Exercises week 43

Kjersti Stangeland October 20-24, 2025

Date: **Deadline Friday October 24 at midnight**

Overarching aims of the exercises for week 43

The aim of the exercises this week is to gain some confidence with ways to visualize the results of a classification problem. We will target three ways of setting up the analysis. The first and simplest one is the

1. so-called confusion matrix. The next one is the so-called
2. ROC curve. Finally we have the
3. Cumulative gain curve.

We will use Logistic Regression as method for the classification in this exercise. You can compare these results with those obtained with your neural network code from project 2 without a hidden layer.

In these exercises we will use binary and multi-class data sets (the Iris data set from week 41).

The underlying mathematics is described here.

Confusion Matrix

A **confusion matrix** summarizes a classifier's performance by tabulating predictions versus true labels. For binary classification, it is a 2×2 table whose entries are counts of outcomes:

$$\begin{array}{l|cc} & \text{Predicted Positive} & \text{Predicted Negative} \\ \hline \text{Actual Positive} & TP & FN \\ \text{Actual Negative} & FP & TN \end{array}.$$

Here TP (true positives) is the number of cases correctly predicted as positive, FP (false positives) is the number incorrectly predicted as positive, TN (true negatives) is correctly predicted negative, and FN (false negatives) is incorrectly predicted negative . In other words, “positive” means class 1 and “negative” means class 0; for example, TP occurs when the prediction and actual are both positive. Formally:

$$\mathrm{TPR} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}, \quad \mathrm{FPR} = \frac{\mathrm{FP}}{\mathrm{FP} + \mathrm{TN}},$$

where TPR and FPR are the true and false positive rates defined below.

In multiclass classification with K classes, the confusion matrix generalizes to a $K \times K$ table. Entry N_{ij} in the table is the count of instances whose true class is i and whose predicted class is j. For example, a three-class confusion matrix can be written as:

	Pred Class 1	Pred Class 2	Pred Class 3
Act Class 1	N_{11}	N_{12}	N_{13}
Act Class 2	N_{21}	N_{22}	N_{23}
Act Class 3	N_{31}	N_{32}	N_{33}

Here the diagonal entries N_{ii} are the true positives for each class, and off-diagonal entries are misclassifications. This matrix allows computation of per-class metrics: e.g. for class i, $\mathrm{TP}_i = N_{ii}$, $\mathrm{FN}_i = \sum_{j \neq i} N_{ij}$, $\mathrm{FP}_i = \sum_{j \neq i} N_{ji}$, and TN_i is the sum of all remaining entries.

As defined above, TPR and FPR come from the binary case. In binary terms with P actual positives and N actual negatives, one has

$$\mathrm{TPR} = \frac{TP}{P} = \frac{TP}{TP + FN}, \quad \mathrm{FPR} = \frac{FP}{N} = \frac{FP}{FP + TN},$$

as used in standard confusion-matrix formulations. These rates will be used in constructing ROC curves.

ROC Curve

The Receiver Operating Characteristic (ROC) curve plots the trade-off between true positives and false positives as a discrimination threshold varies. Specifically, for a binary classifier that outputs a score or probability, one varies the threshold t for declaring **positive**, and computes at each t the true positive rate $\mathrm{TPR}(t)$ and false positive rate $\mathrm{FPR}(t)$ using the confusion matrix at that threshold. The ROC curve is then the graph of TPR versus FPR. By definition,

$$\mathrm{TPR} = \frac{TP}{TP + FN}, \qquad \mathrm{FPR} = \frac{FP}{FP + TN},$$

where TP, FP, TN, FN are counts determined by threshold t. A perfect classifier would reach the point (FPR=0, TPR=1) at some threshold.

Formally, the ROC curve is obtained by plotting $(\mathrm{FPR}(t), \mathrm{TPR}(t))$ for all $t \in [0, 1]$ (or as t sweeps through the sorted scores). The Area Under the ROC Curve (AUC) quantifies the average performance over all thresholds. It can be interpreted probabilistically: $\mathrm{AUC} = \Pr\big(s(X^+) > s(X^-)\big)$, the probability that a random positive instance X^+ receives a higher score s than a random negative instance X^- . Equivalently, the AUC is the integral under the ROC curve:

$$\mathrm{AUC} \;=\; \int_0^1 \mathrm{TPR}(f)\, df,$$

where f ranges over FPR (or fraction of negatives). A model that guesses at random yields a diagonal ROC (AUC=0.5), whereas a perfect model yields AUC=1.0.

Cumulative Gain

The cumulative gain curve (or gains chart) evaluates how many positives are captured as one targets an increasing fraction of the population, sorted by model confidence. To construct it, sort all instances by decreasing predicted probability of the positive class. Then, for the top α fraction of instances, compute the fraction of all actual positives that fall in this subset. In formula form, if P is the total number of positive instances and $P(\alpha)$ is the number of positives among the top α of the data, the cumulative gain at level α is

$$\mathrm{Gain}(\alpha) \;=\; \frac{P(\alpha)}{P}.$$

For example, cutting off at the top 10% of predictions yields a gain equal to (positives in top 10%) divided by (total positives) . Plotting $\mathrm{Gain}(\alpha)$ versus α (often in percent) gives the gain curve. The baseline (random) curve is the diagonal $\mathrm{Gain}(\alpha) = \alpha$, while an ideal model has a steep climb toward 1.

A related measure is the {\em lift}, often called the gain ratio. It is the ratio of the model's capture rate to that of random selection. Equivalently,

$$\mathrm{Lift}(\alpha) \;=\; \frac{\mathrm{Gain}(\alpha)}{\alpha}.$$

A lift > 1 indicates better-than-random targeting. In practice, gain and lift charts (used e.g.\ in marketing or imbalanced classification) show how many positives can be "gained" by focusing on a fraction of the population .

Other measures: Precision, Recall, and the F$_1$ Measure

Precision and recall (sensitivity) quantify binary classification accuracy in terms of positive predictions. They are defined from the confusion matrix as:

$$\text{Precision} = \frac{TP}{TP + FP}, \qquad \text{Recall} = \frac{TP}{TP + FN}.$$

Precision is the fraction of predicted positives that are correct, and recall is the fraction of actual positives that are correctly identified . A high-precision classifier makes few false-positive errors, while a high-recall classifier makes few false-negative errors.

The F$_1$ score (balanced F-measure) combines precision and recall into a single metric via their harmonic mean. The usual formula is:

$$F_1 = 2\frac{\text{Precision}\times\text{Recall}}{\text{Precision} + \text{Recall}}.$$

This can be shown to equal

$$\frac{2\,TP}{2\,TP + FP + FN}.$$

The F$_1$ score ranges from 0 (worst) to 1 (best), and balances the trade-off between precision and recall.

For multi-class classification, one computes per-class precision/recall/F$_1$ (treating each class as "positive" in a one-vs-rest manner) and then averages. Common averaging methods are:

Micro-averaging: Sum all true positives, false positives, and false negatives across classes, then compute precision/recall/F$_1$ from these totals. Macro-averaging: Compute the F1 score $F1, i$ for each class i separately, then take the unweighted mean: $F_{1,\text{macro}} = \frac{1}{K}\sum_{i=1}^{K} F_{1,i}$. This treats all classes equally regardless of size. Weighted-averaging: Like macro-average, but weight each class's $F_{1,i}$ by its support n_i (true count): $F_{1,\text{weighted}} = \frac{1}{N}\sum_{i=1}^{K} n_i F_{1,i}$, where $N = \sum_i n_i$. This accounts for class imbalance by giving more weight to larger classes .

Each of these averages has different use-cases. Micro-average is dominated by common classes, macro-average highlights performance on rare classes, and weighted-average is a compromise. These formulas and concepts allow rigorous evaluation of classifier performance in both binary and multi-class settings.

Exercises

Here is a simple code example which uses the Logistic regression machinery from **scikit-learn**. At the end it sets up the confusion matrix and the ROC and cumulative gain curves. Feel free to use these functionalities (we don't expect you to write your own code for say the confusion matrix).

In [5]:
```python
%matplotlib inline
import scipy
scipy.interp = np.interp
import scikitplot as skplt
import matplotlib.pyplot as plt
import numpy as np
from sklearn.model_selection import  train_test_split
from sklearn.datasets import load_breast_cancer
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import cross_validate
```

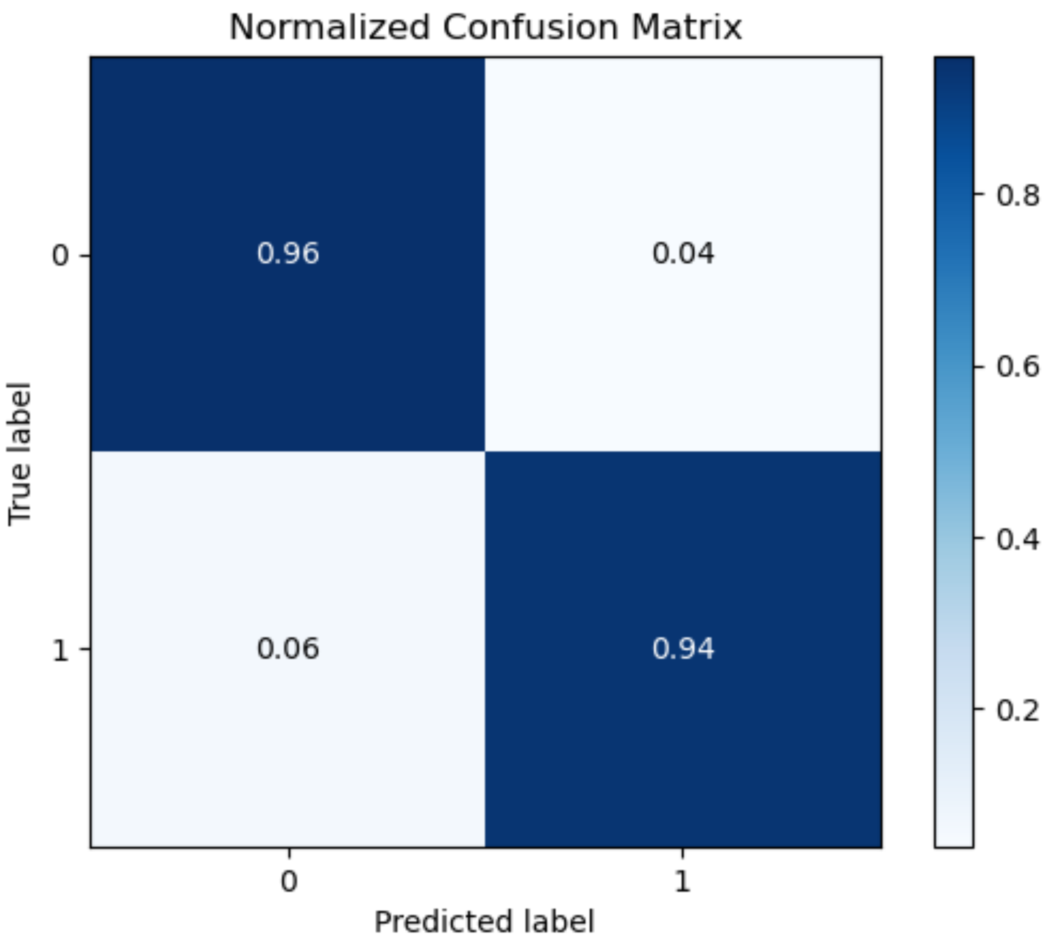
In [8]:
```python
# Load the data, fill inn
mydata = load_breast_cancer()

X_train, X_test, y_train, y_test = train_test_split(mydata.data, mydata.t
print(X_train.shape)
print(X_test.shape)
# Logistic Regression
logreg = LogisticRegression(solver='lbfgs')
logreg.fit(X_train, y_train)

#Cross validation
accuracy = cross_validate(logreg,X_test,y_test,cv=10)['test_score']
print(accuracy)
print("Test set accuracy with Logistic Regression: {:.2f}".format(logreg.
y_pred = logreg.predict(X_test)
skplt.metrics.plot_confusion_matrix(y_test, y_pred, normalize=True)
plt.show()
y_probas = logreg.predict_proba(X_test)
skplt.metrics.plot_roc(y_test, y_probas)
plt.show()
skplt.metrics.plot_cumulative_gain(y_test, y_probas)
plt.show()
```

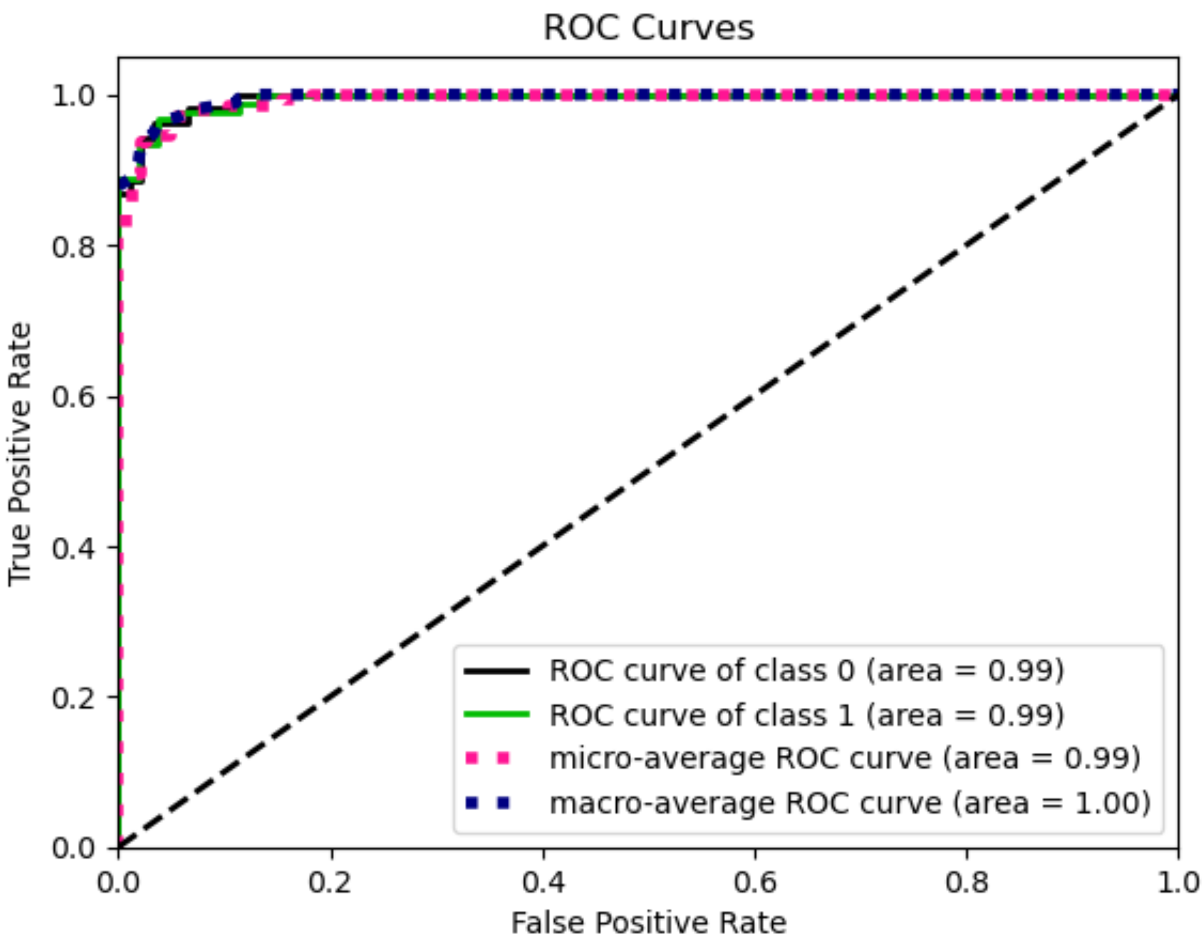
```
(426, 30)
(143, 30)
[1.         0.86666667 1.         0.92857143 1.         0.85714286
 1.         0.92857143 0.92857143 1.        ]
Test set accuracy with Logistic Regression: 0.95
```

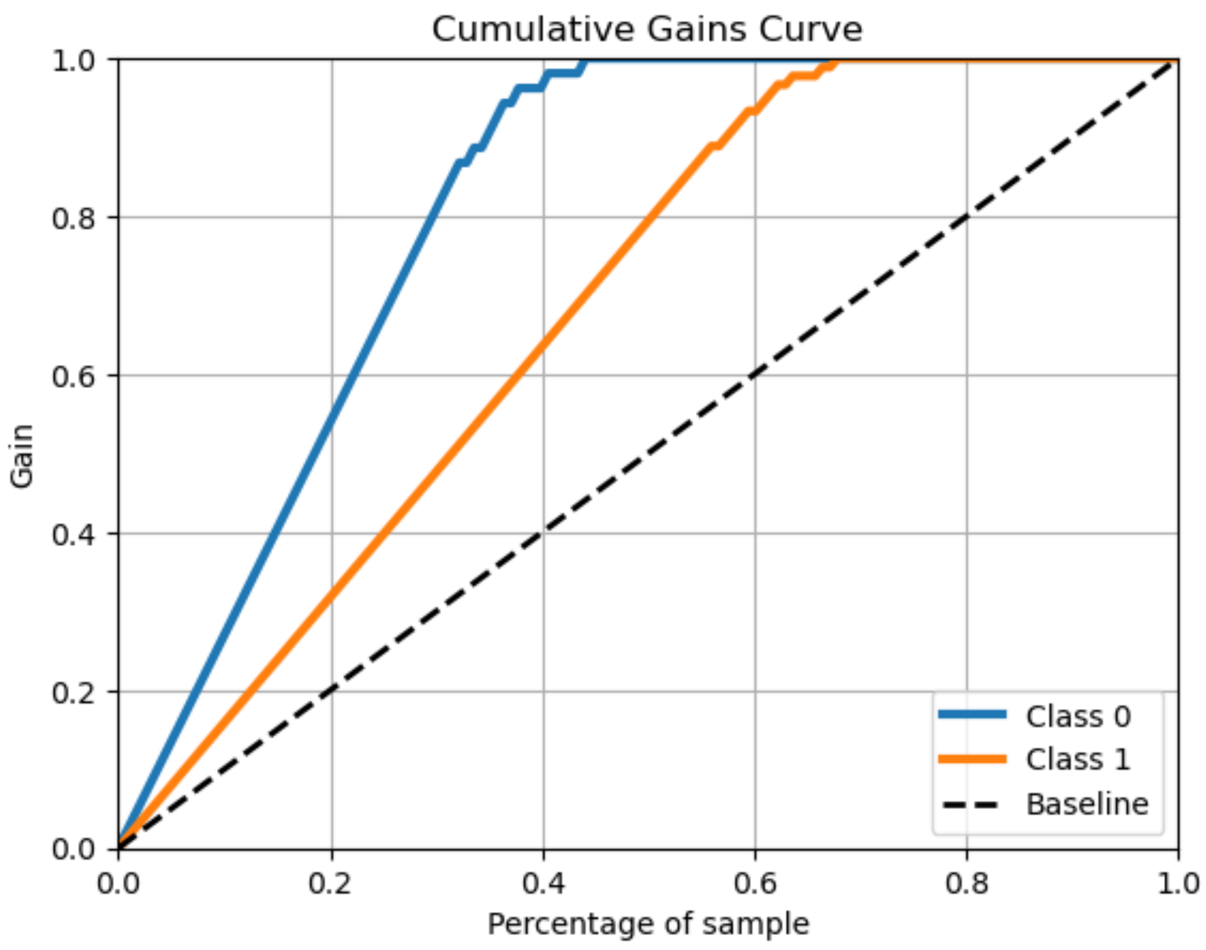
```
/Users/kjesta/miniconda3/envs/exercises/lib/python3.13/site-packages/sklea
rn/linear_model/_logistic.py:473: ConvergenceWarning: lbfgs failed to conv
erge after 100 iteration(s) (status=1):
STOP: TOTAL NO. OF ITERATIONS REACHED LIMIT

Increase the number of iterations to improve the convergence (max_iter=10
0).
You might also want to scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-reg
ression
  n_iter_i = _check_optimize_result(
/Users/kjesta/miniconda3/envs/exercises/lib/python3.13/site-packages/sklea
rn/linear_model/_logistic.py:473: ConvergenceWarning: lbfgs failed to conv
erge after 100 iteration(s) (status=1):
STOP: TOTAL NO. OF ITERATIONS REACHED LIMIT

Increase the number of iterations to improve the convergence (max_iter=10
0).
You might also want to scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-reg
ression
  n_iter_i = _check_optimize_result(
/Users/kjesta/miniconda3/envs/exercises/lib/python3.13/site-packages/sklea
rn/linear_model/_logistic.py:473: ConvergenceWarning: lbfgs failed to conv
erge after 100 iteration(s) (status=1):
STOP: TOTAL NO. OF ITERATIONS REACHED LIMIT

Increase the number of iterations to improve the convergence (max_iter=10
0).
You might also want to scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-reg
ression
  n_iter_i = _check_optimize_result(
/Users/kjesta/miniconda3/envs/exercises/lib/python3.13/site-packages/sklea
```

```
rn/linear_model/_logistic.py:473: ConvergenceWarning: lbfgs failed to conv
erge after 100 iteration(s) (status=1):
STOP: TOTAL NO. OF ITERATIONS REACHED LIMIT

Increase the number of iterations to improve the convergence (max_iter=10
0).
You might also want to scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-reg
ression
  n_iter_i = _check_optimize_result(
/Users/kjesta/miniconda3/envs/exercises/lib/python3.13/site-packages/sklea
rn/linear_model/_logistic.py:473: ConvergenceWarning: lbfgs failed to conv
erge after 100 iteration(s) (status=1):
STOP: TOTAL NO. OF ITERATIONS REACHED LIMIT

Increase the number of iterations to improve the convergence (max_iter=10
0).
You might also want to scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-reg
ression
  n_iter_i = _check_optimize_result(
/Users/kjesta/miniconda3/envs/exercises/lib/python3.13/site-packages/sklea
rn/linear_model/_logistic.py:473: ConvergenceWarning: lbfgs failed to conv
erge after 100 iteration(s) (status=1):
STOP: TOTAL NO. OF ITERATIONS REACHED LIMIT

Increase the number of iterations to improve the convergence (max_iter=10
0).
You might also want to scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-reg
ression
  n_iter_i = _check_optimize_result(
/Users/kjesta/miniconda3/envs/exercises/lib/python3.13/site-packages/sklea
rn/linear_model/_logistic.py:473: ConvergenceWarning: lbfgs failed to conv
erge after 100 iteration(s) (status=1):
STOP: TOTAL NO. OF ITERATIONS REACHED LIMIT

Increase the number of iterations to improve the convergence (max_iter=10
0).
You might also want to scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-reg
ression
  n_iter_i = _check_optimize_result(
/Users/kjesta/miniconda3/envs/exercises/lib/python3.13/site-packages/sklea
rn/linear_model/_logistic.py:473: ConvergenceWarning: lbfgs failed to conv
```

```
erge after 100 iteration(s) (status=1):
STOP: TOTAL NO. OF ITERATIONS REACHED LIMIT

Increase the number of iterations to improve the convergence (max_iter=10
0).
You might also want to scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-reg
ression
  n_iter_i = _check_optimize_result(
/Users/kjesta/miniconda3/envs/exercises/lib/python3.13/site-packages/sklea
rn/linear_model/_logistic.py:473: ConvergenceWarning: lbfgs failed to conv
erge after 100 iteration(s) (status=1):
STOP: TOTAL NO. OF ITERATIONS REACHED LIMIT

Increase the number of iterations to improve the convergence (max_iter=10
0).
You might also want to scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-reg
ression
  n_iter_i = _check_optimize_result(
/Users/kjesta/miniconda3/envs/exercises/lib/python3.13/site-packages/sklea
rn/linear_model/_logistic.py:473: ConvergenceWarning: lbfgs failed to conv
erge after 100 iteration(s) (status=1):
STOP: TOTAL NO. OF ITERATIONS REACHED LIMIT

Increase the number of iterations to improve the convergence (max_iter=10
0).
You might also want to scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-reg
ression
  n_iter_i = _check_optimize_result(
/Users/kjesta/miniconda3/envs/exercises/lib/python3.13/site-packages/sklea
rn/linear_model/_logistic.py:473: ConvergenceWarning: lbfgs failed to conv
erge after 100 iteration(s) (status=1):
STOP: TOTAL NO. OF ITERATIONS REACHED LIMIT

Increase the number of iterations to improve the convergence (max_iter=10
0).
You might also want to scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-reg
ression
  n_iter_i = _check_optimize_result(
```


Normalized Confusion Matrix
True label
Predicted label
0
1
0
1
0.96
0.04
0.06
0.94
0.8
0.6
0.4
0.2


ROC Curves
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve of class 0 (area = 0.99)
ROC curve of class 1 (area = 0.99)
micro-average ROC curve (area = 0.99)
macro-average ROC curve (area = 1.00)



Exercise a)

Convince yourself about the mathematics for the confusion matrix, the ROC and the cumlative gain curves for both a binary and a multiclass classification problem.

Exercise b)

Use a binary classification data available from **scikit-learn**. As an example you can use the MNIST data set and just specialize to two numbers. To do so you can use the following code lines

In [9]:
```
from sklearn.datasets import load_digits
digits = load_digits(n_class=2) # Load only two classes, e.g., 0 and 1
X, y = digits.data, digits.target
```

Alternatively, you can use the *make*$|
$classification_functionality. This function generates a random n$-class classification dataset, which can be configured for binary classification by setting n_classes=2. You can also control the number of samples, features, informative features, redundant features, and more.

In [11]:
```
from sklearn.datasets import make_classification
X, y = make_classification(n_samples=1000, n_features=20, n_informative=1
```

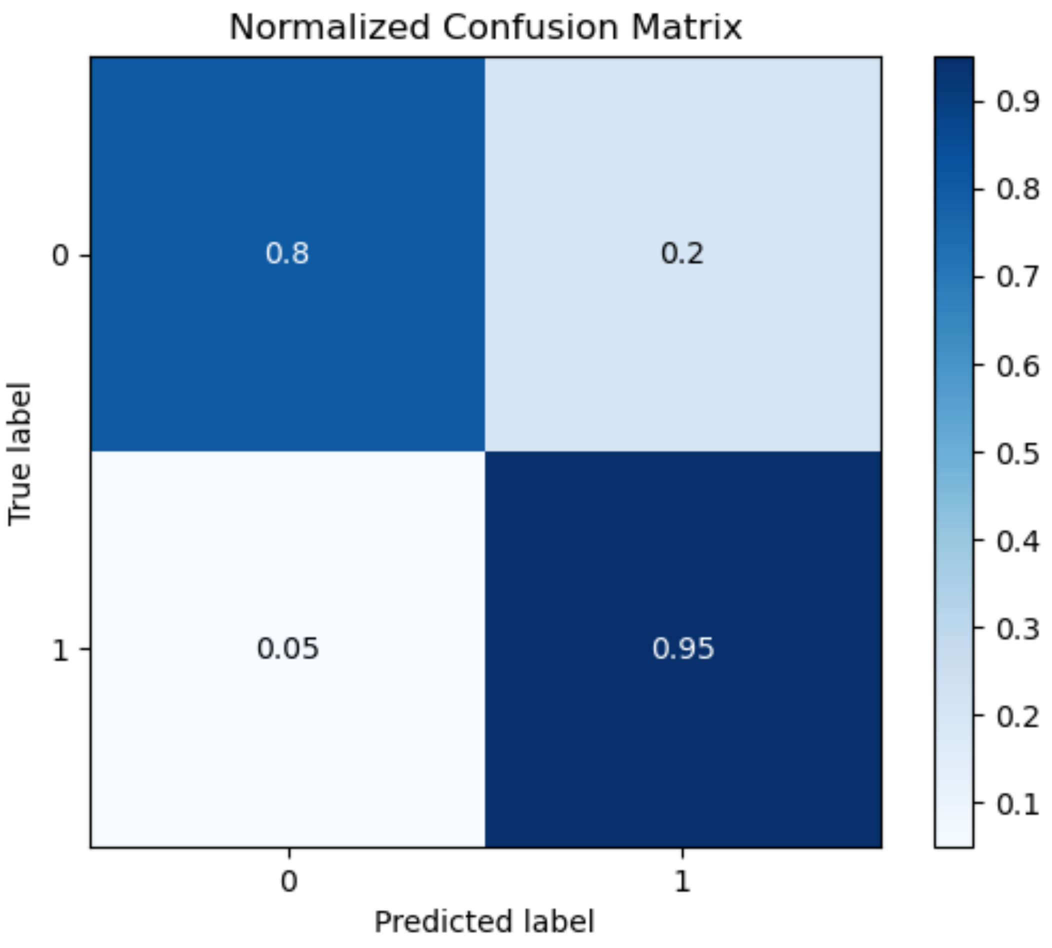
You can use this option for the multiclass case as well, see the next exercise. If you prefer to study other binary classification datasets, feel free to replace the above suggestions with your own dataset.

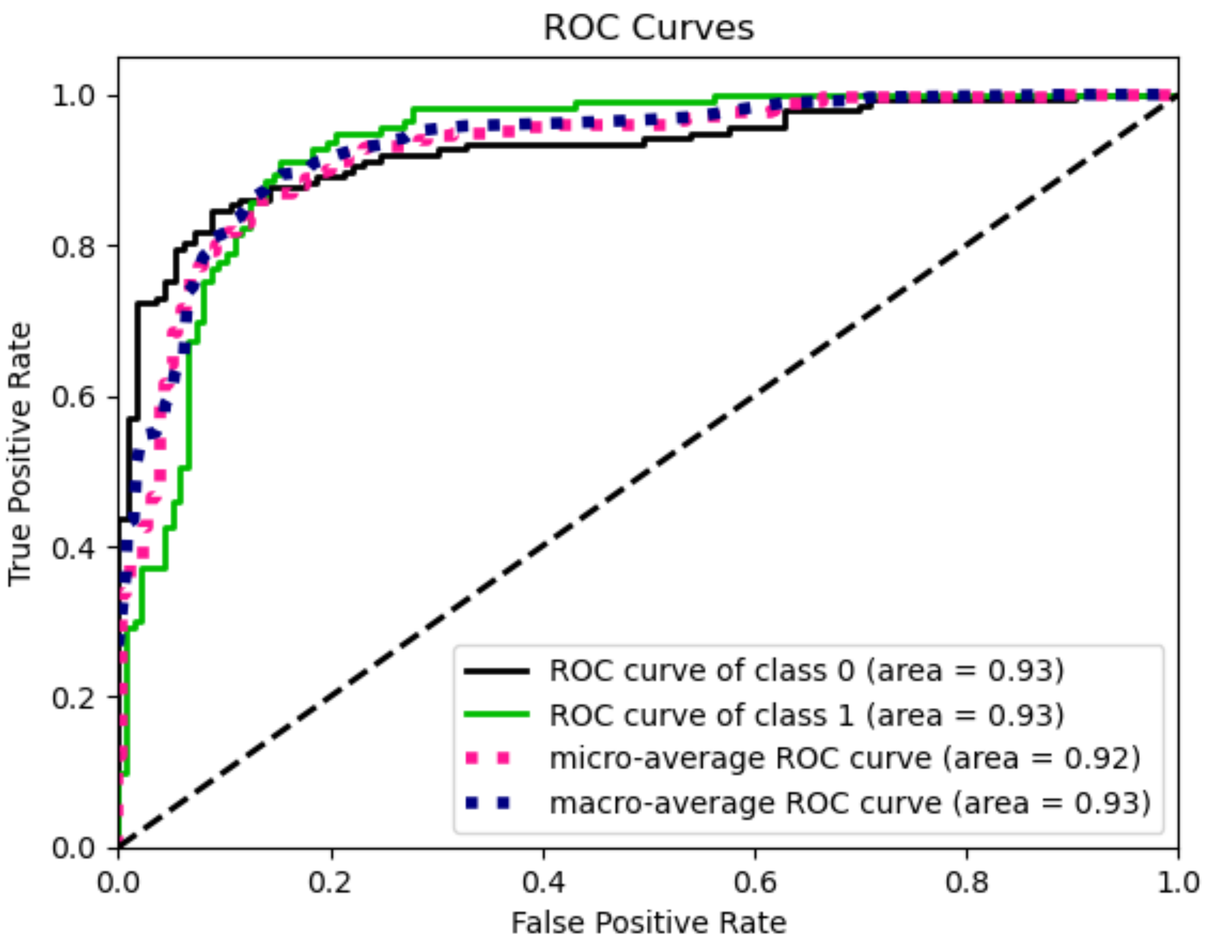
Make plots of the confusion matrix, the ROC curve and the cumulative gain curve.

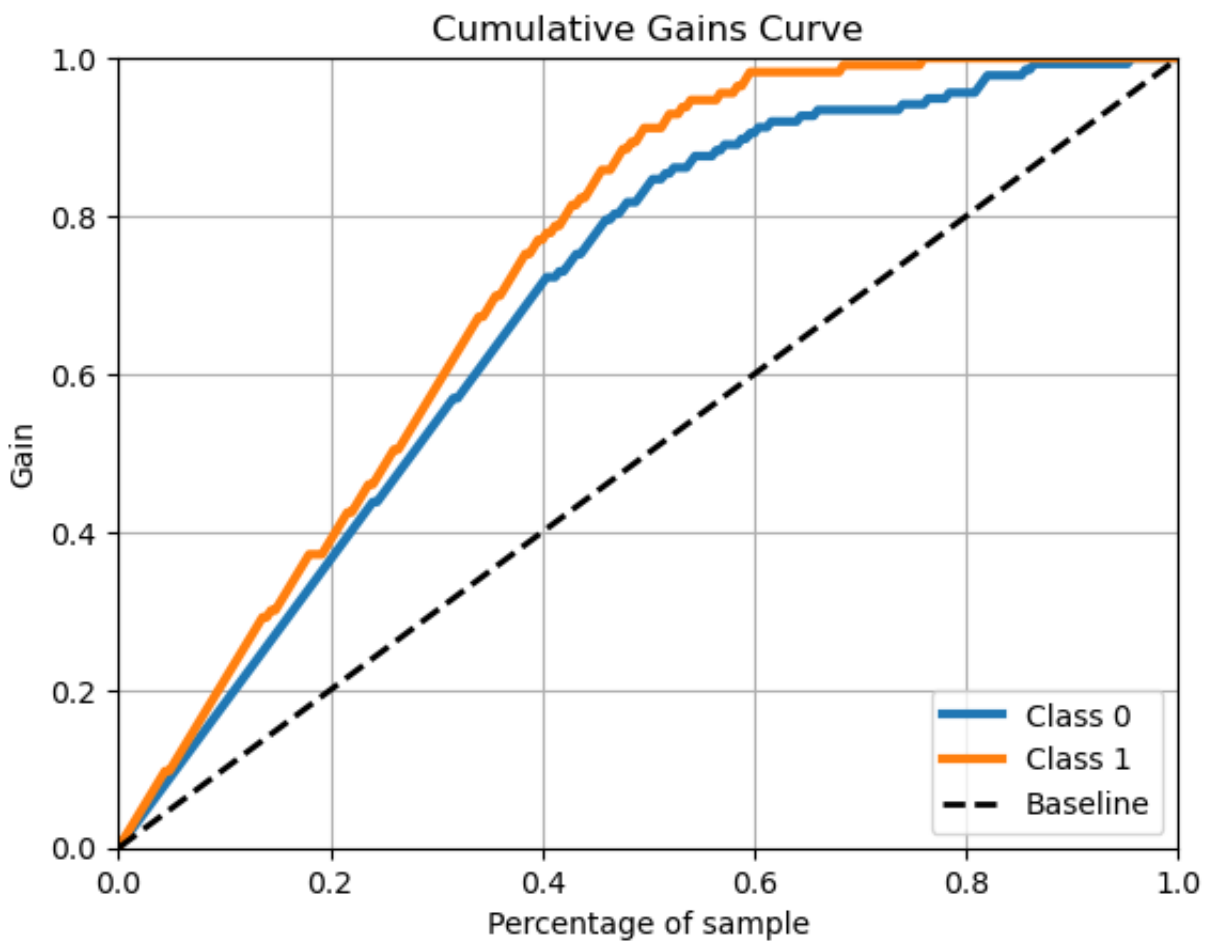
In [12]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
print(X_train.shape)
print(X_test.shape)

# Logistic Regression
# define which type of problem, binary or multiclass
logreg = LogisticRegression(solver='lbfgs')
logreg.fit(X_train, y_train)

#Cross validation
accuracy = cross_validate(logreg,X_test,y_test,cv=10)['test_score']
print(accuracy)
print("Test set accuracy with Logistic Regression: {:.2f}".format(logreg.
```

```
(750, 20)
(250, 20)
[0.64 0.92 0.96 0.8  0.88 0.88 0.88 0.92 0.84 0.68]
Test set accuracy with Logistic Regression: 0.86
```

In [13]:
```
y_pred = logreg.predict(X_test)
skplt.metrics.plot_confusion_matrix(y_test, y_pred, normalize=True)
plt.show()
y_probas = logreg.predict_proba(X_test)
skplt.metrics.plot_roc(y_test, y_probas)
plt.show()
skplt.metrics.plot_cumulative_gain(y_test, y_probas)
plt.show()
```


Normalized Confusion Matrix
True label
Predicted label
0
1
0
1
0.8
0.2
0.05
0.95
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1


ROC Curves
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve of class 0 (area = 0.93)
ROC curve of class 1 (area = 0.93)
micro-average ROC curve (area = 0.92)
macro-average ROC curve (area = 0.93)



Exercise c) week 43

As a multiclass problem, we will use the Iris data set discussed in the exercises from weeks 41 and 42. This is a three-class data set and you can set it up using **scikit-learn**,

In [ ]:
```
from sklearn.datasets import load_iris
iris = load_iris()
X = iris.data  # Features
y = iris.target # Target labels
```

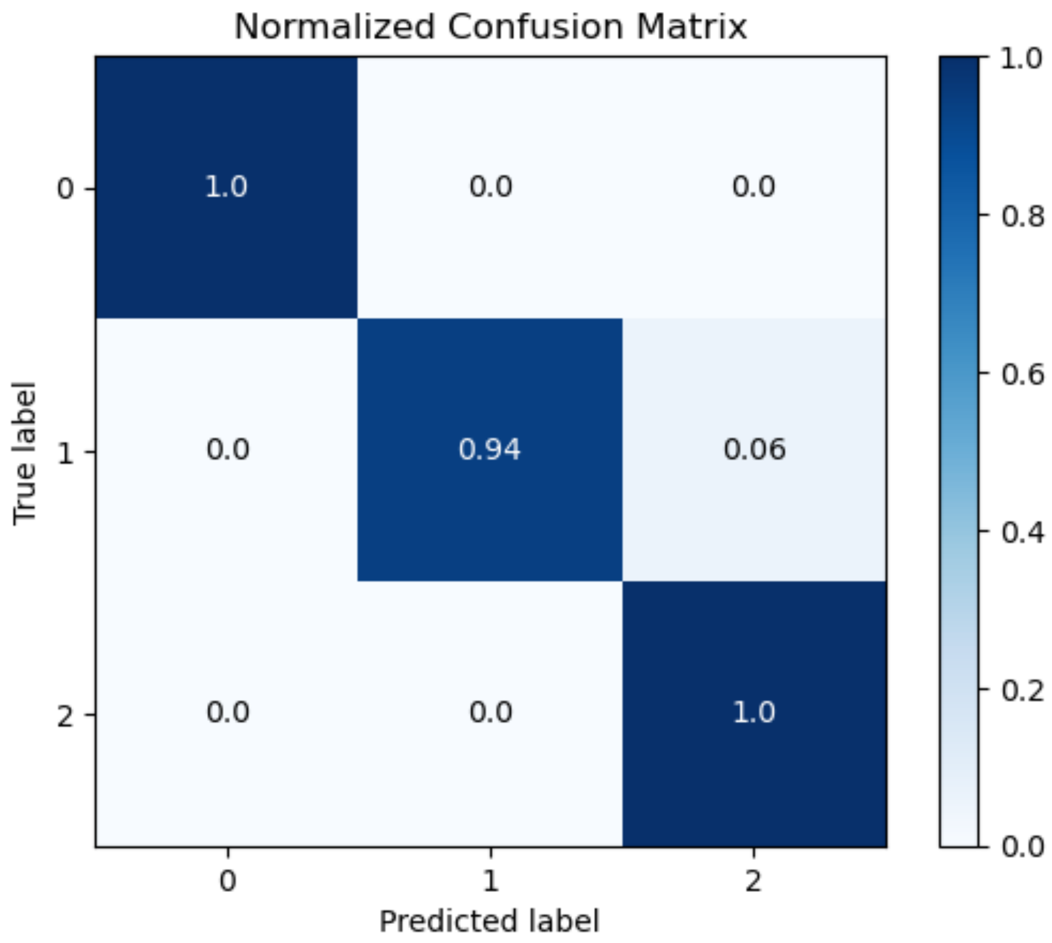
Make plots of the confusion matrix, the ROC curve and the cumulative gain curve for this (or other) multiclass data set.

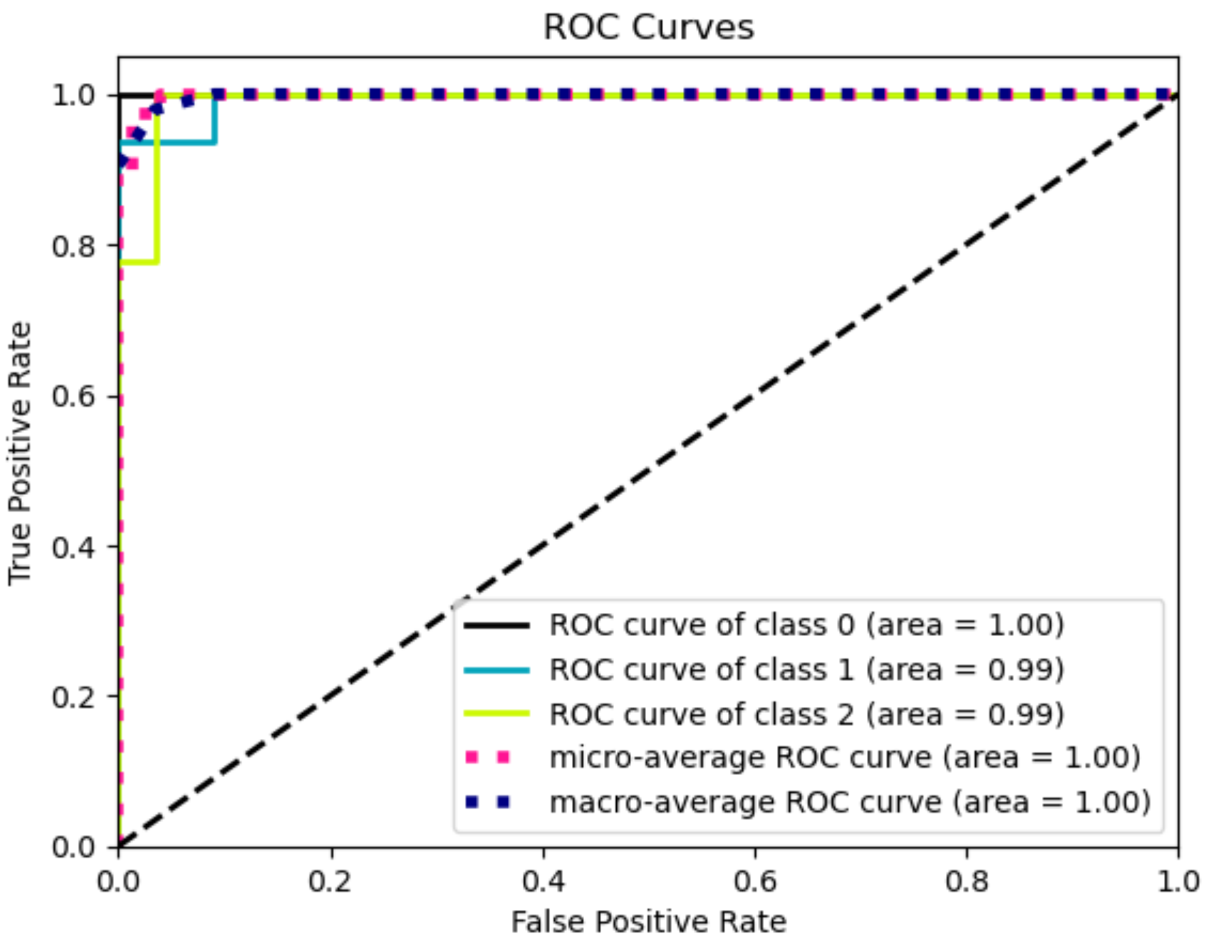
In [21]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
print(X_train.shape)
print(X_test.shape)

# Logistic Regression
logreg = LogisticRegression(solver='lbfgs')
logreg.fit(X_train, y_train)

#Cross validation
```

```python
accuracy = cross_validate(logreg,X_test,y_test,cv=10)['test_score']
print(accuracy)
print("Test set accuracy with Logistic Regression: {:.2f}".format(logreg.

# Confusion matrix
y_pred = logreg.predict(X_test)
skplt.metrics.plot_confusion_matrix(y_test, y_pred, normalize=True)
plt.show()

# ROC curve
y_probas = logreg.predict_proba(X_test)
skplt.metrics.plot_roc(y_test, y_probas)
plt.show()
```

```
(112, 4)
(38, 4)
[1.   1.   1.   0.75 1.   0.75 1.   0.75 1.   1.  ]
Test set accuracy with Logistic Regression: 0.97
```

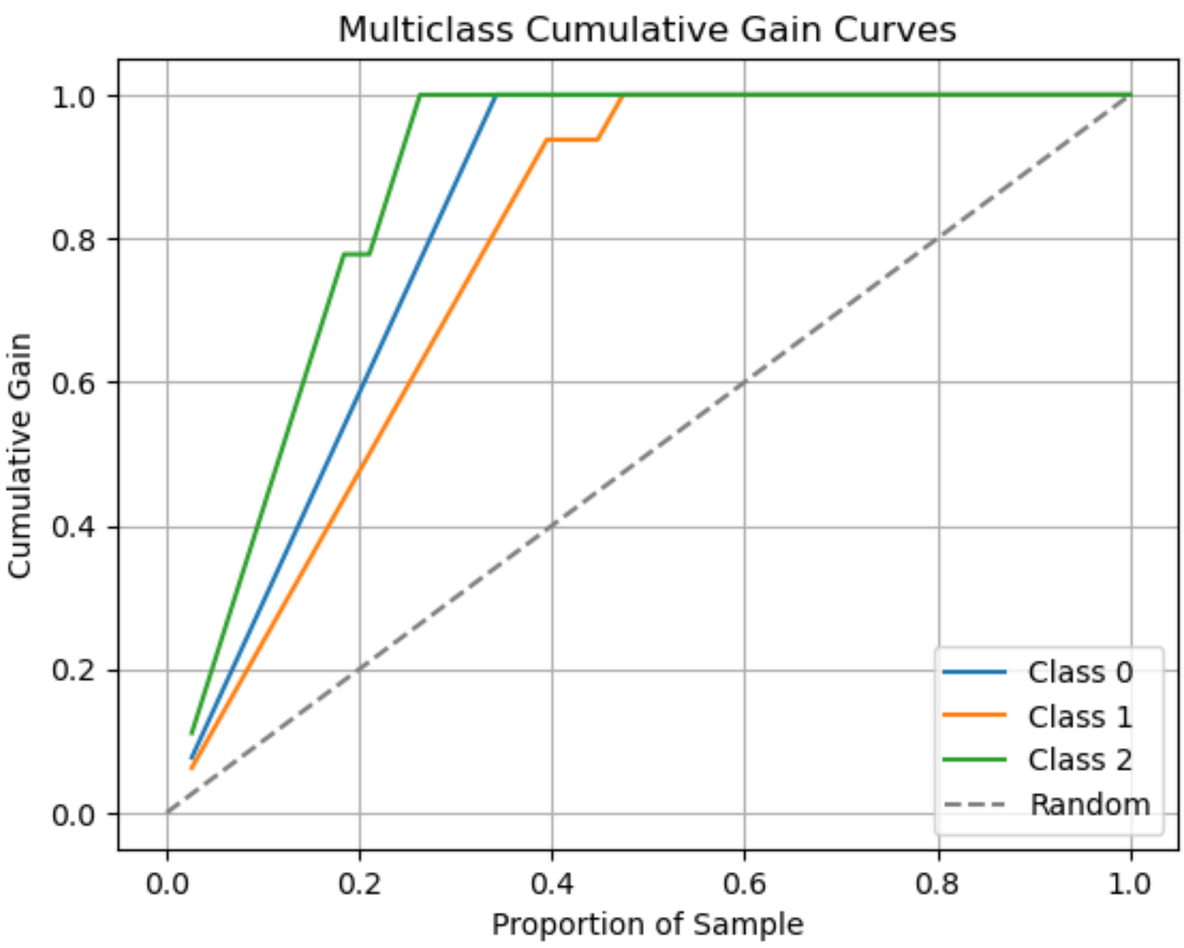
```
/Users/kjesta/miniconda3/envs/exercises/lib/python3.13/site-packages/sklea
rn/model_selection/_split.py:811: UserWarning: The least populated class i
n y has only 9 members, which is less than n_splits=10.
  warnings.warn(
```





In [27]:
```python
from sklearn.preprocessing import label_binarize
def multiclass_cumulative_gain(y_test, y_probas):
    classes = np.unique(y_test)

    y_true_bins = label_binarize(y_test, classes=classes)

    for i, cl in enumerate(classes):
        probability = y_probas[:, i]
        # Want to sort the probabilities in descending order
        sorted_indices = np.argsort(probability)[::-1]
        y_true_sorted = y_true_bins[sorted_indices, i]

        # Sum of cumulative true positives
        cum_sum_true = np.cumsum(y_true_sorted)
        # The cumulative gain
        cum_gain = cum_sum_true / np.sum(y_true_sorted)
        # Percentage of samples considered
        perc_population = np.arange(1, len(y_test) + 1) / len(y_test)

        plt.plot(perc_population, cum_gain, label=f"Class {cl}")

    # Random baseline
    plt.plot([0,1], [0,1], '--', color='gray', label="Random")

    plt.xlabel("Proportion of Sample")
    plt.ylabel("Cumulative Gain")
    plt.title("Multiclass Cumulative Gain Curves")
    plt.legend()
    plt.grid(True)
    plt.show()
```

In [28]:
```
multiclass_cumulative_gain(y_test, y_probas)
```


Multiclass Cumulative Gain Curves
Cumulative Gain
Proportion of Sample
Class 0
Class 1
Class 2
Random