

15a949460d ▾

...

scikit-learn / sklearn / metrics / _classification.py / <> Jump to ▾



agamemnonc DOC Balanced accuracy score adjusted doc fix (#19309)



46 contributors



+29

2510 lines (2063 sloc) | 94.9 KB

...

```
1  """Metrics to assess performance on classification task given class prediction.
2
3  Functions named as ``*_score`` return a scalar value to maximize: the higher
4  the better.
5
6  Function named as ``*_error`` or ``*_loss`` return a scalar value to minimize:
7  the lower the better.
8  """
9
10 # Authors: Alexandre Gramfort <alexandre.gramfort@inria.fr>
11 #           Mathieu Blondel <mathieu@mblondel.org>
12 #           Olivier Grisel <olivier.grisel@ensta.org>
13 #           Arnaud Joly <a.joly@ulg.ac.be>
14 #           Jochen Wersdorfer <jochen@wersdoerfer.de>
15 #           Lars Buitinck
16 #           Joel Nothman <joel.nothman@gmail.com>
17 #           Noel Dawe <noel@dawe.me>
18 #           Jatin Shah <jatindshah@gmail.com>
19 #           Saurabh Jha <saurabh.jhaa@gmail.com>
20 #           Bernardo Stein <bernardovstein@gmail.com>
21 #           Shangwu Yao <shangwuyao@gmail.com>
22 # License: BSD 3 clause
23
24
25 import warnings
26 import numpy as np
27
28 from scipy.sparse import coo_matrix
29 from scipy.sparse import csr_matrix
30
31 from ..preprocessing import LabelBinarizer
32 from ..preprocessing import LabelEncoder
```

```

33 from ..utils import assert_all_finite
34 from ..utils import check_array
35 from ..utils import check_consistent_length
36 from ..utils import column_or_1d
37 from ..utils.multiclass import unique_labels
38 from ..utils.multiclass import type_of_target
39 from ..utils.validation import _num_samples
40 from ..utils.validation import _deprecate_positional_args
41 from ..utils.sparsefuncs import count_nonzero
42 from ..exceptions import UndefinedMetricWarning
43
44 from ._base import _check_pos_label_consistency
45
46
47 def _check_zero_division(zero_division):
48     if isinstance(zero_division, str) and zero_division == "warn":
49         return
50     elif isinstance(zero_division, (int, float)) and zero_division in [0, 1]:
51         return
52     raise ValueError('Got zero_division={0}.'
53                      ' Must be one of ["warn", 0, 1]'.format(zero_division))
54
55
56 def _check_targets(y_true, y_pred):
57     """Check that y_true and y_pred belong to the same classification task.
58
59     This converts multiclass or binary types to a common shape, and raises a
60     ValueError for a mix of multilabel and multiclass targets, a mix of
61     multilabel formats, for the presence of continuous-valued or multioutput
62     targets, or for targets of different lengths.
63
64     Column vectors are squeezed to 1d, while multilabel formats are returned
65     as CSR sparse label indicators.
66
67     Parameters
68     -----
69     y_true : array-like
70
71     y_pred : array-like
72
73     Returns
74     -----
75     type_true : one of {'multilabel-indicator', 'multiclass', 'binary'}
76         The type of the true target data, as output by
77         ``utils.multiclass.type_of_target``.
78
79     y_true : array or indicator matrix
80
81     y_pred : array or indicator matrix
82     """
83     check_consistent_length(y_true, y_pred)
84     type_true = type_of_target(y_true)

```

```

85     type_pred = type_of_target(y_pred)
86
87     y_type = {type_true, type_pred}
88     if y_type == {"binary", "multiclass"}:
89         y_type = {"multiclass"}
90
91     if len(y_type) > 1:
92         raise ValueError("Classification metrics can't handle a mix of {0} "
93                           "and {1} targets".format(type_true, type_pred))
94
95     # We can't have more than one value on y_type => The set is no more needed
96     y_type = y_type.pop()
97
98     # No metrics support "multiclass-multioutput" format
99     if (y_type not in ["binary", "multiclass", "multilabel-indicator"]):
100         raise ValueError("{0} is not supported".format(y_type))
101
102     if y_type in ["binary", "multiclass"]:
103         y_true = column_or_1d(y_true)
104         y_pred = column_or_1d(y_pred)
105         if y_type == "binary":
106             try:
107                 unique_values = np.union1d(y_true, y_pred)
108             except TypeError as e:
109                 # We expect y_true and y_pred to be of the same data type.
110                 # If `y_true` was provided to the classifier as strings,
111                 # `y_pred` given by the classifier will also be encoded with
112                 # strings. So we raise a meaningful error
113                 raise TypeError(
114                     f"Labels in y_true and y_pred should be of the same type. "
115                     f"Got y_true={np.unique(y_true)} and "
116                     f"y_pred={np.unique(y_pred)}. Make sure that the "
117                     f"predictions provided by the classifier coincides with "
118                     f"the true labels."
119                 ) from e
120                 if len(unique_values) > 2:
121                     y_type = "multiclass"
122
123     if y_type.startswith('multilabel'):
124         y_true = csr_matrix(y_true)
125         y_pred = csr_matrix(y_pred)
126         y_type = 'multilabel-indicator'
127
128     return y_type, y_true, y_pred
129
130
131 def _weighted_sum(sample_score, sample_weight, normalize=False):
132     if normalize:
133         return np.average(sample_score, weights=sample_weight)
134     elif sample_weight is not None:
135         return np.dot(sample_score, sample_weight)
136     else:

```

```

137         return sample_score.sum()
138
139
140 @_deprecated_positional_args
141 def accuracy_score(y_true, y_pred, *, normalize=True, sample_weight=None):
142     """Accuracy classification score.
143
144     In multilabel classification, this function computes subset accuracy:
145     the set of labels predicted for a sample must exactly match the
146     corresponding set of labels in y_true.
147
148     Read more in the :ref:`User Guide <accuracy_score>`.
149
150     Parameters
151     -----
152     y_true : 1d array-like, or label indicator array / sparse matrix
153             Ground truth (correct) labels.
154
155     y_pred : 1d array-like, or label indicator array / sparse matrix
156             Predicted labels, as returned by a classifier.
157
158     normalize : bool, default=True
159             If ``False``, return the number of correctly classified samples.
160             Otherwise, return the fraction of correctly classified samples.
161
162     sample_weight : array-like of shape (n_samples,), default=None
163             Sample weights.
164
165     Returns
166     -----
167     score : float
168             If ``normalize == True``, return the fraction of correctly
169             classified samples (float), else returns the number of correctly
170             classified samples (int).
171
172             The best performance is 1 with ``normalize == True`` and the number
173             of samples with ``normalize == False``.
174
175     See Also
176     -----
177     jaccard_score, hamming_loss, zero_one_loss
178
179     Notes
180     -----
181     In binary and multiclass classification, this function is equal
182     to the ``jaccard_score`` function.
183
184     Examples
185     -----
186     >>> from sklearn.metrics import accuracy_score
187
188     >>> y_pred = [0, 2, 1, 3]
189     >>> y_true = [0, 1, 2, 3]

```

```

189 >>> accuracy_score(y_true, y_pred)
190 0.5
191 >>> accuracy_score(y_true, y_pred, normalize=False)
192 2
193
194 In the multilabel case with binary label indicators:
195
196 >>> import numpy as np
197 >>> accuracy_score(np.array([[0, 1], [1, 1]]), np.ones((2, 2)))
198 0.5
199 """
200
201 # Compute accuracy for each possible representation
202 y_type, y_true, y_pred = _check_targets(y_true, y_pred)
203 check_consistent_length(y_true, y_pred, sample_weight)
204 if y_type.startswith('multilabel'):
205     differing_labels = count_nonzero(y_true - y_pred, axis=1)
206     score = differing_labels == 0
207 else:
208     score = y_true == y_pred
209
210 return _weighted_sum(score, sample_weight, normalize)
211
212
213 @_deprecate_positional_args
214 def confusion_matrix(y_true, y_pred, *, labels=None, sample_weight=None,
215                     normalize=None):
216     """Compute confusion matrix to evaluate the accuracy of a classification.
217
218     By definition a confusion matrix  $C$  is such that  $C_{i,j}$ 
219     is equal to the number of observations known to be in group  $i$  and
220     predicted to be in group  $j$ .
221
222     Thus in binary classification, the count of true negatives is
223      $C_{0,0}$ , false negatives is  $C_{1,0}$ , true positives is
224      $C_{1,1}$  and false positives is  $C_{0,1}$ .
225
226     Read more in the :ref:`User Guide <confusion_matrix>`.
227
228     Parameters
229     -----
230     y_true : array-like of shape (n_samples,)
231         Ground truth (correct) target values.
232
233     y_pred : array-like of shape (n_samples,)
234         Estimated targets as returned by a classifier.
235
236     labels : array-like of shape (n_classes), default=None
237         List of labels to index the matrix. This may be used to reorder
238         or select a subset of labels.
239
240         If ``None`` is given, those that appear at least once
241         in ``y_true`` or ``y_pred`` are used in sorted order.

```

```
sample_weight : array-like of shape (n_samples,), default=None
    Sample weights.

    .. versionadded:: 0.18

normalize : {'true', 'pred', 'all'}, default=None
    Normalizes confusion matrix over the true (rows), predicted (columns)
    conditions or all the population. If None, confusion matrix will not be
    normalized.
```

Returns

```
-----
C : ndarray of shape (n_classes, n_classes)
    Confusion matrix whose i-th row and j-th
    column entry indicates the number of
    samples with true label being i-th class
    and predicted label being j-th class.
```

See Also

```
-----
ConfusionMatrixDisplay.from_estimator : Plot the confusion matrix
    given an estimator, the data, and the label.
ConfusionMatrixDisplay.from_predictions : Plot the confusion matrix
    given the true and predicted labels.
ConfusionMatrixDisplay : Confusion Matrix visualization.
```

References

```
-----
.. [1] `Wikipedia entry for the Confusion matrix
    <https://en.wikipedia.org/wiki/Confusion\_matrix>`_
    (Wikipedia and other references may use a different
    convention for axes).
```

Examples

```
-----
>>> from sklearn.metrics import confusion_matrix
>>> y_true = [2, 0, 2, 2, 0, 1]
>>> y_pred = [0, 0, 2, 2, 0, 2]
>>> confusion_matrix(y_true, y_pred)
array([[2, 0, 0],
       [0, 0, 1],
       [1, 0, 2]])

>>> y_true = ["cat", "ant", "cat", "cat", "ant", "bird"]
>>> y_pred = ["ant", "ant", "cat", "cat", "ant", "cat"]
>>> confusion_matrix(y_true, y_pred, labels=["ant", "bird", "cat"])
array([[2, 0, 0],
       [0, 0, 1],
       [1, 0, 2]])
```

In the binary case, we can extract true positives, etc as follows:

```

293
294 >>> tn, fp, fn, tp = confusion_matrix([0, 1, 0, 1], [1, 1, 1, 0]).ravel()
295 >>> (tn, fp, fn, tp)
296 (0, 2, 1, 1)
297
298 """
299 y_type, y_true, y_pred = _check_targets(y_true, y_pred)
300 if y_type not in ("binary", "multiclass"):
301     raise ValueError("%s is not supported" % y_type)
302
303 if labels is None:
304     labels = unique_labels(y_true, y_pred)
305 else:
306     labels = np.asarray(labels)
307     n_labels = labels.size
308     if n_labels == 0:
309         raise ValueError("'labels' should contains at least one label.")
310     elif y_true.size == 0:
311         return np.zeros((n_labels, n_labels), dtype=int)
312     elif np.all([l not in y_true for l in labels]):
313         raise ValueError("At least one label specified must be in y_true")
314
315 if sample_weight is None:
316     sample_weight = np.ones(y_true.shape[0], dtype=np.int64)
317 else:
318     sample_weight = np.asarray(sample_weight)
319
320 check_consistent_length(y_true, y_pred, sample_weight)
321
322 if normalize not in ['true', 'pred', 'all', None]:
323     raise ValueError("normalize must be one of {'true', 'pred', "
324                     "'all', None}")
325
326 n_labels = labels.size
327 label_to_ind = {y: x for x, y in enumerate(labels)}
328 # convert yt, yp into index
329 y_pred = np.array([label_to_ind.get(x, n_labels + 1) for x in y_pred])
330 y_true = np.array([label_to_ind.get(x, n_labels + 1) for x in y_true])
331
332 # intersect y_pred, y_true with labels, eliminate items not in labels
333 ind = np.logical_and(y_pred < n_labels, y_true < n_labels)
334 y_pred = y_pred[ind]
335 y_true = y_true[ind]
336 # also eliminate weights of eliminated items
337 sample_weight = sample_weight[ind]
338
339 # Choose the accumulator dtype to always have high precision
340 if sample_weight.dtype.kind in {'i', 'u', 'b'}:
341     dtype = np.int64
342 else:
343     dtype = np.float64
344

```

```

345     cm = coo_matrix((sample_weight, (y_true, y_pred)),
346                     shape=(n_labels, n_labels), dtype=dtype,
347                     ).toarray()
348
349     with np.errstate(all='ignore'):
350         if normalize == 'true':
351             cm = cm / cm.sum(axis=1, keepdims=True)
352         elif normalize == 'pred':
353             cm = cm / cm.sum(axis=0, keepdims=True)
354         elif normalize == 'all':
355             cm = cm / cm.sum()
356         cm = np.nan_to_num(cm)
357
358     return cm
359
360
361 @_deprecate_positional_args
362 def multilabel_confusion_matrix(y_true, y_pred, *, sample_weight=None,
363                                labels=None, samplewise=False):
364     """Compute a confusion matrix for each class or sample.
365
366     .. versionadded:: 0.21
367
368     Compute class-wise (default) or sample-wise (samplewise=True) multilabel
369     confusion matrix to evaluate the accuracy of a classification, and output
370     confusion matrices for each class or sample.
371
372     In multilabel confusion matrix :math:`MCM`, the count of true negatives
373     is :math:`MCM_{(:,0,0)}`, false negatives is :math:`MCM_{(:,1,0)}`,
374     true positives is :math:`MCM_{(:,1,1)}` and false positives is
375     :math:`MCM_{(:,0,1)}`.
376
377     Multiclass data will be treated as if binarized under a one-vs-rest
378     transformation. Returned confusion matrices will be in the order of
379     sorted unique labels in the union of (y_true, y_pred).
380
381     Read more in the :ref:`User Guide <multilabel_confusion_matrix>`.
382
383     Parameters
384     -----
385     y_true : {array-like, sparse matrix} of shape (n_samples, n_outputs) or \
386             (n_samples,)
387             Ground truth (correct) target values.
388
389     y_pred : {array-like, sparse matrix} of shape (n_samples, n_outputs) or \
390             (n_samples,)
391             Estimated targets as returned by a classifier.
392
393     sample_weight : array-like of shape (n_samples,), default=None
394                     Sample weights.
395
396     labels : array-like of shape (n_classes,), default=None

```



```

397     A list of classes or column indices to select some (or to force
398     inclusion of classes absent from the data).
399
400 samplewise : bool, default=False
401     In the multilabel case, this calculates a confusion matrix per sample.
402
403 Returns
404 -----
405 multi_confusion : ndarray of shape (n_outputs, 2, 2)
406     A 2x2 confusion matrix corresponding to each output in the input.
407     When calculating class-wise multi_confusion (default), then
408     n_outputs = n_labels; when calculating sample-wise multi_confusion
409     (samplewise=True), n_outputs = n_samples. If ``labels`` is defined,
410     the results will be returned in the order specified in ``labels``,
411     otherwise the results will be returned in sorted order by default.
412
413 See Also
414 -----
415 confusion_matrix
416
417 Notes
418 -----
419 The multilabel_confusion_matrix calculates class-wise or sample-wise
420 multilabel confusion matrices, and in multiclass tasks, labels are
421 binarized under a one-vs-rest way; while confusion_matrix calculates
422 one confusion matrix for confusion between every two classes.
423
424 Examples
425 -----
426 Multilabel-indicator case:
427
428 >>> import numpy as np
429 >>> from sklearn.metrics import multilabel_confusion_matrix
430 >>> y_true = np.array([[1, 0, 1],
431 ...                    [0, 1, 0]])
432 >>> y_pred = np.array([[1, 0, 0],
433 ...                    [0, 1, 1]])
434 >>> multilabel_confusion_matrix(y_true, y_pred)
435 array([[[1, 0],
436         [0, 1]],
437 <BLANKLINE>
438        [[1, 0],
439         [0, 1]],
440 <BLANKLINE>
441        [[0, 1],
442         [1, 0]])]
443
444 Multiclass case:
445
446 >>> y_true = ["cat", "ant", "cat", "cat", "ant", "bird"]
447 >>> y_pred = ["ant", "ant", "cat", "cat", "ant", "cat"]
448 >>> multilabel_confusion_matrix(y_true, y_pred,

```

```

449         ...                                labels=["ant", "bird", "cat"])
450     array([[3, 1],
451           [0, 2]],
452     <BLANKLINE>
453           [[5, 0],
454           [1, 0]],
455     <BLANKLINE>
456           [[2, 1],
457           [1, 2]]])
458     """
459     y_type, y_true, y_pred = _check_targets(y_true, y_pred)
460     if sample_weight is not None:
461         sample_weight = column_or_1d(sample_weight)
462         check_consistent_length(y_true, y_pred, sample_weight)
463
464     if y_type not in ("binary", "multiclass", "multilabel-indicator"):
465         raise ValueError("%s is not supported" % y_type)
466
467     present_labels = unique_labels(y_true, y_pred)
468     if labels is None:
469         labels = present_labels
470         n_labels = None
471     else:
472         n_labels = len(labels)
473         labels = np.hstack([labels, np.setdiff1d(present_labels, labels,
474                                                  assume_unique=True)])
475
476     if y_true.ndim == 1:
477         if samplewise:
478             raise ValueError("Samplewise metrics are not available outside of "
479                               "multilabel classification.")
480
481         le = LabelEncoder()
482         le.fit(labels)
483         y_true = le.transform(y_true)
484         y_pred = le.transform(y_pred)
485         sorted_labels = le.classes_
486
487         # labels are now from 0 to len(labels) - 1 -> use bincount
488         tp = y_true == y_pred
489         tp_bins = y_true[tp]
490         if sample_weight is not None:
491             tp_bins_weights = np.asarray(sample_weight)[tp]
492         else:
493             tp_bins_weights = None
494
495         if len(tp_bins):
496             tp_sum = np.bincount(tp_bins, weights=tp_bins_weights,
497                                   minlength=len(labels))
498         else:
499             # Pathological case
500             true_sum = pred_sum = tp_sum = np.zeros(len(labels))

```

```

501     if len(y_pred):
502         pred_sum = np.bincount(y_pred, weights=sample_weight,
503                                minlength=len(labels))
504     if len(y_true):
505         true_sum = np.bincount(y_true, weights=sample_weight,
506                                minlength=len(labels))
507
508     # Retain only selected labels
509     indices = np.searchsorted(sorted_labels, labels[:n_labels])
510     tp_sum = tp_sum[indices]
511     true_sum = true_sum[indices]
512     pred_sum = pred_sum[indices]
513
514 else:
515     sum_axis = 1 if samplewise else 0
516
517     # All labels are index integers for multilabel.
518     # Select labels:
519     if not np.array_equal(labels, present_labels):
520         if np.max(labels) > np.max(present_labels):
521             raise ValueError('All labels must be in [0, n labels) for '
522                               'multilabel targets. '
523                               'Got %d > %d' %
524                               (np.max(labels), np.max(present_labels)))
525         if np.min(labels) < 0:
526             raise ValueError('All labels must be in [0, n labels) for '
527                               'multilabel targets. '
528                               'Got %d < 0' % np.min(labels))
529
530     if n_labels is not None:
531         y_true = y_true[:, labels[:n_labels]]
532         y_pred = y_pred[:, labels[:n_labels]]
533
534     # calculate weighted counts
535     true_and_pred = y_true.multiply(y_pred)
536     tp_sum = count_nonzero(true_and_pred, axis=sum_axis,
537                            sample_weight=sample_weight)
538     pred_sum = count_nonzero(y_pred, axis=sum_axis,
539                              sample_weight=sample_weight)
540     true_sum = count_nonzero(y_true, axis=sum_axis,
541                              sample_weight=sample_weight)
542
543     fp = pred_sum - tp_sum
544     fn = true_sum - tp_sum
545     tp = tp_sum
546
547     if sample_weight is not None and samplewise:
548         sample_weight = np.array(sample_weight)
549         tp = np.array(tp)
550         fp = np.array(fp)
551         fn = np.array(fn)
552         tn = sample_weight * y_true.shape[1] - tp - fp - fn

```

```

553     elif sample_weight is not None:
554         tn = sum(sample_weight) - tp - fp - fn
555     elif samplewise:
556         tn = y_true.shape[1] - tp - fp - fn
557     else:
558         tn = y_true.shape[0] - tp - fp - fn
559
560     return np.array([tn, fp, fn, tp]).T.reshape(-1, 2, 2)
561
562
563 @_deprecate_positional_args
564 def cohen_kappa_score(y1, y2, *, labels=None, weights=None,
565                      sample_weight=None):
566     r"""Cohen's kappa: a statistic that measures inter-annotator agreement.
567
568     This function computes Cohen's kappa [1]_, a score that expresses the level
569     of agreement between two annotators on a classification problem. It is
570     defined as
571
572     .. math::
573         \kappa = (p_o - p_e) / (1 - p_e)
574
575     where :math:`p_o` is the empirical probability of agreement on the label
576     assigned to any sample (the observed agreement ratio), and :math:`p_e` is
577     the expected agreement when both annotators assign labels randomly.
578     :math:`p_e` is estimated using a per-annotator empirical prior over the
579     class labels [2]_.
580
581     Read more in the :ref:`User Guide <cohen_kappa>`.
582
583     Parameters
584     -----
585     y1 : array of shape (n_samples,)
586         Labels assigned by the first annotator.
587
588     y2 : array of shape (n_samples,)
589         Labels assigned by the second annotator. The kappa statistic is
590         symmetric, so swapping ``y1`` and ``y2`` doesn't change the value.
591
592     labels : array-like of shape (n_classes,), default=None
593         List of labels to index the matrix. This may be used to select a
594         subset of labels. If None, all labels that appear at least once in
595         ``y1`` or ``y2`` are used.
596
597     weights : {'linear', 'quadratic'}, default=None
598         Weighting type to calculate the score. None means no weighted;
599         "linear" means linear weighted; "quadratic" means quadratic weighted.
600
601     sample_weight : array-like of shape (n_samples,), default=None
602         Sample weights.
603
604     Returns

```

```

-----
kappa : float
    The kappa statistic, which is a number between -1 and 1. The maximum
    value means complete agreement; zero or lower means chance agreement.

References
-----
.. [1] J. Cohen (1960). "A coefficient of agreement for nominal scales".
    Educational and Psychological Measurement 20(1):37-46.
    doi:10.1177/001316446002000104.
.. [2] `R. Artstein and M. Poesio (2008). "Inter-coder agreement for
    computational linguistics". Computational Linguistics 34(4):555-596
    <https://www.mitpressjournals.org/doi/pdf/10.1162/coli.07-034-R2>`.
.. [3] `Wikipedia entry for the Cohen's kappa
    <https://en.wikipedia.org/wiki/Cohen%27s_kappa>`.
"""
confusion = confusion_matrix(y1, y2, labels=labels,
                             sample_weight=sample_weight)
n_classes = confusion.shape[0]
sum0 = np.sum(confusion, axis=0)
sum1 = np.sum(confusion, axis=1)
expected = np.outer(sum0, sum1) / np.sum(sum0)

if weights is None:
    w_mat = np.ones([n_classes, n_classes], dtype=int)
    w_mat.flat[: n_classes + 1] = 0
elif weights == "linear" or weights == "quadratic":
    w_mat = np.zeros([n_classes, n_classes], dtype=int)
    w_mat += np.arange(n_classes)
    if weights == "linear":
        w_mat = np.abs(w_mat - w_mat.T)
    else:
        w_mat = (w_mat - w_mat.T) ** 2
else:
    raise ValueError("Unknown kappa weighting type.")

k = np.sum(w_mat * confusion) / np.sum(w_mat * expected)
return 1 - k

@_deprecated_positional_args
def jaccard_score(y_true, y_pred, *, labels=None, pos_label=1,
                  average='binary', sample_weight=None, zero_division="warn"):
    """Jaccard similarity coefficient score.

    The Jaccard index [1], or Jaccard similarity coefficient, defined as
    the size of the intersection divided by the size of the union of two label
    sets, is used to compare set of predicted labels for a sample to the
    corresponding set of labels in ``y_true``.

    Read more in the :ref:`User Guide <jaccard_similarity_score>`.

```

```

Parameters
-----
y_true : 1d array-like, or label indicator array / sparse matrix
    Ground truth (correct) labels.

y_pred : 1d array-like, or label indicator array / sparse matrix
    Predicted labels, as returned by a classifier.

labels : array-like of shape (n_classes,), default=None
    The set of labels to include when ``average != 'binary'``, and their
    order if ``average is None``. Labels present in the data can be
    excluded, for example to calculate a multiclass average ignoring a
    majority negative class, while labels not present in the data will
    result in 0 components in a macro average. For multilabel targets,
    labels are column indices. By default, all labels in ``y_true`` and
    ``y_pred`` are used in sorted order.

pos_label : str or int, default=1
    The class to report if ``average='binary'`` and the data is binary.
    If the data are multiclass or multilabel, this will be ignored;
    setting ``labels=[pos_label]`` and ``average != 'binary'`` will report
    scores for that label only.

average : {None, 'micro', 'macro', 'samples', 'weighted', \
    'binary'}, default='binary'
    If ``None``, the scores for each class are returned. Otherwise, this
    determines the type of averaging performed on the data:

    ``'binary'``:
        Only report results for the class specified by ``pos_label``.
        This is applicable only if targets (``y_{true,pred}``) are binary.
    ``'micro'``:
        Calculate metrics globally by counting the total true positives,
        false negatives and false positives.
    ``'macro'``:
        Calculate metrics for each label, and find their unweighted
        mean. This does not take label imbalance into account.
    ``'weighted'``:
        Calculate metrics for each label, and find their average, weighted
        by support (the number of true instances for each label). This
        alters 'macro' to account for label imbalance.
    ``'samples'``:
        Calculate metrics for each instance, and find their average (only
        meaningful for multilabel classification).

sample_weight : array-like of shape (n_samples,), default=None
    Sample weights.

zero_division : "warn", {0.0, 1.0}, default="warn"
    Sets the value to return when there is a zero division, i.e. when there
    are no negative values in predictions and labels. If set to
    "warn", this acts like 0, but a warning is also raised.

```

Returns

score : float (if average is not None) or array of floats, shape =\n [n_unique_labels]

See Also

accuracy_score, f_score, multilabel_confusion_matrix

Notes

:func:`jaccard_score` may be a poor metric if there are no positives for some samples or classes. Jaccard is undefined if there are no true or predicted labels, and our implementation will return a score of 0 with a warning.

References

.. [1] `Wikipedia entry for the Jaccard index\n <https://en.wikipedia.org/wiki/Jaccard_index>`_.

Examples

```
>>> import numpy as np\n>>> from sklearn.metrics import jaccard_score\n>>> y_true = np.array([[0, 1, 1],\n...                    [1, 1, 0]])\n>>> y_pred = np.array([[1, 1, 1],\n...                    [1, 0, 0]])
```

In the binary case:

```
>>> jaccard_score(y_true[0], y_pred[0])\n0.6666...
```

In the multilabel case:

```
>>> jaccard_score(y_true, y_pred, average='samples')\n0.5833...\n>>> jaccard_score(y_true, y_pred, average='macro')\n0.6666...\n>>> jaccard_score(y_true, y_pred, average=None)\narray([0.5, 0.5, 1. ])
```

In the multiclass case:

```
>>> y_pred = [0, 2, 1, 2]\n>>> y_true = [0, 1, 2, 2]\n>>> jaccard_score(y_true, y_pred, average=None)\narray([1. , 0. , 0.33...])\n"""
```

```

761     labels = _check_set_wise_labels(y_true, y_pred, average, labels,
762                                     pos_label)
763     samplewise = average == 'samples'
764     MCM = multilabel_confusion_matrix(y_true, y_pred,
765                                       sample_weight=sample_weight,
766                                       labels=labels, samplewise=samplewise)
767     numerator = MCM[:, 1, 1]
768     denominator = MCM[:, 1, 1] + MCM[:, 0, 1] + MCM[:, 1, 0]
769
770     if average == 'micro':
771         numerator = np.array([numerator.sum()])
772         denominator = np.array([denominator.sum()])
773
774     jaccard = _prf_divide(numerator, denominator, 'jaccard',
775                          'true or predicted', average, ('jaccard',),
776                          zero_division=zero_division)
777     if average is None:
778         return jaccard
779     if average == 'weighted':
780         weights = MCM[:, 1, 0] + MCM[:, 1, 1]
781         if not np.any(weights):
782             # numerator is 0, and warning should have already been issued
783             weights = None
784     elif average == 'samples' and sample_weight is not None:
785         weights = sample_weight
786     else:
787         weights = None
788     return np.average(jaccard, weights=weights)
789
790
791 @_deprecate_positional_args
792 def matthews_corrcoef(y_true, y_pred, *, sample_weight=None):
793     """Compute the Matthews correlation coefficient (MCC).
794
795     The Matthews correlation coefficient is used in machine learning as a
796     measure of the quality of binary and multiclass classifications. It takes
797     into account true and false positives and negatives and is generally
798     regarded as a balanced measure which can be used even if the classes are of
799     very different sizes. The MCC is in essence a correlation coefficient value
800     between -1 and +1. A coefficient of +1 represents a perfect prediction, 0
801     an average random prediction and -1 an inverse prediction. The statistic
802     is also known as the phi coefficient. [source: Wikipedia]
803
804     Binary and multiclass labels are supported. Only in the binary case does
805     this relate to information about true and false positives and negatives.
806     See references below.
807
808     Read more in the :ref:`User Guide <matthews_corrcoef>`.
809
810     Parameters
811     -----
812     y_true : array, shape = [n_samples]

```



```

813         Ground truth (correct) target values.
814
815     y_pred : array, shape = [n_samples]
816         Estimated targets as returned by a classifier.
817
818     sample_weight : array-like of shape (n_samples,), default=None
819         Sample weights.
820
821     .. versionadded:: 0.18
822
823     Returns
824     -----
825     mcc : float
826         The Matthews correlation coefficient (+1 represents a perfect
827         prediction, 0 an average random prediction and -1 and inverse
828         prediction).
829
830     References
831     -----
832     .. [1] `Baldi, Brunak, Chauvin, Andersen and Nielsen, (2000). Assessing the
833         accuracy of prediction algorithms for classification: an overview
834         <https://doi.org/10.1093/bioinformatics/16.5.412>`_.
835
836     .. [2] `Wikipedia entry for the Matthews Correlation Coefficient
837         <https://en.wikipedia.org/wiki/Matthews\_correlation\_coefficient>`_.
838
839     .. [3] `Gorodkin, (2004). Comparing two K-category assignments by a
840         K-category correlation coefficient
841         <https://www.sciencedirect.com/science/article/pii/S1476927104000799>`_.
842
843     .. [4] `Jurman, Riccadonna, Furlanello, (2012). A Comparison of MCC and CEN
844         Error Measures in MultiClass Prediction
845         <https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0041882>`_.
846
847     Examples
848     -----
849     >>> from sklearn.metrics import matthews_corrcoef
850     >>> y_true = [+1, +1, +1, -1]
851     >>> y_pred = [+1, -1, +1, +1]
852     >>> matthews_corrcoef(y_true, y_pred)
853     -0.33...
854     """
855     y_type, y_true, y_pred = _check_targets(y_true, y_pred)
856     check_consistent_length(y_true, y_pred, sample_weight)
857     if y_type not in {"binary", "multiclass"}:
858         raise ValueError("%s is not supported" % y_type)
859
860     lb = LabelEncoder()
861     lb.fit(np.hstack([y_true, y_pred]))
862     y_true = lb.transform(y_true)
863     y_pred = lb.transform(y_pred)
864

```

```

865 C = confusion_matrix(y_true, y_pred, sample_weight=sample_weight)
866 t_sum = C.sum(axis=1, dtype=np.float64)
867 p_sum = C.sum(axis=0, dtype=np.float64)
868 n_correct = np.trace(C, dtype=np.float64)
869 n_samples = p_sum.sum()
870 cov_ytyp = n_correct * n_samples - np.dot(t_sum, p_sum)
871 cov_ypyp = n_samples ** 2 - np.dot(p_sum, p_sum)
872 cov_ytyt = n_samples ** 2 - np.dot(t_sum, t_sum)
873 mcc = cov_ytyp / np.sqrt(cov_ytyt * cov_ypyp)
874
875 if np.isnan(mcc):
876     return 0.
877 else:
878     return mcc
879
880
881 @_deprecated_positional_args
882 def zero_one_loss(y_true, y_pred, *, normalize=True, sample_weight=None):
883     """Zero-one classification loss.
884
885     If normalize is ``True``, return the fraction of misclassifications
886     (float), else it returns the number of misclassifications (int). The best
887     performance is 0.
888
889     Read more in the :ref:`User Guide <zero_one_loss>`.
890
891     Parameters
892     -----
893     y_true : 1d array-like, or label indicator array / sparse matrix
894         Ground truth (correct) labels.
895
896     y_pred : 1d array-like, or label indicator array / sparse matrix
897         Predicted labels, as returned by a classifier.
898
899     normalize : bool, default=True
900         If ``False``, return the number of misclassifications.
901         Otherwise, return the fraction of misclassifications.
902
903     sample_weight : array-like of shape (n_samples,), default=None
904         Sample weights.
905
906     Returns
907     -----
908     loss : float or int,
909         If ``normalize == True``, return the fraction of misclassifications
910         (float), else it returns the number of misclassifications (int).
911
912     Notes
913     -----
914     In multilabel classification, the zero_one_loss function corresponds to
915     the subset zero-one loss: for each sample, the entire set of labels must be
916     correctly predicted, otherwise the loss for that sample is equal to one.

```

See Also

`accuracy_score`, `hamming_loss`, `jaccard_score`

Examples

```
>>> from sklearn.metrics import zero_one_loss
```

```
>>> y_pred = [1, 2, 3, 4]
```

```
>>> y_true = [2, 2, 3, 4]
```

```
>>> zero_one_loss(y_true, y_pred)
```

```
0.25
```

```
>>> zero_one_loss(y_true, y_pred, normalize=False)
```

```
1
```

In the multilabel case with binary label indicators:

```
>>> import numpy as np
```

```
>>> zero_one_loss(np.array([[0, 1], [1, 1]]), np.ones((2, 2)))
```

```
0.5
```

```
"""
```

```
score = accuracy_score(y_true, y_pred,
```

```
                        normalize=normalize,
```

```
                        sample_weight=sample_weight)
```

```
if normalize:
```

```
    return 1 - score
```

```
else:
```

```
    if sample_weight is not None:
```

```
        n_samples = np.sum(sample_weight)
```

```
    else:
```

```
        n_samples = _num_samples(y_true)
```

```
    return n_samples - score
```

```
@_deprecated_positional_args
```

```
def f1_score(y_true, y_pred, *, labels=None, pos_label=1, average='binary',
```

```
            sample_weight=None, zero_division="warn"):
```

```
    """Compute the F1 score, also known as balanced F-score or F-measure.
```

The F1 score can be interpreted as a weighted average of the precision and recall, where an F1 score reaches its best value at 1 and worst score at 0. The relative contribution of precision and recall to the F1 score are equal. The formula for the F1 score is::

$$F1 = 2 * (precision * recall) / (precision + recall)$$

In the multi-class and multi-label case, this is the average of the F1 score of each class with weighting depending on the ``average`` parameter.

Read more in the :ref:`User Guide <precision_recall_f_measure_metrics>`.

Parameters

`y_true` : 1d array-like, or label indicator array / sparse matrix

Ground truth (correct) target values.

`y_pred` : 1d array-like, or label indicator array / sparse matrix

Estimated targets as returned by a classifier.

`labels` : array-like, default=None

The set of labels to include when `average != 'binary'`, and their order if `average is None`. Labels present in the data can be excluded, for example to calculate a multiclass average ignoring a majority negative class, while labels not present in the data will result in 0 components in a macro average. For multilabel targets, labels are column indices. By default, all labels in `y_true` and `y_pred` are used in sorted order.

.. versionchanged:: 0.17

Parameter `labels` improved for multiclass problem.

`pos_label` : str or int, default=1

The class to report if `average='binary'` and the data is binary.

If the data are multiclass or multilabel, this will be ignored;

setting `labels=[pos_label]` and `average != 'binary'` will report scores for that label only.

`average` : {'micro', 'macro', 'samples', 'weighted', 'binary'} or None, \
default='binary'

This parameter is required for multiclass/multilabel targets.

If `None`, the scores for each class are returned. Otherwise, this determines the type of averaging performed on the data:

`'binary'`:

Only report results for the class specified by `pos_label`.

This is applicable only if targets (`y_{true,pred}`) are binary.

`'micro'`:

Calculate metrics globally by counting the total true positives, false negatives and false positives.

`'macro'`:

Calculate metrics for each label, and find their unweighted mean. This does not take label imbalance into account.

`'weighted'`:

Calculate metrics for each label, and find their average weighted by support (the number of true instances for each label). This alters 'macro' to account for label imbalance; it can result in an F-score that is not between precision and recall.

`'samples'`:

Calculate metrics for each instance, and find their average (only meaningful for multilabel classification where this differs from :func:`accuracy_score`).

```

1021 sample_weight : array-like of shape (n_samples,), default=None
1022     Sample weights.
1023
1024 zero_division : "warn", 0 or 1, default="warn"
1025     Sets the value to return when there is a zero division, i.e. when all
1026     predictions and labels are negative. If set to "warn", this acts as 0,
1027     but warnings are also raised.
1028
1029 Returns
1030 -----
1031 f1_score : float or array of float, shape = [n_unique_labels]
1032     F1 score of the positive class in binary classification or weighted
1033     average of the F1 scores of each class for the multiclass task.
1034
1035 See Also
1036 -----
1037 fbeta_score, precision_recall_fscore_support, jaccard_score,
1038 multilabel_confusion_matrix
1039
1040 References
1041 -----
1042 .. [1] `Wikipedia entry for the F1-score
1043     <https://en.wikipedia.org/wiki/F1\_score>`_.
1044
1045 Examples
1046 -----
1047 >>> from sklearn.metrics import f1_score
1048 >>> y_true = [0, 1, 2, 0, 1, 2]
1049 >>> y_pred = [0, 2, 1, 0, 0, 1]
1050 >>> f1_score(y_true, y_pred, average='macro')
1051 0.26...
1052 >>> f1_score(y_true, y_pred, average='micro')
1053 0.33...
1054 >>> f1_score(y_true, y_pred, average='weighted')
1055 0.26...
1056 >>> f1_score(y_true, y_pred, average=None)
1057 array([0.8, 0. , 0. ])
1058 >>> y_true = [0, 0, 0, 0, 0, 0]
1059 >>> y_pred = [0, 0, 0, 0, 0, 0]
1060 >>> f1_score(y_true, y_pred, zero_division=1)
1061 1.0...
1062
1063 Notes
1064 -----
1065 When ``true positive + false positive == 0``, precision is undefined.
1066 When ``true positive + false negative == 0``, recall is undefined.
1067 In such cases, by default the metric will be set to 0, as will f-score,
1068 and ``UndefinedMetricWarning`` will be raised. This behavior can be
1069 modified with ``zero_division``.
1070 """"
1071
1072 return fbeta_score(y_true, y_pred, beta=1, labels=labels,
1073                    pos_label=pos_label, average=average,

```

```

1073         sample_weight=sample_weight,
1074         zero_division=zero_division)
1075
1076
1077 @_deprecate_positional_args
1078 def fbeta_score(y_true, y_pred, *, beta, labels=None, pos_label=1,
1079                average='binary', sample_weight=None, zero_division="warn"):
1080     """Compute the F-beta score.
1081
1082     The F-beta score is the weighted harmonic mean of precision and recall,
1083     reaching its optimal value at 1 and its worst value at 0.
1084
1085     The `beta` parameter determines the weight of recall in the combined
1086     score. ``beta < 1`` lends more weight to precision, while ``beta > 1``
1087     favors recall (``beta -> 0`` considers only precision, ``beta -> +inf``
1088     only recall).
1089
1090     Read more in the :ref:`User Guide <precision_recall_f_measure_metrics>`.
1091
1092     Parameters
1093     -----
1094     y_true : 1d array-like, or label indicator array / sparse matrix
1095         Ground truth (correct) target values.
1096
1097     y_pred : 1d array-like, or label indicator array / sparse matrix
1098         Estimated targets as returned by a classifier.
1099
1100     beta : float
1101         Determines the weight of recall in the combined score.
1102
1103     labels : array-like, default=None
1104         The set of labels to include when ``average != 'binary'``, and their
1105         order if ``average is None``. Labels present in the data can be
1106         excluded, for example to calculate a multiclass average ignoring a
1107         majority negative class, while labels not present in the data will
1108         result in 0 components in a macro average. For multilabel targets,
1109         labels are column indices. By default, all labels in ``y_true`` and
1110         ``y_pred`` are used in sorted order.
1111
1112     .. versionchanged:: 0.17
1113         Parameter `labels` improved for multiclass problem.
1114
1115     pos_label : str or int, default=1
1116         The class to report if ``average='binary'`` and the data is binary.
1117         If the data are multiclass or multilabel, this will be ignored;
1118         setting ``labels=[pos_label]`` and ``average != 'binary'`` will report
1119         scores for that label only.
1120
1121     average : {'micro', 'macro', 'samples', 'weighted', 'binary'} or None \
1122         default='binary'
1123
1124         This parameter is required for multiclass/multilabel targets.
1125         If ``None``, the scores for each class are returned. Otherwise, this

```

```

1125     determines the type of averaging performed on the data:
1126
1127     ``'binary'``:
1128         Only report results for the class specified by ``pos_label``.
1129         This is applicable only if targets (``y_{true,pred}``) are binary.
1130     ``'micro'``:
1131         Calculate metrics globally by counting the total true positives,
1132         false negatives and false positives.
1133     ``'macro'``:
1134         Calculate metrics for each label, and find their unweighted
1135         mean. This does not take label imbalance into account.
1136     ``'weighted'``:
1137         Calculate metrics for each label, and find their average weighted
1138         by support (the number of true instances for each label). This
1139         alters 'macro' to account for label imbalance; it can result in an
1140         F-score that is not between precision and recall.
1141     ``'samples'``:
1142         Calculate metrics for each instance, and find their average (only
1143         meaningful for multilabel classification where this differs from
1144         :func:`accuracy_score`).
1145
1146 sample_weight : array-like of shape (n_samples,), default=None
1147     Sample weights.
1148
1149 zero_division : "warn", 0 or 1, default="warn"
1150     Sets the value to return when there is a zero division, i.e. when all
1151     predictions and labels are negative. If set to "warn", this acts as 0,
1152     but warnings are also raised.
1153
1154 Returns
1155 -----
1156 fbeta_score : float (if average is not None) or array of float, shape =\
1157     [n_unique_labels]
1158     F-beta score of the positive class in binary classification or weighted
1159     average of the F-beta score of each class for the multiclass task.
1160
1161 See Also
1162 -----
1163 precision_recall_fscore_support, multilabel_confusion_matrix
1164
1165 Notes
1166 -----
1167 When ``true positive + false positive == 0`` or
1168 ``true positive + false negative == 0``, f-score returns 0 and raises
1169 ``UndefinedMetricWarning``. This behavior can be
1170 modified with ``zero_division``.
1171
1172 References
1173 -----
1174 .. [1] R. Baeza-Yates and B. Ribeiro-Neto (2011).
1175     Modern Information Retrieval. Addison Wesley, pp. 327-328.
1176

```

```

1177 .. [2] `Wikipedia entry for the F1-score
1178      <https://en.wikipedia.org/wiki/F1\_score>`.
1179
1180 Examples
1181 -----
1182 >>> from sklearn.metrics import fbeta_score
1183 >>> y_true = [0, 1, 2, 0, 1, 2]
1184 >>> y_pred = [0, 2, 1, 0, 0, 1]
1185 >>> fbeta_score(y_true, y_pred, average='macro', beta=0.5)
1186 0.23...
1187 >>> fbeta_score(y_true, y_pred, average='micro', beta=0.5)
1188 0.33...
1189 >>> fbeta_score(y_true, y_pred, average='weighted', beta=0.5)
1190 0.23...
1191 >>> fbeta_score(y_true, y_pred, average=None, beta=0.5)
1192 array([0.71..., 0.          , 0.          ])
1193 ""
1194
1195 _, _, f, _ = precision_recall_fscore_support(y_true, y_pred,
1196                                             beta=beta,
1197                                             labels=labels,
1198                                             pos_label=pos_label,
1199                                             average=average,
1200                                             warn_for=('f-score',),
1201                                             sample_weight=sample_weight,
1202                                             zero_division=zero_division)
1203
1204 return f
1205
1206 def _prf_divide(numerator, denominator, metric,
1207                modifier, average, warn_for, zero_division="warn"):
1208     """Performs division and handles divide-by-zero.
1209
1210     On zero-division, sets the corresponding result elements equal to
1211     0 or 1 (according to ``zero_division``). Plus, if
1212     ``zero_division != "warn"`` raises a warning.
1213
1214     The metric, modifier and average arguments are used only for determining
1215     an appropriate warning.
1216     """
1217     mask = denominator == 0.0
1218     denominator = denominator.copy()
1219     denominator[mask] = 1 # avoid infs/nans
1220     result = numerator / denominator
1221
1222     if not np.any(mask):
1223         return result
1224
1225     # if ``zero_division=1``, set those with denominator == 0 equal to 1
1226     result[mask] = 0.0 if zero_division in ["warn", 0] else 1.0
1227
1228     # the user will be removing warnings if zero_division is set to something

```



```

1229 # different than its default value. If we are computing only f-score
1230 # the warning will be raised only if precision and recall are ill-defined
1231 if zero_division != "warn" or metric not in warn_for:
1232     return result
1233
1234 # build appropriate warning
1235 # E.g. "Precision and F-score are ill-defined and being set to 0.0 in
1236 # labels with no predicted samples. Use ``zero_division`` parameter to
1237 # control this behavior."
1238
1239 if metric in warn_for and 'f-score' in warn_for:
1240     msg_start = '{0} and F-score are'.format(metric.title())
1241 elif metric in warn_for:
1242     msg_start = '{0} is'.format(metric.title())
1243 elif 'f-score' in warn_for:
1244     msg_start = 'F-score is'
1245 else:
1246     return result
1247
1248 _warn_prf(average, modifier, msg_start, len(result))
1249
1250 return result
1251
1252
1253 def _warn_prf(average, modifier, msg_start, result_size):
1254     axis0, axis1 = 'sample', 'label'
1255     if average == 'samples':
1256         axis0, axis1 = axis1, axis0
1257     msg = ('{0} ill-defined and being set to 0.0 {{0}} '
1258           'no {1} {2}s. Use `zero_division` parameter to control'
1259           ' this behavior.'.format(msg_start, modifier, axis0))
1260     if result_size == 1:
1261         msg = msg.format('due to')
1262     else:
1263         msg = msg.format('in {0}s with'.format(axis1))
1264     warnings.warn(msg, UndefinedMetricWarning, stacklevel=2)
1265
1266
1267 def _check_set_wise_labels(y_true, y_pred, average, labels, pos_label):
1268     """Validation associated with set-wise metrics.
1269
1270     Returns identified labels.
1271     """
1272     average_options = (None, 'micro', 'macro', 'weighted', 'samples')
1273     if average not in average_options and average != 'binary':
1274         raise ValueError('average has to be one of ' +
1275                           str(average_options))
1276
1277     y_type, y_true, y_pred = _check_targets(y_true, y_pred)
1278     # Convert to Python primitive type to avoid NumPy type / Python str
1279     # comparison. See https://github.com/numpy/numpy/issues/6784
1280     present_labels = unique_labels(y_true, y_pred).tolist()

```

```

1281     if average == 'binary':
1282         if y_type == 'binary':
1283             if pos_label not in present_labels:
1284                 if len(present_labels) >= 2:
1285                     raise ValueError(
1286                         f"pos_label={pos_label} is not a valid label. It "
1287                         f"should be one of {present_labels}"
1288                     )
1289                 labels = [pos_label]
1290         else:
1291             average_options = list(average_options)
1292             if y_type == 'multiclass':
1293                 average_options.remove('samples')
1294             raise ValueError("Target is %s but average='binary'. Please "
1295                               "choose another average setting, one of %r."
1296                               % (y_type, average_options))
1297     elif pos_label not in (None, 1):
1298         warnings.warn("Note that pos_label (set to %r) is ignored when "
1299                       "average != 'binary' (got %r). You may use "
1300                       "labels=[pos_label] to specify a single positive class."
1301                       % (pos_label, average), UserWarning)
1302     return labels
1303
1304
1305 @_deprecate_positional_args
1306 def precision_recall_fscore_support(y_true, y_pred, *, beta=1.0, labels=None,
1307                                    pos_label=1, average=None,
1308                                    warn_for=('precision', 'recall',
1309                                              'f-score'),
1310                                    sample_weight=None,
1311                                    zero_division="warn"):
1312     """Compute precision, recall, F-measure and support for each class.
1313
1314     The precision is the ratio ``tp / (tp + fp)`` where ``tp`` is the number of
1315     true positives and ``fp`` the number of false positives. The precision is
1316     intuitively the ability of the classifier not to label as positive a sample
1317     that is negative.
1318
1319     The recall is the ratio ``tp / (tp + fn)`` where ``tp`` is the number of
1320     true positives and ``fn`` the number of false negatives. The recall is
1321     intuitively the ability of the classifier to find all the positive samples.
1322
1323     The F-beta score can be interpreted as a weighted harmonic mean of
1324     the precision and recall, where an F-beta score reaches its best
1325     value at 1 and worst score at 0.
1326
1327     The F-beta score weights recall more than precision by a factor of
1328     ``beta``. ``beta == 1.0`` means recall and precision are equally important.
1329
1330     The support is the number of occurrences of each class in ``y_true``.
1331
1332     If ``pos_label is None`` and in binary classification, this function

```

returns the average precision, recall and F-measure if ``average`` is one of ``'micro'``, ``'macro'``, ``'weighted'`` or ``'samples'``.

Read more in the :ref:`User Guide <precision_recall_f_measure_metrics>`.

Parameters

y_true : 1d array-like, or label indicator array / sparse matrix
Ground truth (correct) target values.

y_pred : 1d array-like, or label indicator array / sparse matrix
Estimated targets as returned by a classifier.

beta : float, default=1.0
The strength of recall versus precision in the F-score.

labels : array-like, default=None
The set of labels to include when ``average != 'binary'``, and their order if ``average is None``. Labels present in the data can be excluded, for example to calculate a multiclass average ignoring a majority negative class, while labels not present in the data will result in 0 components in a macro average. For multilabel targets, labels are column indices. By default, all labels in ``y_true`` and ``y_pred`` are used in sorted order.

pos_label : str or int, default=1
The class to report if ``average='binary'`` and the data is binary. If the data are multiclass or multilabel, this will be ignored; setting ``labels=[pos_label]`` and ``average != 'binary'`` will report scores for that label only.

average : {'binary', 'micro', 'macro', 'samples', 'weighted'}, \ default=None
If ``None``, the scores for each class are returned. Otherwise, this determines the type of averaging performed on the data:

``'binary'``:
Only report results for the class specified by ``pos_label``. This is applicable only if targets (``y_{true,pred}``) are binary.

``'micro'``:
Calculate metrics globally by counting the total true positives, false negatives and false positives.

``'macro'``:
Calculate metrics for each label, and find their unweighted mean. This does not take label imbalance into account.

``'weighted'``:
Calculate metrics for each label, and find their average weighted by support (the number of true instances for each label). This alters 'macro' to account for label imbalance; it can result in an F-score that is not between precision and recall.

``'samples'``:
Calculate metrics for each instance, and find their average (only

```

1385         meaningful for multilabel classification where this differs from
1386         :func:`accuracy_score`).
1387
1388     warn_for : tuple or set, for internal use
1389         This determines which warnings will be made in the case that this
1390         function is being used to return only one of its metrics.
1391
1392     sample_weight : array-like of shape (n_samples,), default=None
1393         Sample weights.
1394
1395     zero_division : "warn", 0 or 1, default="warn"
1396         Sets the value to return when there is a zero division:
1397         - recall: when there are no positive labels
1398         - precision: when there are no positive predictions
1399         - f-score: both
1400
1401         If set to "warn", this acts as 0, but warnings are also raised.
1402
1403     Returns
1404     -----
1405     precision : float (if average is not None) or array of float, shape =\
1406         [n_unique_labels]
1407
1408     recall : float (if average is not None) or array of float, , shape =\
1409         [n_unique_labels]
1410
1411     fbeta_score : float (if average is not None) or array of float, shape =\
1412         [n_unique_labels]
1413
1414     support : None (if average is not None) or array of int, shape =\
1415         [n_unique_labels]
1416         The number of occurrences of each label in ``y_true``.
1417
1418     Notes
1419     -----
1420     When ``true positive + false positive == 0``, precision is undefined.
1421     When ``true positive + false negative == 0``, recall is undefined.
1422     In such cases, by default the metric will be set to 0, as will f-score,
1423     and ``UndefinedMetricWarning`` will be raised. This behavior can be
1424     modified with ``zero_division``.
1425
1426     References
1427     -----
1428     .. [1] `Wikipedia entry for the Precision and recall
1429         <https://en.wikipedia.org/wiki/Precision\_and\_recall>`_.
1430
1431     .. [2] `Wikipedia entry for the F1-score
1432         <https://en.wikipedia.org/wiki/F1\_score>`_.
1433
1434     .. [3] `Discriminative Methods for Multi-labeled Classification Advances
1435         in Knowledge Discovery and Data Mining (2004), pp. 22-30 by Shantanu
1436         Godbole, Sunita Sarawagi

```

<<http://www.godbole.net/shantanu/pubs/multilabelsvm-pakdd04.pdf>>`_.

Examples

```
>>> import numpy as np
>>> from sklearn.metrics import precision_recall_fscore_support
>>> y_true = np.array(['cat', 'dog', 'pig', 'cat', 'dog', 'pig'])
>>> y_pred = np.array(['cat', 'pig', 'dog', 'cat', 'cat', 'dog'])
>>> precision_recall_fscore_support(y_true, y_pred, average='macro')
(0.22..., 0.33..., 0.26..., None)
>>> precision_recall_fscore_support(y_true, y_pred, average='micro')
(0.33..., 0.33..., 0.33..., None)
>>> precision_recall_fscore_support(y_true, y_pred, average='weighted')
(0.22..., 0.33..., 0.26..., None)
```

It is possible to compute per-label precisions, recalls, F1-scores and supports instead of averaging:

```
>>> precision_recall_fscore_support(y_true, y_pred, average=None,
... labels=['pig', 'dog', 'cat'])
(array([0.          , 0.          , 0.66...]),
 array([0., 0., 1.]), array([0. , 0. , 0.8]),
 array([2, 2, 2]))
"""
_check_zero_division(zero_division)
if beta < 0:
    raise ValueError("beta should be >=0 in the F-beta score")
labels = _check_set_wise_labels(y_true, y_pred, average, labels,
                                pos_label)

# Calculate tp_sum, pred_sum, true_sum ###
samplewise = average == 'samples'
MCM = multilabel_confusion_matrix(y_true, y_pred,
                                sample_weight=sample_weight,
                                labels=labels, samplewise=samplewise)

tp_sum = MCM[:, 1, 1]
pred_sum = tp_sum + MCM[:, 0, 1]
true_sum = tp_sum + MCM[:, 1, 0]

if average == 'micro':
    tp_sum = np.array([tp_sum.sum()])
    pred_sum = np.array([pred_sum.sum()])
    true_sum = np.array([true_sum.sum()])

# Finally, we have all our sufficient statistics. Divide! #
beta2 = beta ** 2

# Divide, and on zero-division, set scores and/or warn according to
# zero_division:
precision = _prf_divide(tp_sum, pred_sum, 'precision',
                        'predicted', average, warn_for, zero_division)
recall = _prf_divide(tp_sum, true_sum, 'recall',
```

```

1489         'true', average, warn_for, zero_division)
1490
1491     # warn for f-score only if zero_division is warn, it is in warn_for
1492     # and BOTH prec and rec are ill-defined
1493     if zero_division == "warn" and ("f-score",) == warn_for:
1494         if (pred_sum[true_sum == 0] == 0).any():
1495             _warn_prf(
1496                 average, "true nor predicted", 'F-score is', len(true_sum)
1497             )
1498
1499     # if tp == 0 F will be 1 only if all predictions are zero, all labels are
1500     # zero, and zero_division=1. In all other case, 0
1501     if np.isposinf(beta):
1502         f_score = recall
1503     else:
1504         denom = beta2 * precision + recall
1505
1506         denom[denom == 0.] = 1 # avoid division by 0
1507         f_score = (1 + beta2) * precision * recall / denom
1508
1509     # Average the results
1510     if average == 'weighted':
1511         weights = true_sum
1512         if weights.sum() == 0:
1513             zero_division_value = np.float64(1.0)
1514             if zero_division in ["warn", 0]:
1515                 zero_division_value = np.float64(0.0)
1516             # precision is zero_division if there are no positive predictions
1517             # recall is zero_division if there are no positive labels
1518             # fscore is zero_division if all labels AND predictions are
1519             # negative
1520             if pred_sum.sum() == 0:
1521                 return (zero_division_value,
1522                         zero_division_value,
1523                         zero_division_value,
1524                         None)
1525             else:
1526                 return (np.float64(0.0),
1527                         zero_division_value,
1528                         np.float64(0.0),
1529                         None)
1530
1531     elif average == 'samples':
1532         weights = sample_weight
1533     else:
1534         weights = None
1535
1536     if average is not None:
1537         assert average != 'binary' or len(precision) == 1
1538         precision = np.average(precision, weights=weights)
1539         recall = np.average(recall, weights=weights)
1540         f_score = np.average(f_score, weights=weights)

```

```

1541         true_sum = None # return no support
1542
1543     return precision, recall, f_score, true_sum
1544
1545
1546 @_deprecated_positional_args
1547 def precision_score(y_true, y_pred, *, labels=None, pos_label=1,
1548                    average='binary', sample_weight=None,
1549                    zero_division="warn"):
1550     """Compute the precision.
1551
1552     The precision is the ratio ``tp / (tp + fp)`` where ``tp`` is the number of
1553     true positives and ``fp`` the number of false positives. The precision is
1554     intuitively the ability of the classifier not to label as positive a sample
1555     that is negative.
1556
1557     The best value is 1 and the worst value is 0.
1558
1559     Read more in the :ref:`User Guide <precision_recall_f_measure_metrics>`.
1560
1561     Parameters
1562     -----
1563     y_true : 1d array-like, or label indicator array / sparse matrix
1564         Ground truth (correct) target values.
1565
1566     y_pred : 1d array-like, or label indicator array / sparse matrix
1567         Estimated targets as returned by a classifier.
1568
1569     labels : array-like, default=None
1570         The set of labels to include when ``average != 'binary'``, and their
1571         order if ``average is None``. Labels present in the data can be
1572         excluded, for example to calculate a multiclass average ignoring a
1573         majority negative class, while labels not present in the data will
1574         result in 0 components in a macro average. For multilabel targets,
1575         labels are column indices. By default, all labels in ``y_true`` and
1576         ``y_pred`` are used in sorted order.
1577
1578     .. versionchanged:: 0.17
1579         Parameter `labels` improved for multiclass problem.
1580
1581     pos_label : str or int, default=1
1582         The class to report if ``average='binary'`` and the data is binary.
1583         If the data are multiclass or multilabel, this will be ignored;
1584         setting ``labels=[pos_label]`` and ``average != 'binary'`` will report
1585         scores for that label only.
1586
1587     average : {'micro', 'macro', 'samples', 'weighted', 'binary'} \
1588         default='binary'
1589         This parameter is required for multiclass/multilabel targets.
1590         If ``None``, the scores for each class are returned. Otherwise, this
1591         determines the type of averaging performed on the data:
1592

```

```

1593     ``'binary'``:
1594         Only report results for the class specified by ``pos_label``.
1595         This is applicable only if targets (``y_{true,pred}``) are binary.
1596     ``'micro'``:
1597         Calculate metrics globally by counting the total true positives,
1598         false negatives and false positives.
1599     ``'macro'``:
1600         Calculate metrics for each label, and find their unweighted
1601         mean. This does not take label imbalance into account.
1602     ``'weighted'``:
1603         Calculate metrics for each label, and find their average weighted
1604         by support (the number of true instances for each label). This
1605         alters 'macro' to account for label imbalance; it can result in an
1606         F-score that is not between precision and recall.
1607     ``'samples'``:
1608         Calculate metrics for each instance, and find their average (only
1609         meaningful for multilabel classification where this differs from
1610         :func:`accuracy_score`).
1611
1612 sample_weight : array-like of shape (n_samples,), default=None
1613     Sample weights.
1614
1615 zero_division : "warn", 0 or 1, default="warn"
1616     Sets the value to return when there is a zero division. If set to
1617     "warn", this acts as 0, but warnings are also raised.
1618
1619 Returns
1620 -----
1621 precision : float (if average is not None) or array of float of shape
1622     (n_unique_labels,)
1623     Precision of the positive class in binary classification or weighted
1624     average of the precision of each class for the multiclass task.
1625
1626 See Also
1627 -----
1628 precision_recall_fscore_support, multilabel_confusion_matrix
1629
1630 Notes
1631 -----
1632 When ``true positive + false positive == 0``, precision returns 0 and
1633 raises ``UndefinedMetricWarning``. This behavior can be
1634 modified with ``zero_division``.
1635
1636 Examples
1637 -----
1638 >>> from sklearn.metrics import precision_score
1639 >>> y_true = [0, 1, 2, 0, 1, 2]
1640 >>> y_pred = [0, 2, 1, 0, 0, 1]
1641 >>> precision_score(y_true, y_pred, average='macro')
1642 0.22...
1643 >>> precision_score(y_true, y_pred, average='micro')
1644 0.33...

```



```

1645 >>> precision_score(y_true, y_pred, average='weighted')
1646 0.22...
1647 >>> precision_score(y_true, y_pred, average=None)
1648 array([0.66..., 0.        , 0.        ])
1649 >>> y_pred = [0, 0, 0, 0, 0, 0]
1650 >>> precision_score(y_true, y_pred, average=None)
1651 array([0.33..., 0.        , 0.        ])
1652 >>> precision_score(y_true, y_pred, average=None, zero_division=1)
1653 array([0.33..., 1.        , 1.        ])
1654
1655 """
1656 p, _, _, _ = precision_recall_fscore_support(y_true, y_pred,
1657                                             labels=labels,
1658                                             pos_label=pos_label,
1659                                             average=average,
1660                                             warn_for=('precision',),
1661                                             sample_weight=sample_weight,
1662                                             zero_division=zero_division)
1663
1664 return p
1665
1666 @_deprecated_positional_args
1667 def recall_score(y_true, y_pred, *, labels=None, pos_label=1, average='binary',
1668                 sample_weight=None, zero_division="warn"):
1669     """Compute the recall.
1670
1671     The recall is the ratio ``tp / (tp + fn)`` where ``tp`` is the number of
1672     true positives and ``fn`` the number of false negatives. The recall is
1673     intuitively the ability of the classifier to find all the positive samples.
1674
1675     The best value is 1 and the worst value is 0.
1676
1677     Read more in the :ref:`User Guide <precision_recall_f_measure_metrics>`.
1678
1679     Parameters
1680     -----
1681     y_true : 1d array-like, or label indicator array / sparse matrix
1682         Ground truth (correct) target values.
1683
1684     y_pred : 1d array-like, or label indicator array / sparse matrix
1685         Estimated targets as returned by a classifier.
1686
1687     labels : array-like, default=None
1688         The set of labels to include when ``average != 'binary'``, and their
1689         order if ``average is None``. Labels present in the data can be
1690         excluded, for example to calculate a multiclass average ignoring a
1691         majority negative class, while labels not present in the data will
1692         result in 0 components in a macro average. For multilabel targets,
1693         labels are column indices. By default, all labels in ``y_true`` and
1694         ``y_pred`` are used in sorted order.
1695
1696     .. versionchanged:: 0.17

```

```

1697         Parameter `labels` improved for multiclass problem.
1698
1699     pos_label : str or int, default=1
1700         The class to report if ``average='binary'`` and the data is binary.
1701         If the data are multiclass or multilabel, this will be ignored;
1702         setting ``labels=[pos_label]`` and ``average != 'binary'`` will report
1703         scores for that label only.
1704
1705     average : {'micro', 'macro', 'samples', 'weighted', 'binary'} \
1706             default='binary'
1707         This parameter is required for multiclass/multilabel targets.
1708         If ``None``, the scores for each class are returned. Otherwise, this
1709         determines the type of averaging performed on the data:
1710
1711         ``'binary'``:
1712             Only report results for the class specified by ``pos_label``.
1713             This is applicable only if targets (``y_{true,pred}``) are binary.
1714         ``'micro'``:
1715             Calculate metrics globally by counting the total true positives,
1716             false negatives and false positives.
1717         ``'macro'``:
1718             Calculate metrics for each label, and find their unweighted
1719             mean. This does not take label imbalance into account.
1720         ``'weighted'``:
1721             Calculate metrics for each label, and find their average weighted
1722             by support (the number of true instances for each label). This
1723             alters 'macro' to account for label imbalance; it can result in an
1724             F-score that is not between precision and recall.
1725         ``'samples'``:
1726             Calculate metrics for each instance, and find their average (only
1727             meaningful for multilabel classification where this differs from
1728             :func:`accuracy_score`).
1729
1730     sample_weight : array-like of shape (n_samples,), default=None
1731         Sample weights.
1732
1733     zero_division : "warn", 0 or 1, default="warn"
1734         Sets the value to return when there is a zero division. If set to
1735         "warn", this acts as 0, but warnings are also raised.
1736
1737     Returns
1738     -----
1739     recall : float (if average is not None) or array of float of shape
1740             (n_unique_labels,)
1741         Recall of the positive class in binary classification or weighted
1742         average of the recall of each class for the multiclass task.
1743
1744     See Also
1745     -----
1746     precision_recall_fscore_support, balanced_accuracy_score,
1747     multilabel_confusion_matrix
1748

```

Notes

When ``true positive + false negative == 0``, recall returns 0 and raises ``UndefinedMetricWarning``. This behavior can be modified with ``zero_division``.

Examples

```
>>> from sklearn.metrics import recall_score
>>> y_true = [0, 1, 2, 0, 1, 2]
>>> y_pred = [0, 2, 1, 0, 0, 1]
>>> recall_score(y_true, y_pred, average='macro')
0.33...
>>> recall_score(y_true, y_pred, average='micro')
0.33...
>>> recall_score(y_true, y_pred, average='weighted')
0.33...
>>> recall_score(y_true, y_pred, average=None)
array([1., 0., 0.])
>>> y_true = [0, 0, 0, 0, 0, 0]
>>> recall_score(y_true, y_pred, average=None)
array([0.5, 0. , 0. ])
>>> recall_score(y_true, y_pred, average=None, zero_division=1)
array([0.5, 1. , 1. ])
"""
_, r, _, _ = precision_recall_fscore_support(y_true, y_pred,
                                             labels=labels,
                                             pos_label=pos_label,
                                             average=average,
                                             warn_for=('recall',),
                                             sample_weight=sample_weight,
                                             zero_division=zero_division)
```

```
return r
```

@_deprecate_positional_args

```
def balanced_accuracy_score(y_true, y_pred, *, sample_weight=None,
                             adjusted=False):
```

"""Compute the balanced accuracy.

The balanced accuracy in binary and multiclass classification problems to deal with imbalanced datasets. It is defined as the average of recall obtained on each class.

The best value is 1 and the worst value is 0 when ``adjusted=False``.

Read more in the :ref:`User Guide <balanced_accuracy_score>`.

.. versionadded:: 0.20

Parameters

```

1801     y_true : 1d array-like
1802         Ground truth (correct) target values.
1803
1804     y_pred : 1d array-like
1805         Estimated targets as returned by a classifier.
1806
1807     sample_weight : array-like of shape (n_samples,), default=None
1808         Sample weights.
1809
1810     adjusted : bool, default=False
1811         When true, the result is adjusted for chance, so that random
1812         performance would score 0, while keeping perfect performance at a score
1813         of 1.
1814
1815     Returns
1816     -----
1817     balanced_accuracy : float
1818
1819     See Also
1820     -----
1821     recall_score, roc_auc_score
1822
1823     Notes
1824     -----
1825     Some literature promotes alternative definitions of balanced accuracy. Our
1826     definition is equivalent to :func:`accuracy_score` with class-balanced
1827     sample weights, and shares desirable properties with the binary case.
1828     See the :ref:`User Guide <balanced_accuracy_score>`.
1829
1830     References
1831     -----
1832     .. [1] Brodersen, K.H.; Ong, C.S.; Stephan, K.E.; Buhmann, J.M. (2010).
1833         The balanced accuracy and its posterior distribution.
1834         Proceedings of the 20th International Conference on Pattern
1835         Recognition, 3121-24.
1836     .. [2] John. D. Kelleher, Brian Mac Namee, Aoife D'Arcy, (2015).
1837         `Fundamentals of Machine Learning for Predictive Data Analytics:
1838         Algorithms, Worked Examples, and Case Studies
1839         <https://mitpress.mit.edu/books/fundamentals-machine-learning-predictive-data-analyt
1840
1841     Examples
1842     -----
1843     >>> from sklearn.metrics import balanced_accuracy_score
1844     >>> y_true = [0, 1, 0, 0, 1, 0]
1845     >>> y_pred = [0, 1, 0, 0, 0, 1]
1846     >>> balanced_accuracy_score(y_true, y_pred)
1847     0.625
1848
1849     """
1850     C = confusion_matrix(y_true, y_pred, sample_weight=sample_weight)
1851     with np.errstate(divide='ignore', invalid='ignore'):
1852         per_class = np.diag(C) / C.sum(axis=1)

```

```

1853     if np.any(np.isnan(per_class)):
1854         warnings.warn('y_pred contains classes not in y_true')
1855         per_class = per_class[~np.isnan(per_class)]
1856     score = np.mean(per_class)
1857     if adjusted:
1858         n_classes = len(per_class)
1859         chance = 1 / n_classes
1860         score -= chance
1861         score /= 1 - chance
1862     return score
1863
1864
1865 @_deprecate_positional_args
1866 def classification_report(y_true, y_pred, *, labels=None, target_names=None,
1867                          sample_weight=None, digits=2, output_dict=False,
1868                          zero_division="warn"):
1869     """Build a text report showing the main classification metrics.
1870
1871     Read more in the :ref:`User Guide <classification_report>`.
1872
1873     Parameters
1874     -----
1875     y_true : 1d array-like, or label indicator array / sparse matrix
1876         Ground truth (correct) target values.
1877
1878     y_pred : 1d array-like, or label indicator array / sparse matrix
1879         Estimated targets as returned by a classifier.
1880
1881     labels : array-like of shape (n_labels,), default=None
1882         Optional list of label indices to include in the report.
1883
1884     target_names : list of str of shape (n_labels,), default=None
1885         Optional display names matching the labels (same order).
1886
1887     sample_weight : array-like of shape (n_samples,), default=None
1888         Sample weights.
1889
1890     digits : int, default=2
1891         Number of digits for formatting output floating point values.
1892         When ``output_dict`` is ``True``, this will be ignored and the
1893         returned values will not be rounded.
1894
1895     output_dict : bool, default=False
1896         If True, return output as dict.
1897
1898         .. versionadded:: 0.20
1899
1900     zero_division : "warn", 0 or 1, default="warn"
1901         Sets the value to return when there is a zero division. If set to
1902         "warn", this acts as 0, but warnings are also raised.
1903
1904     Returns

```

```
-----
report : string / dict
    Text summary of the precision, recall, F1 score for each class.
    Dictionary returned if output_dict is True. Dictionary has the
    following structure::
```

```
        {'label 1': {'precision':0.5,
                      'recall':1.0,
                      'f1-score':0.67,
                      'support':1},
         'label 2': { ... },
         ...
        }
```

The reported averages include macro average (averaging the unweighted mean per label), weighted average (averaging the support-weighted mean per label), and sample average (only for multilabel classification). Micro average (averaging the total true positives, false negatives and false positives) is only shown for multi-label or multi-class with a subset of classes, because it corresponds to accuracy otherwise and would be the same for all metrics. See also :func:`precision_recall_fscore_support` for more details on averages.

Note that in binary classification, recall of the positive class is also known as "sensitivity"; recall of the negative class is "specificity".

See Also

```
-----
precision_recall_fscore_support, confusion_matrix,
multilabel_confusion_matrix
```

Examples

```
-----
>>> from sklearn.metrics import classification_report
>>> y_true = [0, 1, 2, 2, 2]
>>> y_pred = [0, 0, 2, 2, 1]
>>> target_names = ['class 0', 'class 1', 'class 2']
>>> print(classification_report(y_true, y_pred, target_names=target_names))
           precision    recall  f1-score   support

<BLANKLINE>
  class 0       0.50      1.00      0.67         1
  class 1       0.00      0.00      0.00         1
  class 2       1.00      0.67      0.80         3

<BLANKLINE>
 accuracy              0.60         5
  macro avg           0.50      0.56      0.49         5
weighted avg           0.70      0.60      0.61         5

<BLANKLINE>
>>> y_pred = [1, 1, 0]
>>> y_true = [1, 1, 1]
```

```

1957 >>> print(classification_report(y_true, y_pred, labels=[1, 2, 3]))
1958             precision    recall  f1-score   support
1959
1960      1             1.00      0.67      0.80         3
1961      2             0.00      0.00      0.00         0
1962      3             0.00      0.00      0.00         0
1963
1964   micro avg       1.00      0.67      0.80         3
1965   macro avg       0.33      0.22      0.27         3
1966 weighted avg       1.00      0.67      0.80         3
1967
1968 <BLANKLINE>
1969
1970 y_type, y_true, y_pred = _check_targets(y_true, y_pred)
1971
1972 if labels is None:
1973     labels = unique_labels(y_true, y_pred)
1974     labels_given = False
1975 else:
1976     labels = np.asarray(labels)
1977     labels_given = True
1978
1979 # labelled micro average
1980 micro_is_accuracy = ((y_type == 'multiclass' or y_type == 'binary') and
1981                      (not labels_given or
1982                       (set(labels) == set(unique_labels(y_true, y_pred)))))
1983
1984 if target_names is not None and len(labels) != len(target_names):
1985     if labels_given:
1986         warnings.warn(
1987             "labels size, {0}, does not match size of target_names, {1}"
1988             .format(len(labels), len(target_names))
1989         )
1990     else:
1991         raise ValueError(
1992             "Number of classes, {0}, does not match size of "
1993             "target_names, {1}. Try specifying the labels "
1994             "parameter".format(len(labels), len(target_names))
1995         )
1996 if target_names is None:
1997     target_names = ['%s' % l for l in labels]
1998
1999 headers = ["precision", "recall", "f1-score", "support"]
2000 # compute per-class results without averaging
2001 p, r, f1, s = precision_recall_fscore_support(y_true, y_pred,
2002                                               labels=labels,
2003                                               average=None,
2004                                               sample_weight=sample_weight,
2005                                               zero_division=zero_division)
2006 rows = zip(target_names, p, r, f1, s)
2007
2008 if y_type.startswith('multilabel'):

```

```

2009         average_options = ('micro', 'macro', 'weighted', 'samples')
2010     else:
2011         average_options = ('micro', 'macro', 'weighted')
2012
2013     if output_dict:
2014         report_dict = {label[0]: label[1:] for label in rows}
2015         for label, scores in report_dict.items():
2016             report_dict[label] = dict(zip(headers,
2017                                         [i.item() for i in scores]))
2018     else:
2019         longest_last_line_heading = 'weighted avg'
2020         name_width = max(len(cn) for cn in target_names)
2021         width = max(name_width, len(longest_last_line_heading), digits)
2022         head_fmt = '{:>{width}s} ' + ' {:>9}' * len(headers)
2023         report = head_fmt.format('', *headers, width=width)
2024         report += '\n\n'
2025         row_fmt = '{:>{width}s} ' + ' {:>9.{digits}f}' * 3 + ' {:>9}\n'
2026         for row in rows:
2027             report += row_fmt.format(*row, width=width, digits=digits)
2028         report += '\n'
2029
2030     # compute all applicable averages
2031     for average in average_options:
2032         if average.startswith('micro') and micro_is_accuracy:
2033             line_heading = 'accuracy'
2034         else:
2035             line_heading = average + ' avg'
2036
2037     # compute averages with specified averaging method
2038     avg_p, avg_r, avg_f1, _ = precision_recall_fscore_support(
2039         y_true, y_pred, labels=labels,
2040         average=average, sample_weight=sample_weight,
2041         zero_division=zero_division)
2042     avg = [avg_p, avg_r, avg_f1, np.sum(s)]
2043
2044     if output_dict:
2045         report_dict[line_heading] = dict(
2046             zip(headers, [i.item() for i in avg]))
2047     else:
2048         if line_heading == 'accuracy':
2049             row_fmt_accuracy = '{:>{width}s} ' + \
2050                 ' {:>9.{digits}}' * 2 + ' {:>9.{digits}f}' + \
2051                 ' {:>9}\n'
2052             report += row_fmt_accuracy.format(line_heading, '', '',
2053                                             *avg[2:], width=width,
2054                                             digits=digits)
2055         else:
2056             report += row_fmt.format(line_heading, *avg,
2057                                     width=width, digits=digits)
2058
2059     if output_dict:
2060         if 'accuracy' in report_dict.keys():

```



```

2061         report_dict['accuracy'] = report_dict['accuracy']['precision']
2062     return report_dict
2063 else:
2064     return report
2065
2066
2067 @_deprecated_positional_args
2068 def hamming_loss(y_true, y_pred, *, sample_weight=None):
2069     """Compute the average Hamming loss.
2070
2071     The Hamming loss is the fraction of labels that are incorrectly predicted.
2072
2073     Read more in the :ref:`User Guide <hamming_loss>`.
2074
2075     Parameters
2076     -----
2077     y_true : 1d array-like, or label indicator array / sparse matrix
2078         Ground truth (correct) labels.
2079
2080     y_pred : 1d array-like, or label indicator array / sparse matrix
2081         Predicted labels, as returned by a classifier.
2082
2083     sample_weight : array-like of shape (n_samples,), default=None
2084         Sample weights.
2085
2086     .. versionadded:: 0.18
2087
2088     Returns
2089     -----
2090     loss : float or int
2091         Return the average Hamming loss between element of ``y_true`` and
2092         ``y_pred``.
2093
2094     See Also
2095     -----
2096     accuracy_score, jaccard_score, zero_one_loss
2097
2098     Notes
2099     -----
2100     In multiclass classification, the Hamming loss corresponds to the Hamming
2101     distance between ``y_true`` and ``y_pred`` which is equivalent to the
2102     subset ``zero_one_loss`` function, when ``normalize`` parameter is set to
2103     True.
2104
2105     In multilabel classification, the Hamming loss is different from the
2106     subset zero-one loss. The zero-one loss considers the entire set of labels
2107     for a given sample incorrect if it does not entirely match the true set of
2108     labels. Hamming loss is more forgiving in that it penalizes only the
2109     individual labels.
2110
2111     The Hamming loss is upperbounded by the subset zero-one loss, when
2112     ``normalize`` parameter is set to True. It is always between 0 and 1,

```

lower being better.

References

.. [1] Grigorios Tsoumakas, Ioannis Katakis. Multi-Label Classification: An Overview. International Journal of Data Warehousing & Mining, 3(3), 1-13, July-September 2007.

.. [2] `Wikipedia entry on the Hamming distance
<https://en.wikipedia.org/wiki/Hamming_distance>`_.

Examples

```
>>> from sklearn.metrics import hamming_loss
>>> y_pred = [1, 2, 3, 4]
>>> y_true = [2, 2, 3, 4]
>>> hamming_loss(y_true, y_pred)
0.25
```

In the multilabel case with binary label indicators:

```
>>> import numpy as np
>>> hamming_loss(np.array([[0, 1], [1, 1]]), np.zeros((2, 2)))
0.75
"""
```

```
y_type, y_true, y_pred = _check_targets(y_true, y_pred)
check_consistent_length(y_true, y_pred, sample_weight)
```

```
if sample_weight is None:
    weight_average = 1.
else:
    weight_average = np.mean(sample_weight)

if y_type.startswith('multilabel'):
    n_differences = count_nonzero(y_true - y_pred,
                                  sample_weight=sample_weight)
    return (n_differences /
            (y_true.shape[0] * y_true.shape[1] * weight_average))

elif y_type in ["binary", "multiclass"]:
    return _weighted_sum(y_true != y_pred, sample_weight, normalize=True)
else:
    raise ValueError("{0} is not supported".format(y_type))
```

@_deprecate_positional_args

```
def log_loss(y_true, y_pred, *, eps=1e-15, normalize=True, sample_weight=None,
             labels=None):
    r"""Log loss, aka logistic loss or cross-entropy loss.
```

This is the loss function used in (multinomial) logistic regression

and extensions of it such as neural networks, defined as the negative log-likelihood of a logistic model that returns ``y_pred`` probabilities for its training data ``y_true``.

The log loss is only defined for two or more labels.

For a single sample with true label $y \in \{0,1\}$ and a probability estimate $p = \operatorname{Pr}(y = 1)$, the log loss is:

```
.. math::
    L_{\log}(y, p) = -(y \log(p) + (1 - y) \log(1 - p))
```

Read more in the :ref:`User Guide <log_loss>`.

Parameters

y_true : array-like or label indicator matrix

Ground truth (correct) labels for `n_samples` samples.

y_pred : array-like of float, shape = (`n_samples`, `n_classes`) or (`n_samples`,)

Predicted probabilities, as returned by a classifier's `predict_proba` method. If ``y_pred.shape = (n_samples,)`` the probabilities provided are assumed to be that of the positive class. The labels in ``y_pred`` are assumed to be ordered alphabetically, as done by `:class:`preprocessing.LabelBinarizer``.

eps : float, default=1e-15

Log loss is undefined for $p=0$ or $p=1$, so probabilities are clipped to $\max(\text{eps}, \min(1 - \text{eps}, p))$.

normalize : bool, default=True

If true, return the mean loss per sample.

Otherwise, return the sum of the per-sample losses.

sample_weight : array-like of shape (`n_samples`,), default=None

Sample weights.

labels : array-like, default=None

If not provided, labels will be inferred from `y_true`. If ``labels`` is ``None`` and ``y_pred`` has shape (`n_samples`,) the labels are assumed to be binary and are inferred from ``y_true``.

.. versionadded:: 0.18

Returns

loss : float

Notes

The logarithm used is the natural logarithm (base-e).

Examples

```
>>> from sklearn.metrics import log_loss
>>> log_loss(["spam", "ham", "ham", "spam"],
...          [[.1, .9], [.9, .1], [.8, .2], [.35, .65]])
0.21616...
```

References

C.M. Bishop (2006). Pattern Recognition and Machine Learning. Springer, p. 209.

"""

```
y_pred = check_array(y_pred, ensure_2d=False)
check_consistent_length(y_pred, y_true, sample_weight)
```

```
lb = LabelBinarizer()
```

```
if labels is not None:
```

```
    lb.fit(labels)
```

```
else:
```

```
    lb.fit(y_true)
```

```
if len(lb.classes_) == 1:
```

```
    if labels is None:
```

```
        raise ValueError('y_true contains only one label ({0}). Please '
                           'provide the true labels explicitly through the '
                           'labels argument.'.format(lb.classes_[0]))
```

```
    else:
```

```
        raise ValueError('The labels array needs to contain at least two '
                           'labels for log_loss, '
                           'got {0}'.format(lb.classes_))
```

```
transformed_labels = lb.transform(y_true)
```

```
if transformed_labels.shape[1] == 1:
```

```
    transformed_labels = np.append(1 - transformed_labels,
                                   transformed_labels, axis=1)
```

```
# Clipping
```

```
y_pred = np.clip(y_pred, eps, 1 - eps)
```

```
# If y_pred is of single dimension, assume y_true to be binary
```

```
# and then check.
```

```
if y_pred.ndim == 1:
```

```
    y_pred = y_pred[:, np.newaxis]
```

```
if y_pred.shape[1] == 1:
```

```
    y_pred = np.append(1 - y_pred, y_pred, axis=1)
```

```
# Check if dimensions are consistent.
```

```
transformed_labels = check_array(transformed_labels)
```

```
if len(lb.classes_) != y_pred.shape[1]:
```

```
    if labels is None:
```

```

2269         raise ValueError("y_true and y_pred contain different number of "
2270                             "classes {0}, {1}. Please provide the true "
2271                             "labels explicitly through the labels argument. "
2272                             "Classes found in "
2273                             "y_true: {2}".format(transformed_labels.shape[1],
2274                                                  y_pred.shape[1],
2275                                                  lb.classes_))
2276     else:
2277         raise ValueError('The number of classes in labels is different '
2278                             'from that in y_pred. Classes found in '
2279                             'labels: {0}'.format(lb.classes_))
2280
2281     # Renormalize
2282     y_pred /= y_pred.sum(axis=1)[:, np.newaxis]
2283     loss = -(transformed_labels * np.log(y_pred)).sum(axis=1)
2284
2285     return _weighted_sum(loss, sample_weight, normalize)
2286
2287
2288 @_deprecate_positional_args
2289 def hinge_loss(y_true, pred_decision, *, labels=None, sample_weight=None):
2290     """Average hinge loss (non-regularized).
2291
2292     In binary class case, assuming labels in y_true are encoded with +1 and -1,
2293     when a prediction mistake is made, ``margin = y_true * pred_decision`` is
2294     always negative (since the signs disagree), implying ``1 - margin`` is
2295     always greater than 1. The cumulated hinge loss is therefore an upper
2296     bound of the number of mistakes made by the classifier.
2297
2298     In multiclass case, the function expects that either all the labels are
2299     included in y_true or an optional labels argument is provided which
2300     contains all the labels. The multilabel margin is calculated according
2301     to Crammer-Singer's method. As in the binary case, the cumulated hinge loss
2302     is an upper bound of the number of mistakes made by the classifier.
2303
2304     Read more in the :ref:`User Guide <hinge_loss>`.
2305
2306     Parameters
2307     -----
2308     y_true : array of shape (n_samples,)
2309         True target, consisting of integers of two values. The positive label
2310         must be greater than the negative label.
2311
2312     pred_decision : array of shape (n_samples,) or (n_samples, n_classes)
2313         Predicted decisions, as output by decision_function (floats).
2314
2315     labels : array-like, default=None
2316         Contains all the labels for the problem. Used in multiclass hinge loss.
2317
2318     sample_weight : array-like of shape (n_samples,), default=None
2319         Sample weights.
2320

```

```

2321 Returns
2322 -----
2323 loss : float
2324
2325 References
2326 -----
2327 .. [1] `Wikipedia entry on the Hinge loss
2328      <https://en.wikipedia.org/wiki/Hinge\_loss>`_.
2329
2330 .. [2] Koby Crammer, Yoram Singer. On the Algorithmic
2331      Implementation of Multiclass Kernel-based Vector
2332      Machines. Journal of Machine Learning Research 2,
2333      (2001), 265-292.
2334
2335 .. [3] `L1 AND L2 Regularization for Multiclass Hinge Loss Models
2336      by Robert C. Moore, John DeNero
2337      <http://www.ttic.edu/sigml/symposium2011/papers/
2338      Moore+DeNero_Regularization.pdf>`_.
2339
2340 Examples
2341 -----
2342 >>> from sklearn import svm
2343 >>> from sklearn.metrics import hinge_loss
2344 >>> X = [[0], [1]]
2345 >>> y = [-1, 1]
2346 >>> est = svm.LinearSVC(random_state=0)
2347 >>> est.fit(X, y)
2348 LinearSVC(random_state=0)
2349 >>> pred_decision = est.decision_function([[-2], [3], [0.5]])
2350 >>> pred_decision
2351 array([-2.18...,  2.36...,  0.09...])
2352 >>> hinge_loss([-1, 1, 1], pred_decision)
2353 0.30...
2354
2355 In the multiclass case:
2356
2357 >>> import numpy as np
2358 >>> X = np.array([[0], [1], [2], [3]])
2359 >>> Y = np.array([0, 1, 2, 3])
2360 >>> labels = np.array([0, 1, 2, 3])
2361 >>> est = svm.LinearSVC()
2362 >>> est.fit(X, Y)
2363 LinearSVC()
2364 >>> pred_decision = est.decision_function([[-1], [2], [3]])
2365 >>> y_true = [0, 2, 3]
2366 >>> hinge_loss(y_true, pred_decision, labels=labels)
2367 0.56...
2368 """"
2369 check_consistent_length(y_true, pred_decision, sample_weight)
2370 pred_decision = check_array(pred_decision, ensure_2d=False)
2371 y_true = column_or_1d(y_true)
2372 y_true_unique = np.unique(labels if labels is not None else y_true)

```

```

2373     if y_true_unique.size > 2:
2374         if (labels is None and pred_decision.ndim > 1 and
2375             (np.size(y_true_unique) != pred_decision.shape[1])):
2376             raise ValueError("Please include all labels in y_true "
2377                              "or pass labels as third argument")
2378         if labels is None:
2379             labels = y_true_unique
2380         le = LabelEncoder()
2381         le.fit(labels)
2382         y_true = le.transform(y_true)
2383         mask = np.ones_like(pred_decision, dtype=bool)
2384         mask[np.arange(y_true.shape[0]), y_true] = False
2385         margin = pred_decision[~mask]
2386         margin -= np.max(pred_decision[mask].reshape(y_true.shape[0], -1),
2387                        axis=1)
2388
2389     else:
2390         # Handles binary class case
2391         # this code assumes that positive and negative labels
2392         # are encoded as +1 and -1 respectively
2393         pred_decision = column_or_1d(pred_decision)
2394         pred_decision = np.ravel(pred_decision)
2395
2396         lbin = LabelBinarizer(neg_label=-1)
2397         y_true = lbin.fit_transform(y_true)[: , 0]
2398
2399         try:
2400             margin = y_true * pred_decision
2401         except TypeError:
2402             raise TypeError("pred_decision should be an array of floats.")
2403
2404     losses = 1 - margin
2405     # The hinge_loss doesn't penalize good enough predictions.
2406     np.clip(losses, 0, None, out=losses)
2407     return np.average(losses, weights=sample_weight)
2408
2409
2410 @_deprecated_positional_args
2411 def brier_score_loss(y_true, y_prob, *, sample_weight=None, pos_label=None):
2412     """Compute the Brier score loss.
2413
2414     The smaller the Brier score loss, the better, hence the naming with "loss".
2415     The Brier score measures the mean squared difference between the predicted
2416     probability and the actual outcome. The Brier score always
2417     takes on a value between zero and one, since this is the largest
2418     possible difference between a predicted probability (which must be
2419     between zero and one) and the actual outcome (which can take on values
2420     of only 0 and 1). It can be decomposed is the sum of refinement loss and
2421     calibration loss.
2422
2423     The Brier score is appropriate for binary and categorical outcomes that
2424     can be structured as true or false, but is inappropriate for ordinal

```

variables which can take on three or more values (this is because the Brier score assumes that all possible outcomes are equivalently "distant" from one another). Which label is considered to be the positive label is controlled via the parameter ``pos_label``, which defaults to the greater label unless ``y_true`` is all 0 or all -1, in which case ``pos_label`` defaults to 1.

Read more in the :ref:`User Guide <brier_score_loss>`.

Parameters

`y_true` : array of shape (n_samples,)

True targets.

`y_prob` : array of shape (n_samples,)

Probabilities of the positive class.

`sample_weight` : array-like of shape (n_samples,), default=None

Sample weights.

`pos_label` : int or str, default=None

Label of the positive class. ``pos_label`` will be inferred in the following manner:

- * if ``y_true`` in {-1, 1} or {0, 1}, ``pos_label`` defaults to 1;
- * else if ``y_true`` contains string, an error will be raised and ``pos_label`` should be explicitly specified;
- * otherwise, ``pos_label`` defaults to the greater label, i.e. ``np.unique(y_true)[-1]``.

Returns

`score` : float

Brier score loss.

Examples

```
>>> import numpy as np
>>> from sklearn.metrics import brier_score_loss
>>> y_true = np.array([0, 1, 1, 0])
>>> y_true_categorical = np.array(["spam", "ham", "ham", "spam"])
>>> y_prob = np.array([0.1, 0.9, 0.8, 0.3])
>>> brier_score_loss(y_true, y_prob)
0.037...
>>> brier_score_loss(y_true, 1-y_prob, pos_label=0)
0.037...
>>> brier_score_loss(y_true_categorical, y_prob, pos_label="ham")
0.037...
>>> brier_score_loss(y_true, np.array(y_prob) > 0.5)
0.0
```

References


```

2477 -----
2478 .. [1] `Wikipedia entry for the Brier score
2479      <https://en.wikipedia.org/wiki/Brier\_score>`.
2480 """
2481 y_true = column_or_1d(y_true)
2482 y_prob = column_or_1d(y_prob)
2483 assert_all_finite(y_true)
2484 assert_all_finite(y_prob)
2485 check_consistent_length(y_true, y_prob, sample_weight)
2486
2487 y_type = type_of_target(y_true)
2488 if y_type != "binary":
2489     raise ValueError(
2490         f"Only binary classification is supported. The type of the target "
2491         f"is {y_type}."
2492     )
2493
2494 if y_prob.max() > 1:
2495     raise ValueError("y_prob contains values greater than 1.")
2496 if y_prob.min() < 0:
2497     raise ValueError("y_prob contains values less than 0.")
2498
2499 try:
2500     pos_label = _check_pos_label_consistency(pos_label, y_true)
2501 except ValueError:
2502     classes = np.unique(y_true)
2503     if classes.dtype.kind not in ('O', 'U', 'S'):
2504         # for backward compatibility, if classes are not string then
2505         # `pos_label` will correspond to the greater label
2506         pos_label = classes[-1]
2507     else:
2508         raise
2509 y_true = np.array(y_true == pos_label, int)
2510 return np.average((y_true - y_prob) ** 2, weights=sample_weight)

```