Precision_Recall

November 23, 2021

[1]:
```
import pandas as pd
import numpy as np
import os
import glob
```

[2]:
```
label_index_mapping = {'10370003': 0, '111288001': 1, '11157007': 2,␣
↪'111975006': 3, '164861001': 4, '164865005': 5, '164867002': 6, '164873001':␣
↪7, '164884008': 8, '164889003': 9, '164890007': 10, '164895002': 11,␣
↪'164896001': 12, '164909002': 13, '164917005': 14, '164921003': 15,␣
↪'164930006': 16, '164931005': 17, '164934002': 18, '164937009': 19,␣
↪'164947007': 20, '164951009': 21, '17338001': 22, '195042002': 23,␣
↪'195060002': 24, '195080001': 25, '195101003': 26, '195126007': 27,␣
↪'233917008': 28, '251120003': 29, '251139008': 30, '251146004': 31,␣
↪'251164006': 32, '251170000': 33, '251180001': 34, '251200008': 35,␣
↪'251259000': 36, '251266004': 37, '251268003': 38, '253339007': 39,␣
↪'253352002': 40, '266249003': 41, '270492004': 42, '27885002': 43,␣
↪'284470004': 44, '29320008': 45, '370365005': 46, '39732003': 47,␣
↪'413444003': 48, '413844008': 49, '425419005': 50, '425623009': 51,␣
↪'426177001': 52, '426434006': 53, '426627000': 54, '426648003': 55,␣
↪'426664006': 56, '426749004': 57, '426761007': 58, '426783006': 59,␣
↪'426995002': 60, '427084000': 61, '427172004': 62, '427393009': 63,␣
↪'428417006': 64, '428750005': 65, '429622005': 66, '445118002': 67,␣
↪'445211001': 68, '446358003': 69, '446813000': 70, '47665007': 71,␣
↪'49578007': 72, '54329005': 73, '55930002': 74, '59118001': 75, '59931005':␣
↪76, '63593006': 77, '6374002': 78, '65778007': 79, '67198005': 80,␣
↪'67741000119109': 81, '698252002': 82, '704997005': 83, '713422000': 84,␣
↪'713426002': 85, '713427006': 86, '74390002': 87, '74615001': 88, '75532003':
↪ 89, '77867006': 90, '81898007': 91, '82226007': 92, '89792004': 93}
```

[14]:
```
#model_folder = 'models/02_03_21-15/architectures_cpc.cpc_combined.CPCCombined'
#model_folder = 'models/09_03_21-18/architectures_cpc.cpc_combined.CPCCombined'
#model_folder = 'models/10_03_21-18/architectures_cpc.cpc_combined.CPCCombined0'
model_folder = 'models/10_03_21-18/architectures_cpc.cpc_combined.CPCCombined1'
```

[15]:
```
pred_path = glob.glob(os.path.join(model_folder ,'*output.csv'))[0]
dfp = pd.read_csv(pred_path)
pred = dfp.values[:, 1:]
```

```
dfp
```

[15]:

	Unnamed: 0	10370003	111288001	11157007	111975006	164861001	164865005	164867002	164873001	164884008	…	713422000	713426002	713427006	74390002	74615001	75532003	77867006	81898007	82226007
0	/media/julian/data/data/ECG/ptbxl_challenge/HR…	2.087841e-04	3.226466e-05	0.003158	0.001773	0.068205	0.006216	0.063119	0.048559	0.026258	…	0.000174	0.000620	0.001715	2.824765e-05	0.000002	0.003151	2.040971e-07	1.931225e-05	0.000013
1	/media/julian/data/data/ECG/ptbxl_challenge/HR…	1.423953e-05	1.303028e-06	0.000152	0.004093	0.023717	0.041257	0.156696	0.000661	0.012075	…	0.000037	0.000125	0.001074	1.507377e-06	0.000104	0.000004	3.131340e-06	4.826618e-07	0.000012
2	/media/julian/data/data/ECG/ptbxl_challenge/HR…	6.672048e-07	3.538130e-07	0.000183	0.003039	0.016408	0.019658	0.011421	0.022186	0.004674	…	0.000022	0.000043	0.000179	1.572609e-06	0.000012	0.000001	1.688296e-06	1.273030e-07	0.000008
3	/media/julian/data/data/ECG/ptbxl_challenge/HR…	3.236617e-04	2.876266e-03	0.003877	0.002087	0.012676	0.046206	0.137421	0.261266	0.016994	…	0.000105	0.000273	0.000794	3.583556e-05	0.000029	0.000700	1.298575e-05	4.868621e-05	0.000199
4	/media/julian/data/data/ECG/ptbxl_challenge/HR…	3.624512e-06	3.865126e-06	0.000066	0.006995	0.003061	0.072974	0.009145	0.008809	0.001167	…	0.000053	0.000397	0.000153	1.913160e-05	0.000006	0.000028	6.346227e-06	2.312360e-06	0.000003
…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…
21832	/media/julian/data/data/ECG/ptbxl_challenge/HR…	8.246302e-05	4.419320e-06	0.000006	0.005282	0.155003	0.233155	0.298385	0.177913	0.002576	…	0.000239	0.006897	0.000196						
21833	/media/julian/data/data/ECG/ptbxl_challenge/HR…	4.787022e-05	2.276055e-05	0.000144	0.002249	0.073271	0.126099	0.583257	0.021028	0.018606	…	0.000107	0.000480	0.001905						
21834	/media/julian/data/data/ECG/ptbxl_challenge/HR…	1.655720e-06	7.314545e-07	0.000164	0.005206	0.025028	0.012917	0.018453	0.015832	0.009050	…	0.000092	0.000016	0.000349						
21835	/media/julian/data/data/ECG/ptbxl_challenge/HR…	1.398510e-06	4.683243e-07	0.000348	0.021632	0.020034	0.046151	0.053379	0.003649	0.003428	…	0.000091	0.000898	0.003881						
21836	/media/julian/data/data/ECG/ptbxl_challenge/HR…	1.177412e-05	6.864070e-07	0.000054	0.004394	0.165494	0.015996	0.024423	0.009429	0.001649	…	0.000022	0.000069	0.000540						

```
…              …        …        …             …             …        …
21832  2.492348e-06  0.000034  0.000032  7.154894e-06  8.904110e-05  0.000036
21833  9.453243e-06  0.000056  0.000020  2.109037e-05  2.250829e-06  0.000039
21834  8.145556e-07  0.000092  0.000002  2.701388e-07  6.755172e-07  0.000029
21835  7.537688e-06  0.000039  0.000007  4.998739e-06  1.383786e-07  0.000022
21836  1.773955e-07  0.000012  0.000001  5.049254e-07  2.647888e-07  0.000004

       89792004
0      0.000169
1      0.000088
2      0.000152
3      0.000520
4      0.000467
…         …
21832  0.002190
21833  0.000584
21834  0.000015
21835  0.000410
21836  0.000014

[21837 rows x 95 columns]
```

[16]:
```
label_path = glob.glob(os.path.join(model_folder ,'*labels*.csv'))[0]
dfl = pd.read_csv(label_path)
labels = dfl.values[:, 1:].astype(int)
dfl
```

[16]:
```
                                                Unnamed: 0  10370003  111288001  \
0      /media/julian/data/data/ECG/ptbxl_challenge/HR…         0.0        0.0
1      /media/julian/data/data/ECG/ptbxl_challenge/HR…         0.0        0.0
2      /media/julian/data/data/ECG/ptbxl_challenge/HR…         0.0        0.0
3      /media/julian/data/data/ECG/ptbxl_challenge/HR…         0.0        0.0
4      /media/julian/data/data/ECG/ptbxl_challenge/HR…         0.0        0.0
…                                                    …         …          …
21832  /media/julian/data/data/ECG/ptbxl_challenge/HR…         0.0        0.0
21833  /media/julian/data/data/ECG/ptbxl_challenge/HR…         0.0        0.0
21834  /media/julian/data/data/ECG/ptbxl_challenge/HR…         0.0        0.0
21835  /media/julian/data/data/ECG/ptbxl_challenge/HR…         0.0        0.0
21836  /media/julian/data/data/ECG/ptbxl_challenge/HR…         0.0        0.0

       11157007  111975006  164861001  164865005  164867002  164873001  \
0           0.0        0.0        1.0        0.0        0.0        0.0
1           0.0        0.0        0.0        0.0        0.0        0.0
2           0.0        0.0        0.0        0.0        0.0        0.0
3           0.0        0.0        0.0        0.0        0.0        1.0
4           0.0        0.0        0.0        0.0        0.0        0.0
…           …          …          …          …          …          …
```

```
21832       0.0        0.0        1.0        0.0        0.0        1.0
21833       0.0        0.0        0.0        0.0        0.0        0.0
21834       0.0        0.0        0.0        0.0        0.0        0.0
21835       0.0        0.0        0.0        1.0        0.0        0.0
21836       0.0        0.0        0.0        0.0        0.0        0.0

       164884008  …  713422000  713426002  713427006  74390002  74615001  \
0            0.0  …        0.0        0.0        0.0       0.0       0.0
1            0.0  …        0.0        0.0        0.0       0.0       0.0
2            0.0  …        0.0        0.0        0.0       0.0       0.0
3            0.0  …        0.0        0.0        0.0       0.0       0.0
4            0.0  …        0.0        0.0        0.0       0.0       0.0
…          …  …        …        …        …        …       …
21832        0.0  …        0.0        0.0        0.0       0.0       0.0
21833        0.0  …        0.0        0.0        0.0       0.0       0.0
21834        0.0  …        0.0        0.0        0.0       0.0       0.0
21835        0.0  …        0.0        0.0        0.0       0.0       0.0
21836        0.0  …        0.0        0.0        0.0       0.0       0.0

       75532003  77867006  81898007  82226007  89792004
0           0.0       0.0       0.0       0.0       0.0
1           0.0       0.0       0.0       0.0       0.0
2           0.0       0.0       0.0       0.0       0.0
3           0.0       0.0       0.0       0.0       0.0
4           0.0       0.0       0.0       0.0       0.0
…         …       …       …       …       …
21832       0.0       0.0       0.0       0.0       0.0
21833       0.0       0.0       0.0       0.0       0.0
21834       0.0       0.0       0.0       0.0       0.0
21835       0.0       0.0       0.0       0.0       0.0
21836       0.0       0.0       0.0       0.0       0.0

[21837 rows x 95 columns]
```

[17]:
```
np.set_printoptions(suppress=True)
np.round(np.sum(pred, axis=0).astype(float), 0)
```

[17]:
```
array([  10.,    1.,   27.,   87., 1566., 2038., 6476.,  836.,  295.,
       1179.,  168.,    3.,   28.,  353.,   17.,    1., 4126.,  276.,
        164.,    1.,    0.,    0.,   69.,   76.,   25.,  242.,    2.,
         15.,   18.,  379.,    1.,    7.,    4.,    5.,   13.,    0.,
          5.,    8.,   10.,    6.,   11.,   43.,  738.,  138.,  674.,
         37.,    1.,  115.,    2.,  635.,   29.,   36.,  795.,   10.,
       2392.,    5.,    1.,    6.,   18.,  559.,   22.,  855., 1265.,
         83.,    7., 5041.,  395.,   68.,    3.,   29.,  150.,    8.,
         15.,  141.,    1.,  224.,   69.,  335.,   21.,   45.,    0.,
         45.,   44.,    2.,   66.,  352.,  462.,    1.,    3.,   14.,
```

```
           2.,    4.,    4.,   41.])
```

[18]:
```
np.sum(labels, axis=0)
```

[18]:
```
array([  296,     0,    82,   118,  2175,  5261,     0,  2359,  1154,
        1514,    73,     0,     0,   536,   548,     0,     0,    28,
        2345,     0,   340,  3389,     0,    14,     0,     0,     0,
           0,     0,    77,     0,   182,     0,     0,    20,   156,
           0,     0,     0,     0,     0,    30,   797,    16,   398,
           0,     0,  5146,     0,     0,   219,   142,   637,    44,
           0,     0,     0,     0,    27, 18092,     0,   826,     0,
         772,     0,   381,  1009,  1626,   177,    99,     0,   343,
           0,   354,   770,     0,   294,   157,     0,     0,    24,
         427,   789,     0,     0,  1118,   542,    80,     0,     0,
           0,     0,     0,   126])
```

[19]:
```
n_classes = labels.shape[1]
n_classes
```

[19]:
```
94
```

0.1 Calculate ROC

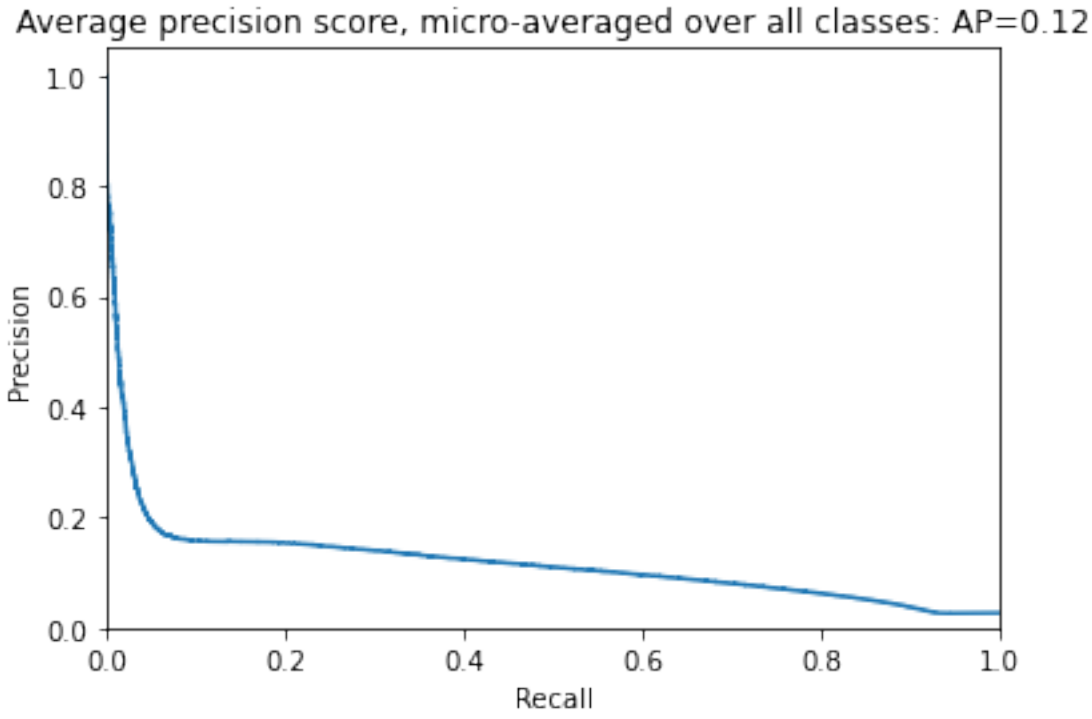
[28]:
```
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import average_precision_score

# For each class
precision = dict()
recall = dict()
average_precision = dict()
for i in range(n_classes):
    precision[i], recall[i], _ = precision_recall_curve(labels[:, i],
                                                        pred[:, i])
    average_precision[i] = average_precision_score(labels[:, i], pred[:, i])

# A "micro-average": quantifying score on all classes jointly
precision["micro"], recall["micro"], _ = precision_recall_curve(labels.ravel(),
    pred.ravel())
average_precision["micro"] = average_precision_score(labels, pred,
                                                     average="micro")
print('Average precision score, micro-averaged over all classes: {0:0.2f}'
      .format(average_precision["micro"]))
```

```
/home/julian/anaconda3/envs/ml/lib/python3.7/site-
packages/sklearn/metrics/_ranking.py:681: RuntimeWarning: invalid value
encountered in true_divide
  recall = tps / tps[-1]
```

```
Average precision score, micro-averaged over all classes: 0.12
```

[29]:
```
import matplotlib.pyplot as plt
def plot_precision_recall_microavg(recall, precision):
    plt.figure()
    plt.step(recall['micro'], precision['micro'], where='post')

    plt.xlabel('Recall')
    plt.ylabel('Precision')
    plt.ylim([0.0, 1.05])
    plt.xlim([0.0, 1.0])
    plt.title(
        'Average precision score, micro-averaged over all classes: AP={0:0.2f}'
        .format(average_precision["micro"]))
```

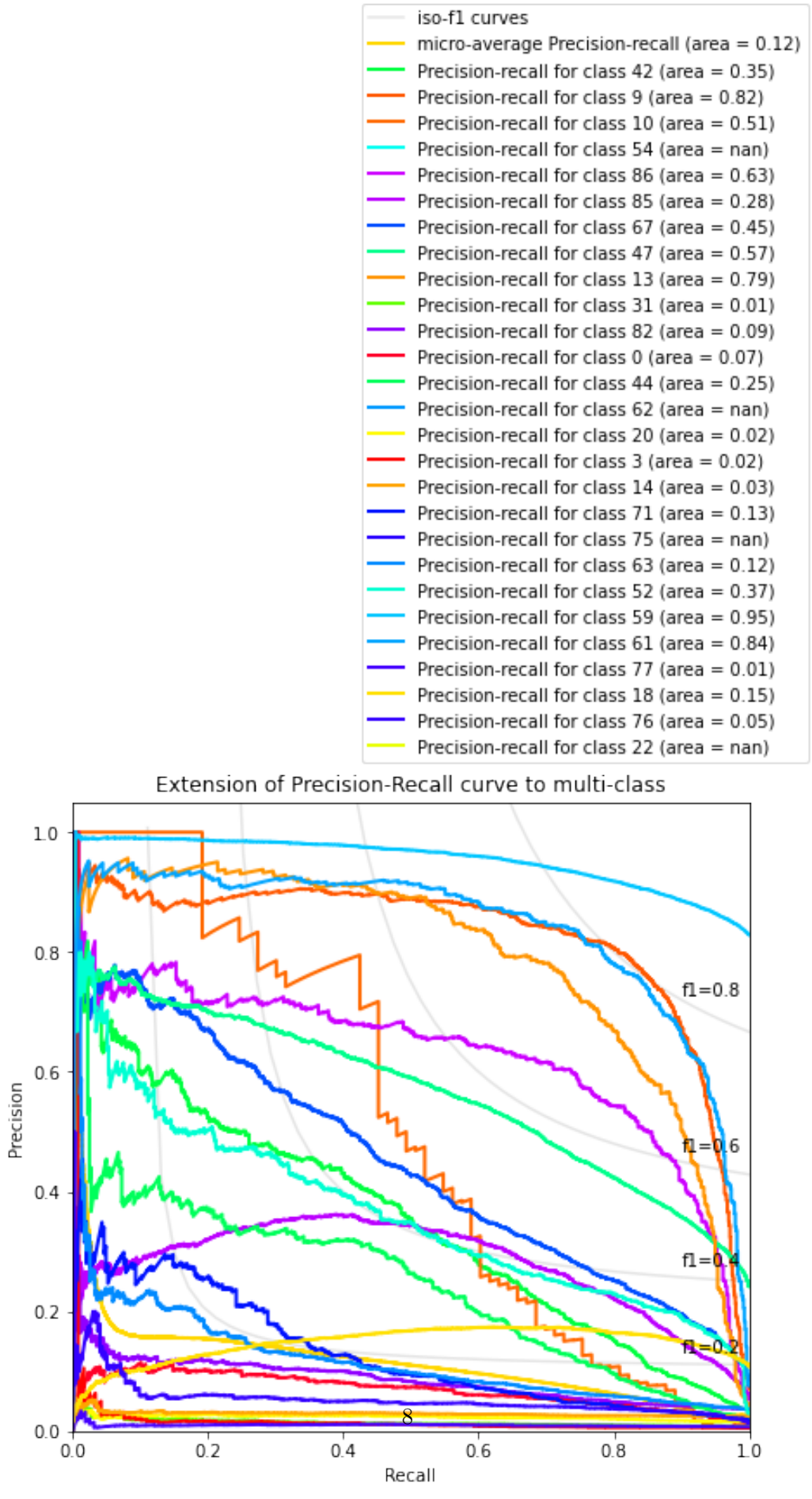
[30]:
```
plot_precision_recall_microavg(recall, precision)
```



[31]:
```
from itertools import cycle
def plot_precision_recall_multiclass(recall, precision, selection=None):
    cm = plt.get_cmap('gist_rainbow')
    if selection is None:
        selection = range(n_classes)
    # setup plot details
    colors = cycle([cm(1.*i/n_classes) for i in selection])
```

```
    plt.figure(figsize=(7, 8))
    f_scores = np.linspace(0.2, 0.8, num=4)
    lines = []
    labels = []
    for f_score in f_scores:
        x = np.linspace(0.01, 1)
        y = f_score * x / (2 * x - f_score)
        l, = plt.plot(x[y >= 0], y[y >= 0], color='gray', alpha=0.2)
        plt.annotate('f1={0:0.1f}'.format(f_score), xy=(0.9, y[45] + 0.02))

    lines.append(l)
    labels.append('iso-f1 curves')
    l, = plt.plot(recall["micro"], precision["micro"], color='gold', lw=2)
    lines.append(l)
    labels.append('micro-average Precision-recall (area = {0:0.2f})'
                  ''.format(average_precision["micro"]))

    for i, color in zip(selection, colors):
        l, = plt.plot(recall[i], precision[i], color=color, lw=2)
        lines.append(l)
        labels.append('Precision-recall for class {0} (area = {1:0.2f})'
                      ''.format(i, average_precision[i]))

    fig = plt.gcf()
    fig.subplots_adjust(bottom=0.25)
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('Recall')
    plt.ylabel('Precision')
    plt.title('Extension of Precision-Recall curve to multi-class')
    plt.legend(lines, labels, loc="lower right", bbox_to_anchor=(1.1, 1.05))


    plt.show()
```

[32]:
```
physionetchallenge_sel =␣
↪'270492004,164889003,164890007,426627000,713427006,713426002,445118002,39732003,164909002,2
↪split(',')
plot_precision_recall_multiclass(recall, precision,␣
↪selection=[label_index_mapping[snomed] for snomed in physionetchallenge_sel])
```


iso-f1 curves
micro-average Precision-recall (area = 0.12)
Precision-recall for class 42 (area = 0.35)
Precision-recall for class 9 (area = 0.82)
Precision-recall for class 10 (area = 0.51)
Precision-recall for class 54 (area = nan)
Precision-recall for class 86 (area = 0.63)
Precision-recall for class 85 (area = 0.28)
Precision-recall for class 67 (area = 0.45)
Precision-recall for class 47 (area = 0.57)
Precision-recall for class 13 (area = 0.79)
Precision-recall for class 31 (area = 0.01)
Precision-recall for class 82 (area = 0.09)
Precision-recall for class 0 (area = 0.07)
Precision-recall for class 44 (area = 0.25)
Precision-recall for class 62 (area = nan)
Precision-recall for class 20 (area = 0.02)
Precision-recall for class 3 (area = 0.02)
Precision-recall for class 14 (area = 0.03)
Precision-recall for class 71 (area = 0.13)
Precision-recall for class 75 (area = nan)
Precision-recall for class 63 (area = 0.12)
Precision-recall for class 52 (area = 0.37)
Precision-recall for class 59 (area = 0.95)
Precision-recall for class 61 (area = 0.84)
Precision-recall for class 77 (area = 0.01)
Precision-recall for class 18 (area = 0.15)
Precision-recall for class 76 (area = 0.05)
Precision-recall for class 22 (area = nan)
Extension of Precision-Recall curve to multi-class
f1=0.8
f1=0.6
f1=0.4
f1=0.2
Precision
Recall
0.0
0.2
0.4
0.6
0.8
1.0

[33]:
```
precision.keys()
```

[33]:
```
dict_keys([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19,
20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39,
40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59,
60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79,
80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 'micro'])
```

Below taken (and changed) from: https://github.com/physionetchallenges/evaluation-2020/blob/master/evaluate_12ECG_score.py

[34]:
```
#!/usr/bin/env python

# This file contains functions for evaluating algorithms for the 2020 PhysioNet/
# Computing in Cardiology Challenge. You can run it as follows:
#
#   python evaluate_12ECG_score.py labels outputs scores.csv
#
# where 'labels' is a directory containing files with the labels, 'outputs' is a
# directory containing files with the outputs from your model, and 'scores.csv'
# (optional) is a collection of scores for the algorithm outputs.
#
# Each file of labels or outputs must have the format described on the Challenge
# webpage. The scores for the algorithm outputs include the area under the
# receiver-operating characteristic curve (AUROC), the area under the recall-
# precision curve (AUPRC), accuracy (fraction of correct recordings), macro F-
# measure, and the Challenge metric, which assigns different weights to
# different misclassification errors.

import numpy as np, os, os.path, sys

def evaluate_12ECG_score(label_scores_data, label_binary_data, output_data):
    # Define the weights, the SNOMED CT code for the normal class, and␣
 ↪equivalent SNOMED CT codes.
    weights_file = 'weights.csv'
    normal_class = '426783006'
    equivalent_classes = [['713427006', '59118001'], ['284470004', '63593006'],␣
 ↪['427172004', '17338001']]

    # Load the scored classes and the weights for the Challenge metric.
    print('Loading weights...')
    classes, weights = load_weights(weights_file, equivalent_classes)

    # Load the label and output files.
    print('Loading label and output files...')
```

```
    binary_outputs, scalar_outputs = label_binary_data, label_scores_data

    # Evaluate the model by comparing the labels and outputs.
    print('Evaluating model...')

    print('- AUROC and AUPRC...')
    auroc, auprc, auroc_classes, auprc_classes = compute_auc(labels,␣
 ↪scalar_outputs)

    print('- Accuracy...')
    accuracy = compute_accuracy(labels, binary_outputs)

    print('- F-measure...')
    f_measure, f_measure_classes = compute_f_measure(labels, binary_outputs)

    print('- F-beta and G-beta measures...')
    f_beta_measure, g_beta_measure = compute_beta_measures(labels,␣
 ↪binary_outputs, beta=2)

    print('- Challenge metric...(skipped)')
    challenge_metric = -1.0 #compute_challenge_metric(weights, labels,␣
 ↪binary_outputs, classes, normal_class)

    print('Done.')

    # Return the results.
    return classes, auroc, auprc, auroc_classes, auprc_classes, accuracy,␣
 ↪f_measure, f_measure_classes, f_beta_measure, g_beta_measure,␣
 ↪challenge_metric

# Check if the input is a number.
def is_number(x):
    try:
        float(x)
        return True
    except ValueError:
        return False

# Load weights.
def load_weights(weight_file, equivalent_classes):
    # Load the weight matrix.
    rows, cols, values = load_table(weight_file)
    assert(rows == cols)

    # For each collection of equivalent classes, replace each class with the␣
 ↪representative class for the set.
```

```
    rows = replace_equivalent_classes(rows, equivalent_classes)

    # Check that equivalent classes have identical weights.
    for j, x in enumerate(rows):
        for k, y in enumerate(rows[j+1:]):
            if x==y:
                assert(np.all(values[j, :]==values[j+1+k, :]))
                assert(np.all(values[:, j]==values[:, j+1+k]))

    # Use representative classes.
    classes = [x for j, x in enumerate(rows) if x not in rows[:j]]
    indices = [rows.index(x) for x in classes]
    weights = values[np.ix_(indices, indices)]

    return classes, weights


# Compute recording-wise accuracy.
def compute_accuracy(labels, outputs):
    num_recordings, num_classes = np.shape(labels)

    num_correct_recordings = 0
    for i in range(num_recordings):
        if np.all(labels[i, :]==outputs[i, :]):
            num_correct_recordings += 1

    return float(num_correct_recordings) / float(num_recordings)

# Compute confusion matrices.
def compute_confusion_matrices(labels, outputs, normalize=False):
    # Compute a binary confusion matrix for each class k:
    #
    #     [TN_k FN_k]
    #     [FP_k TP_k]
    #
    # If the normalize variable is set to true, then normalize the contributions
    # to the confusion matrix by the number of labels per recording.
    num_recordings, num_classes = np.shape(labels)

    if not normalize:
        A = np.zeros((num_classes, 2, 2))
        for i in range(num_recordings):
            for j in range(num_classes):
                if labels[i, j]==1 and outputs[i, j]==1: # TP
                    A[j, 1, 1] += 1
                elif labels[i, j]==0 and outputs[i, j]==1: # FP
                    A[j, 1, 0] += 1
```

```
                elif labels[i, j]==1 and outputs[i, j]==0: # FN
                    A[j, 0, 1] += 1
                elif labels[i, j]==0 and outputs[i, j]==0: # TN
                    A[j, 0, 0] += 1
                else: # This condition should not happen.
                    raise ValueError('Error in computing the confusion matrix.')
    else:
        A = np.zeros((num_classes, 2, 2))
        for i in range(num_recordings):
            normalization = float(max(np.sum(labels[i, :]), 1))
            for j in range(num_classes):
                if labels[i, j]==1 and outputs[i, j]==1: # TP
                    A[j, 1, 1] += 1.0/normalization
                elif labels[i, j]==0 and outputs[i, j]==1: # FP
                    A[j, 1, 0] += 1.0/normalization
                elif labels[i, j]==1 and outputs[i, j]==0: # FN
                    A[j, 0, 1] += 1.0/normalization
                elif labels[i, j]==0 and outputs[i, j]==0: # TN
                    A[j, 0, 0] += 1.0/normalization
                else: # This condition should not happen.
                    raise ValueError('Error in computing the confusion matrix.')

    return A

# Compute macro F-measure.
def compute_f_measure(labels, outputs):
    num_recordings, num_classes = np.shape(labels)

    A = compute_confusion_matrices(labels, outputs)

    f_measure = np.zeros(num_classes)
    for k in range(num_classes):
        tp, fp, fn, tn = A[k, 1, 1], A[k, 1, 0], A[k, 0, 1], A[k, 0, 0]
        if 2 * tp + fp + fn:
            f_measure[k] = float(2 * tp) / float(2 * tp + fp + fn)
        else:
            f_measure[k] = float('nan')

    macro_f_measure = np.nanmean(f_measure)

    return macro_f_measure, f_measure

# Compute F-beta and G-beta measures from the unofficial phase of the Challenge.
def compute_beta_measures(labels, outputs, beta):
    num_recordings, num_classes = np.shape(labels)

    A = compute_confusion_matrices(labels, outputs, normalize=True)
```

```
    f_beta_measure = np.zeros(num_classes)
    g_beta_measure = np.zeros(num_classes)
    for k in range(num_classes):
        tp, fp, fn, tn = A[k, 1, 1], A[k, 1, 0], A[k, 0, 1], A[k, 0, 0]
        if (1+beta**2)*tp + fp + beta**2*fn:
            f_beta_measure[k] = float((1+beta**2)*tp) / float((1+beta**2)*tp +␣
 ↪fp + beta**2*fn)
        else:
            f_beta_measure[k] = float('nan')
        if tp + fp + beta*fn:
            g_beta_measure[k] = float(tp) / float(tp + fp + beta*fn)
        else:
            g_beta_measure[k] = float('nan')

    macro_f_beta_measure = np.nanmean(f_beta_measure)
    macro_g_beta_measure = np.nanmean(g_beta_measure)

    return macro_f_beta_measure, macro_g_beta_measure

# Compute macro AUROC and macro AUPRC.
def compute_auc(labels, outputs):
    num_recordings, num_classes = np.shape(labels)

    # Compute and summarize the confusion matrices for each class across at␣
 ↪distinct output values.
    auroc = np.zeros(num_classes)
    auprc = np.zeros(num_classes)

    for k in range(num_classes):
        # We only need to compute TPs, FPs, FNs, and TNs at distinct output␣
 ↪values.
        thresholds = np.unique(outputs[:, k])
        thresholds = np.append(thresholds, thresholds[-1]+1)
        thresholds = thresholds[::-1]
        num_thresholds = len(thresholds)

        # Initialize the TPs, FPs, FNs, and TNs.
        tp = np.zeros(num_thresholds)
        fp = np.zeros(num_thresholds)
        fn = np.zeros(num_thresholds)
        tn = np.zeros(num_thresholds)
        fn[0] = np.sum(labels[:, k]==1)
        tn[0] = np.sum(labels[:, k]==0)

        # Find the indices that result in sorted output values.
        idx = np.argsort(outputs[:, k])[::-1]
```

```
        # Compute the TPs, FPs, FNs, and TNs for class k across thresholds.
        i = 0
        for j in range(1, num_thresholds):
            # Initialize TPs, FPs, FNs, and TNs using values at previous␣
↪threshold.
            tp[j] = tp[j-1]
            fp[j] = fp[j-1]
            fn[j] = fn[j-1]
            tn[j] = tn[j-1]

            # Update the TPs, FPs, FNs, and TNs at i-th output value.
            while i < num_recordings and outputs[idx[i], k] >= thresholds[j]:
                if labels[idx[i], k]:
                    tp[j] += 1
                    fn[j] -= 1
                else:
                    fp[j] += 1
                    tn[j] -= 1
                i += 1

        # Summarize the TPs, FPs, FNs, and TNs for class k.
        tpr = np.zeros(num_thresholds)
        tnr = np.zeros(num_thresholds)
        ppv = np.zeros(num_thresholds)
        for j in range(num_thresholds):
            if tp[j] + fn[j]:
                tpr[j] = float(tp[j]) / float(tp[j] + fn[j])
            else:
                tpr[j] = float('nan')
            if fp[j] + tn[j]:
                tnr[j] = float(tn[j]) / float(fp[j] + tn[j])
            else:
                tnr[j] = float('nan')
            if tp[j] + fp[j]:
                ppv[j] = float(tp[j]) / float(tp[j] + fp[j])
            else:
                ppv[j] = float('nan')

        # Compute AUROC as the area under a piecewise linear function with TPR/
        # sensitivity (x-axis) and TNR/specificity (y-axis) and AUPRC as the␣
↪area
        # under a piecewise constant with TPR/recall (x-axis) and PPV/precision
        # (y-axis) for class k.
        for j in range(num_thresholds-1):
            auroc[k] += 0.5 * (tpr[j+1] - tpr[j]) * (tnr[j+1] + tnr[j])
            auprc[k] += (tpr[j+1] - tpr[j]) * ppv[j+1]
```

```
    # Compute macro AUROC and macro AUPRC across classes.
    macro_auroc = np.nanmean(auroc)
    macro_auprc = np.nanmean(auprc)

    return macro_auroc, macro_auprc, auroc, auprc

# Compute modified confusion matrix for multi-class, multi-label tasks.
def compute_modified_confusion_matrix(labels, outputs):
    # Compute a binary multi-class, multi-label confusion matrix, where the rows
    # are the labels and the columns are the outputs.
    num_recordings, num_classes = np.shape(labels)
    A = np.zeros((num_classes, num_classes))

    # Iterate over all of the recordings.
    for i in range(num_recordings):
        # Calculate the number of positive labels and/or outputs.
        normalization = float(max(np.sum(np.any((labels[i, :], outputs[i, :]),␣
 ↪axis=0)), 1))
        # Iterate over all of the classes.
        for j in range(num_classes):
            # Assign full and/or partial credit for each positive class.
            if labels[i, j]:
                for k in range(num_classes):
                    if outputs[i, k]:
                        A[j, k] += 1.0/normalization

    return A

# Compute the evaluation metric for the Challenge.
def compute_challenge_metric(weights, labels, outputs, classes, normal_class):
    num_recordings, num_classes = np.shape(labels)
    normal_index = classes.index(normal_class)

    # Compute the observed score.
    A = compute_modified_confusion_matrix(labels, outputs)
    observed_score = np.nansum(weights * A)

    # Compute the score for the model that always chooses the correct label(s).
    correct_outputs = labels
    A = compute_modified_confusion_matrix(labels, correct_outputs)
    correct_score = np.nansum(weights * A)

    # Compute the score for the model that always chooses the normal class.
    inactive_outputs = np.zeros((num_recordings, num_classes), dtype=np.bool)
    inactive_outputs[:, normal_index] = 1
    A = compute_modified_confusion_matrix(labels, inactive_outputs)
```

```
    inactive_score = np.nansum(weights * A)

    if correct_score != inactive_score:
        normalized_score = float(observed_score - inactive_score) /␣
↪float(correct_score - inactive_score)
    else:
        normalized_score = 0.0

    return normalized_score
```

[ ]: