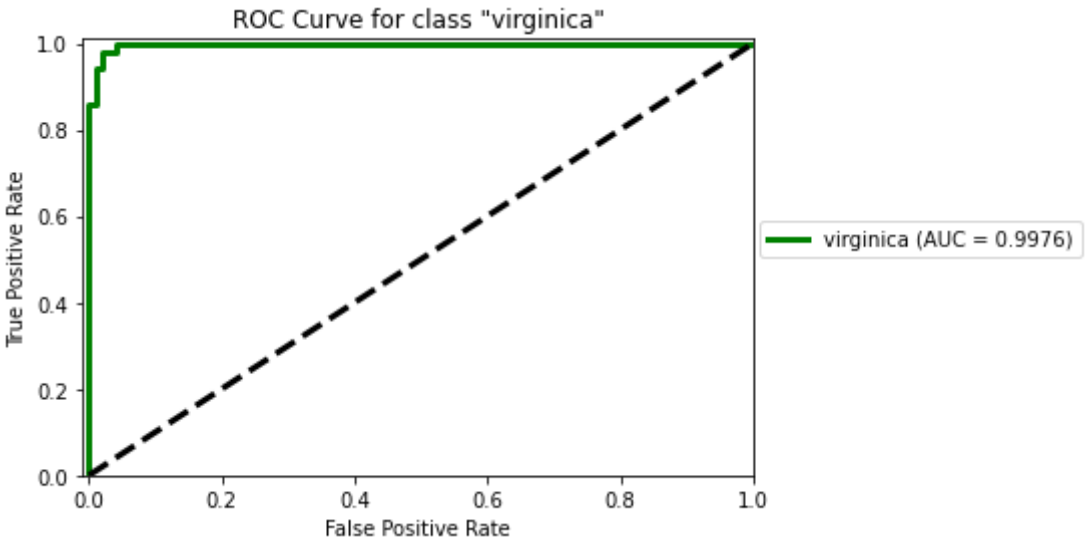
In [12]:
```
1 plot_ROC_curve(0, fpr, tpr, roc_auc)
```



In [13]:
```
1 plot_ROC_curve(1, fpr, tpr, roc_auc)
```



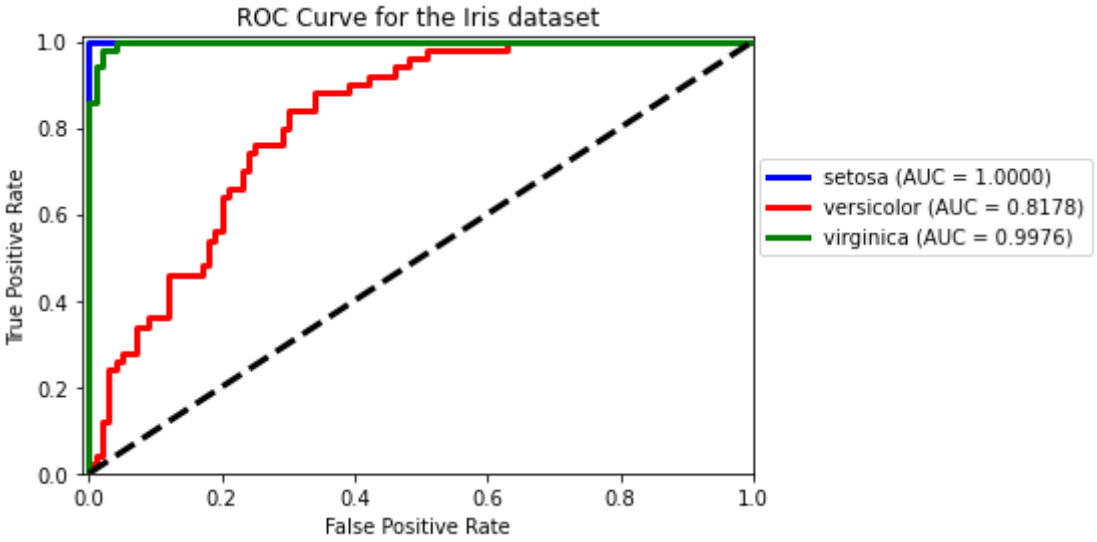
In [14]:
```
plot_ROC_curve(2, fpr, tpr, roc_auc)
```



Let's write a function to plot all above together:

In [15]:
```
def plot_all_ROC_curves(n_classes, fpr, tpr, roc_auc):
    buffer = 0.01 # to be able to clearly visualize the graph close to
    lw = 3  # line width

    plt.figure()

    for n_class in range(n_classes):
        plt.plot(fpr[n_class], tpr[n_class], color=colors[n_class],
                 lw=lw, label="{} (AUC = {:0.4f})".format(Y_names[n_cla

    plt.plot([0, 1], [0, 1], color='black', lw=lw, linestyle='--')
    plt.xlim([-buffer, 1.0])
    plt.ylim([0.0, 1.0+buffer])
    plt.xlabel("False Positive Rate")
    plt.ylabel("True Positive Rate")
    plt.title("ROC Curve for the Iris dataset")
    plt.legend(loc=(1.01, 0.5))
    plt.show()
```

In [16]:
```
1 plot_all_ROC_curves(n_classes, fpr, tpr, roc_auc)
```



Computing from the Ground up

Now, let's compute the ROC curve points ourselves. For a given threshold, we will compute the TPR and FPR of the results. Let's start with an example threshold of 0.5:

In [17]:
```
threshold = 0.5

n_class = 2

N, P, TP, FP = 0, 0, 0, 0

for j in range(len(Y_pred_prob)):
    if Y[j][n_class] == 1:
        P += 1
    if Y[j][n_class] == 0:
        N += 1
    if Y_pred_prob[j][n_class] >= threshold:
        if Y[j][n_class] == 1:
            TP += 1
        if Y[j][n_class] == 0:
            FP += 1

my_fpr = FP/float(N)
my_tpr = TP/float(P)

print(N, P, TP, FP, my_fpr, my_tpr)
```

```
100 50 49 3 0.03 0.98
```

This is just one of the points on the plot. We need to repeat this for a set of thresholds. We will use 100 values in the range [0, 1], i.e. 0.00, 0.01, 0.02, ..., 1.00:

In [18]:
```
n_class = 1

NUM_THRESHOLDS = 10000

my_fpr = {}
my_tpr = {}

my_fpr[n_class] = []
my_tpr[n_class] = []

for i in range(0, NUM_THRESHOLDS+1):
    threshold = i/float(NUM_THRESHOLDS)
    N, P, TP, FP = 0, 0, 0, 0

    for j in range(len(Y_pred_prob)):
        if Y[j][n_class] == 1:
            P += 1
        if Y[j][n_class] == 0:
            N += 1
        if Y_pred_prob[j][n_class] >= threshold:
            if Y[j][n_class] == 1:
                TP += 1
            if Y[j][n_class] == 0:
                FP += 1

    my_fpr[n_class].append(FP/float(N))
    my_tpr[n_class].append(TP/float(P))

print(my_fpr[n_class], my_tpr[n_class])
```

```
0.99, 0.99, 0.99, 0.99, 0.99, 0.99, 0.99, 0.99, 0.99, 0.99, 0.99, 0.99,
0.99, 0.99, 0.99, 0.99, 0.99, 0.99, 0.99, 0.99, 0.99, 0.99, 0.99, 0.99,
0.99, 0.99, 0.99, 0.99, 0.99, 0.99, 0.99, 0.99, 0.99, 0.98, 0.98, 0.98,
0.98, 0.98, 0.98, 0.98, 0.98, 0.98, 0.98, 0.98, 0.98, 0.98, 0.98, 0.98,
0.98, 0.98, 0.98, 0.98, 0.98, 0.98, 0.98, 0.98, 0.98, 0.98, 0.98, 0.98,
0.98, 0.98, 0.98, 0.98, 0.98, 0.98, 0.98, 0.98, 0.98, 0.98, 0.98, 0.98,
0.98, 0.98, 0.98, 0.98, 0.98, 0.98, 0.98, 0.98, 0.98, 0.98, 0.98, 0.98,
0.98, 0.98, 0.98, 0.97, 0.97, 0.97, 0.97, 0.97, 0.97, 0.97, 0.97, 0.97,

0.97, 0.97, 0.97, 0.97, 0.97, 0.97, 0.97, 0.97, 0.97, 0.97, 0.97, 0.97,
0.97, 0.97, 0.97, 0.97, 0.97, 0.97, 0.97, 0.97, 0.97, 0.97, 0.97, 0.97,
0.96, 0.96, 0.96, 0.96, 0.96, 0.96, 0.96, 0.96, 0.96, 0.96, 0.96, 0.96,
0.96, 0.96, 0.96, 0.96, 0.96, 0.96, 0.96, 0.96, 0.96, 0.96, 0.96, 0.96,
0.96, 0.96, 0.96, 0.96, 0.96, 0.96, 0.96, 0.96, 0.96, 0.96, 0.96, 0.96,
0.96, 0.96, 0.96, 0.96, 0.96, 0.96, 0.96, 0.96, 0.96, 0.96, 0.96, 0.96,
0.96, 0.95, 0.95, 0.95, 0.95, 0.95, 0.95, 0.95, 0.95, 0.95, 0.95, 0.95,
0.95, 0.95, 0.95, 0.95, 0.95, 0.95, 0.95, 0.95, 0.95, 0.95, 0.95, 0.95,
0.95, 0.95, 0.95, 0.95, 0.95, 0.95, 0.95, 0.95, 0.95, 0.95, 0.95, 0.95,
0.95, 0.95, 0.95, 0.95, 0.95, 0.95, 0.95, 0.95, 0.95, 0.95, 0.95, 0.95,
```

Let's compute the AUC using the rectanges formed by the y coordinates with all widths equal to 1/NUM_THRESHOLDS :

In [19]:
```
1/float(NUM_THRESHOLDS)
```

Out[19]:
```
0.0001
```

To compute the AUC, first we have to sort the points with respect to the first and second coordinates, so that they form the staircase shape of the ROC Curve:

In [20]:
```
all_points = list(zip(my_fpr[n_class], my_tpr[n_class]))
all_points.sort(key=itemgetter(0, 1))
all_points
```

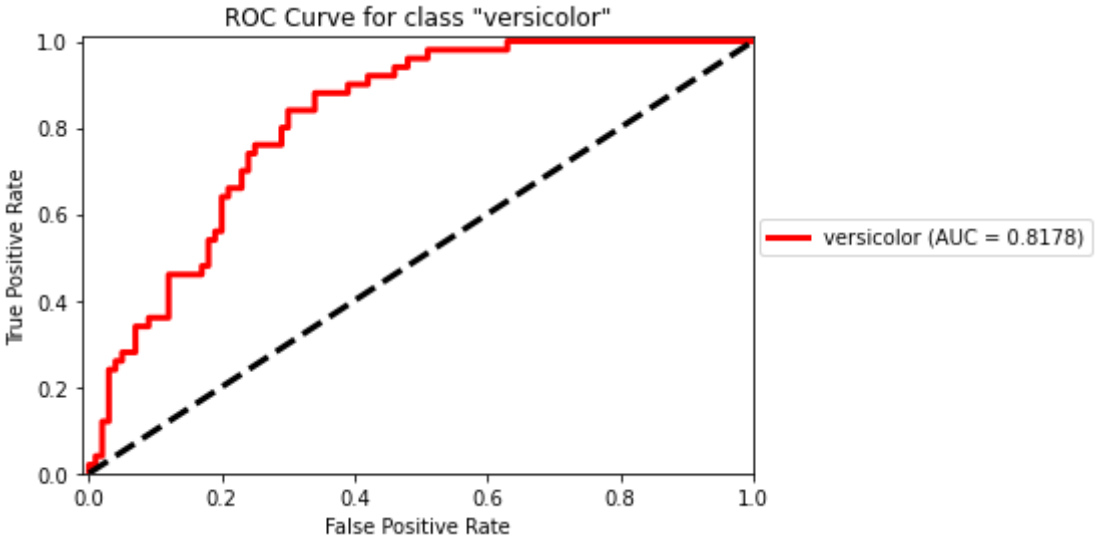
```
(0.0, 0.0),
(0.0, 0.0),
(0.0, 0.0),
(0.0, 0.0),
(0.0, 0.0),
(0.0, 0.0),
(0.0, 0.0),
(0.0, 0.0),
(0.0, 0.0),
(0.0, 0.0),
(0.0, 0.0),
(0.0, 0.0),
(0.0, 0.0),
(0.0, 0.0),
(0.0, 0.0),
(0.0, 0.0),
(0.0, 0.0),
(0.0, 0.0),
(0.0, 0.0),
(0.0, 0.0),
```

In [21]:
```
my_roc_auc = {}
my_roc_auc[n_class] = 0

for i in range(1,len(all_points)):
    height = all_points[i][0] - all_points[i-1][0]
    base_average = (all_points[i][1] + all_points[i-1][1]) / float(2)
    my_roc_auc[n_class] += height * base_average

print(my_roc_auc[n_class])
```

```
0.8178000000000002
```

Let's plot this using our earlier example:

In [22]:
```
1 plot_ROC_curve(1, my_fpr, my_tpr, my_roc_auc)
```



Let's combine all the above and repeat it for all classes:

In [23]:

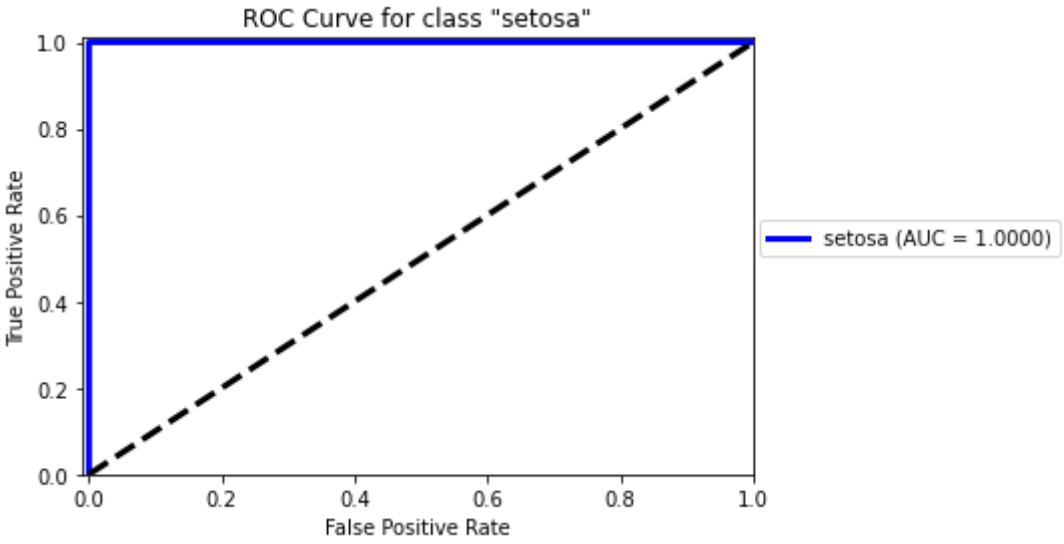
```
my_fpr = {}
my_tpr = {}
my_roc_auc = {}

for n_class in range(n_classes):

    my_fpr[n_class] = []
    my_tpr[n_class] = []

    for i in range(0, NUM_THRESHOLDS+1):
        threshold = i/float(NUM_THRESHOLDS)
        N, P, TP, FP = 0, 0, 0, 0

        for j in range(len(Y_pred_prob)):
            if Y[j][n_class] == 1:
                P += 1
            if Y[j][n_class] == 0:
                N += 1
            if Y_pred_prob[j][n_class] >= threshold:
                if Y[j][n_class] == 1:
                    TP += 1
                if Y[j][n_class] == 0:
                    FP += 1

        my_fpr[n_class].append(FP/float(N))
        my_tpr[n_class].append(TP/float(P))

    all_points = list(zip(my_fpr[n_class], my_tpr[n_class]))
    all_points.sort(key=itemgetter(0,1))

    my_roc_auc[n_class] = 0

    for i in range(1,len(all_points)):
        height = all_points[i][0] - all_points[i-1][0]
        base_average = (all_points[i][1] + all_points[i-1][1]) / float(
        my_roc_auc[n_class] += height * base_average

    print(my_fpr[n_class], my_tpr[n_class], my_roc_auc[n_class])
```

```
6, 0.06, 0.06, 0.06, 0.06, 0.06, 0.06, 0.06, 0.06, 0.06, 0.06, 0.06, 0.0
6, 0.06, 0.06, 0.06, 0.06, 0.06, 0.06, 0.06, 0.06, 0.06, 0.06, 0.06, 0.0
6, 0.06, 0.06, 0.06, 0.06, 0.06, 0.06, 0.06, 0.06, 0.06, 0.06, 0.06, 0.0
6, 0.06, 0.06, 0.06, 0.06, 0.06, 0.06, 0.06, 0.06, 0.06, 0.06, 0.06, 0.0
6, 0.06, 0.06, 0.06, 0.06, 0.06, 0.06, 0.06, 0.06, 0.06, 0.06, 0.06, 0.0
6, 0.06, 0.06, 0.06, 0.06, 0.06, 0.06, 0.06, 0.06, 0.06, 0.06, 0.06, 0.0
6, 0.06, 0.06, 0.06, 0.06, 0.06, 0.06, 0.06, 0.06, 0.06, 0.06, 0.06, 0.0
6, 0.06, 0.06, 0.06, 0.06, 0.06, 0.06, 0.06, 0.06, 0.06, 0.06, 0.06, 0.0
6, 0.06, 0.06, 0.06, 0.06, 0.06, 0.06, 0.06, 0.06, 0.06, 0.06, 0.06, 0.0
6, 0.06, 0.06, 0.06, 0.06, 0.06, 0.06, 0.06, 0.06, 0.06, 0.05, 0.05, 0.0
5, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.04, 0.04, 0.04, 0.04, 0.0
4, 0.04, 0.04, 0.04, 0.04, 0.04, 0.04, 0.04, 0.04, 0.04, 0.04, 0.04, 0.0
4, 0.04, 0.04, 0.04, 0.04, 0.04, 0.04, 0.04, 0.04, 0.04, 0.04, 0.04, 0.0
4, 0.04, 0.04, 0.04, 0.04, 0.04, 0.04, 0.04, 0.04, 0.04, 0.04, 0.04, 0.0
4, 0.04, 0.04, 0.04, 0.04, 0.04, 0.04, 0.04, 0.04, 0.04, 0.04, 0.04, 0.0
4, 0.04, 0.04, 0.04, 0.04, 0.04, 0.04, 0.04, 0.04, 0.04, 0.04, 0.04, 0.0
4, 0.04, 0.04, 0.04, 0.04, 0.04, 0.04, 0.04, 0.04, 0.04, 0.04, 0.04, 0.0
4, 0.04, 0.04, 0.04, 0.04, 0.04, 0.04, 0.04, 0.04, 0.04, 0.04, 0.04, 0.0
```

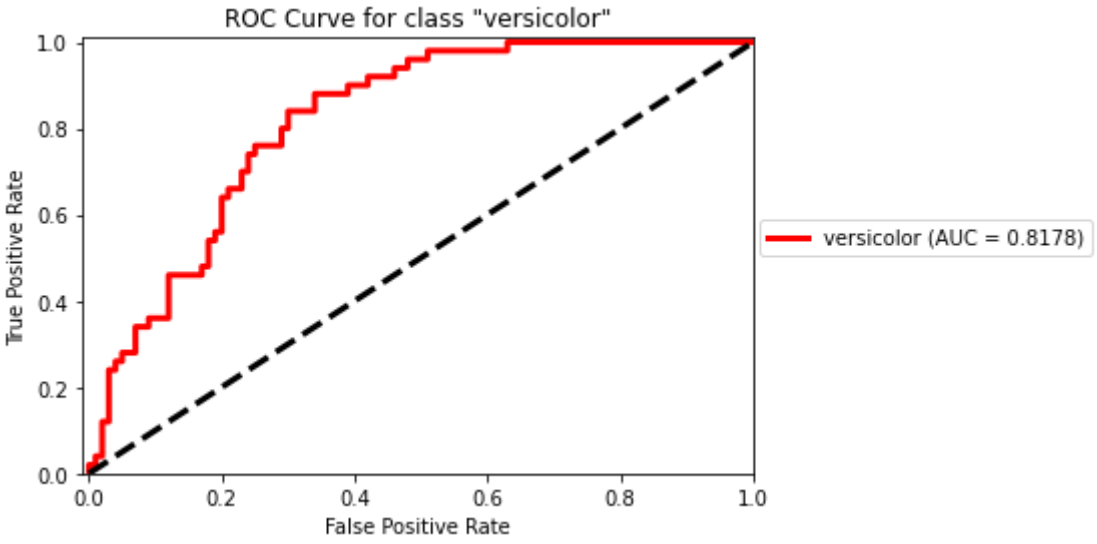
```
4, 0.04, 0.04, 0.04, 0.04, 0.04, 0.04, 0.04, 0.04, 0.04, 0.04, 0.04, 0.0
```

Let's plot these new values now:

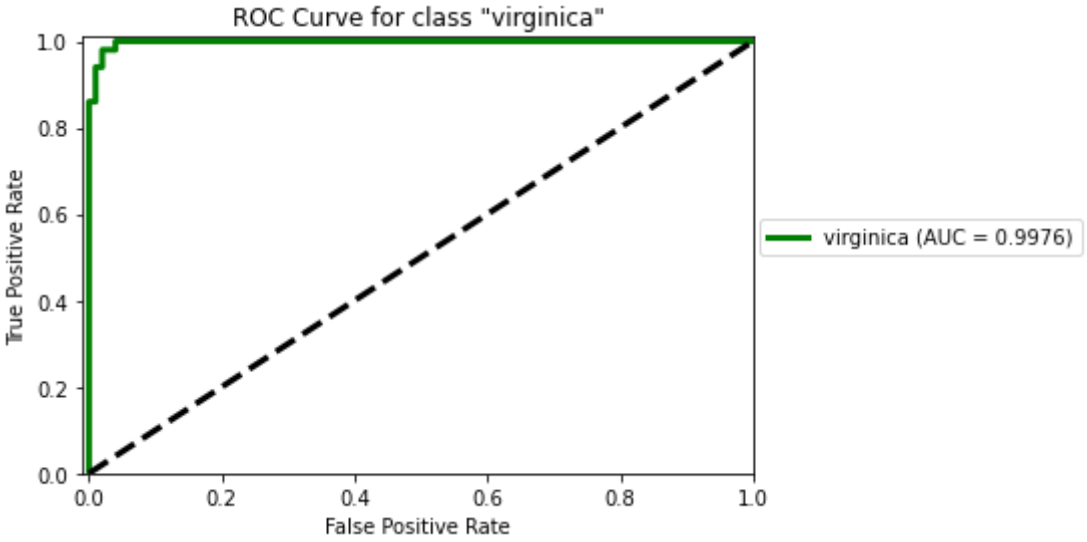
In [24]:
```
1 plot_ROC_curve(0, my_fpr, my_tpr, my_roc_auc)
```



In [25]:
```
1 plot_ROC_curve(1, my_fpr, my_tpr, my_roc_auc)
```

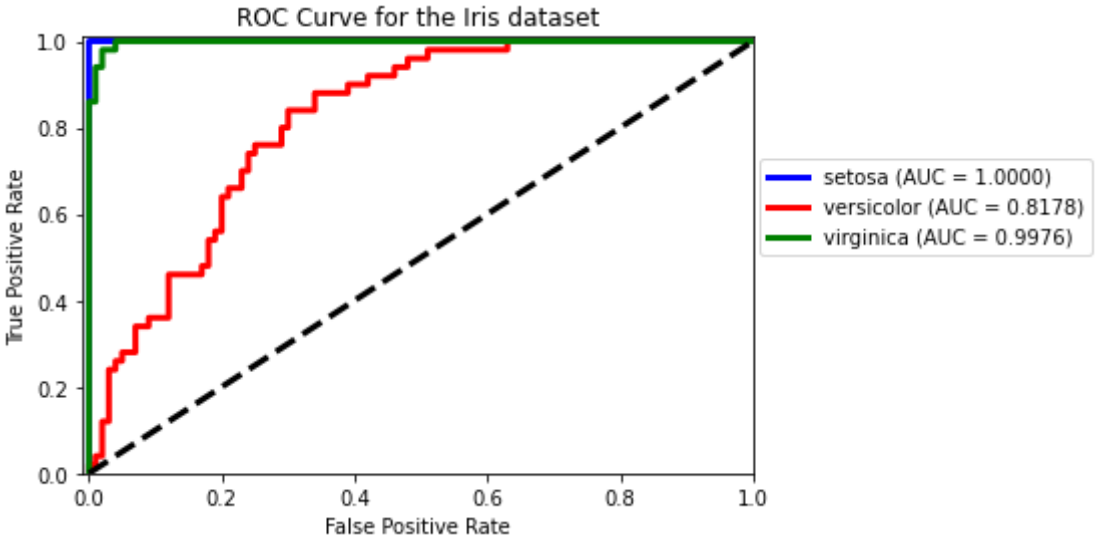


In [26]:
```
1 plot_ROC_curve(2, my_fpr, my_tpr, my_roc_auc)
```



Let's again plot them all together using the function we wrote above:

In [27]:
```
1 plot_all_ROC_curves(n_classes, my_fpr, my_tpr, my_roc_auc)
```



The results are identical to the ones computed by the metrics module. There might have been small

differences here due to different set of thresholds used and roundoff errors, e.g. is you choose to work with 1000 points, you may see some differences.

We can check our results by using the "auc" function of the metrics module on our data:

In [28]:
```
for n_class in range(n_classes):
    my_auc = auc(my_fpr[n_class], my_tpr[n_class])
    print("Class {}  auc={:0.4f}".format(Y_names[n_class], my_auc))
```

```
Class setosa  auc=1.0000
Class versicolor  auc=0.8178
Class virginica  auc=0.9976
```

These are exactly the same results as our AUC computation, so our FPR and TPR computations are correct.

Homework

- Repeat the above for a different dataset, e.g. digits dataset.
- Use different values for NUM_THRESHOLDS, e.g. 100 or 1000 and check if there will be differences in the results.
- Use the thresholds returned by the "roc_curve" function and check if you still get the same results. (note that the set of thresholds changes from one label to another)
- Note that we generated each point on the ROC curve without first sorting the prediction probabilities, thus we had to sort the final points by both x and y coordinates. Repeat the same computation by first sorting the prediction probabilities but changing the order of the the final points and show that this method produces the same ROC curve (make sure to use the same thresholds as above).