Classification_report get precision

def multi_class_classification(data_X.data_Y): ''' calculate the classification of multiple classes and return related evaluation metrics ''' svc = svm. SVC(C=1, kernel='linear') # X_train, X_test, y_train, y_test = train_test_split(data_X, data_Y, test_size=0.4, random_state=0) clf = svc.fit(data_X, data_Y) #svm # array = svc.coef_ # print array predicted = cross_val_predict(clf, data_X, data_Y, cv=2) print precision, metrics.accuracy_score(data_Y, predicted) to print f1 score macro, metrics.f1_score(data_Y, predicted, average='macro') print f1 score micro, metrics.f1_score(data_Y, predicted, average='micro') print precision score, metrics.precision_score (data_Y, predicted, average ='macro') print recall score,metrics.recall_score (data_Y, predicted, average='macro') print hamming_loss, metrics.hamming_loss(data_Y, predicted) print classification_report, metrics.classification_report(data_Y, predicted) print jaccard_similarity_score, metrics.jaccard_similarity_score(data_Y , he predicted) # print log_loss, Metrics.log_loss(data_Y, predicted) print zero_one_loss, metrics.zero_one_loss(data_Y, predicted) # print AUC&roc,metrics.roc_auc_score(data_Y, predicted) # print matthews_corrcoef, metrics.matthews_corrcoef(data_Y, predicted) Photo by Mika Baumeister in UnsplashThese follow-up to Confused About The Confusion Matrix?. Please read this first one before reading this how I will use the examples of this blog post. Original example: Classification models have been trained to predict whether a person likes dogs (True) or not (False) based on different variables. In the previous blog post I ended up using the hypothetical naive bayes model. If the matrix of confusion doesn't look so perfect what do we do? In this case, we will see how to calculate the scikit-learn ranking report. Let's take a look at the example of the table of confusion of the previous post and explain what the terms mean. No, no, no, no. True Negative: The actual value was False, and the model predicted False. — It has been correctly identified that the person does not like dogs. No, no, no, no. False positive: the real value was false, and the model predicted True. (FN) False negative: the actual value was True, and the model predicted False. This is also known as a Type II bug. — He predicted that no, the person does not like dogs, but actually they do. (TP) True Positive: The real value was true, and the model predicted True – It was correctly identified that the person does like dogs. What's the real question we're trying to answer? Do we want to have fewer false positives, fewer false negatives or global accuracy? No model will correctly identify, with 100% certainty, true negative and true positive values, there is always a compromise. We pretend that we are dealing with to a rescue shelter for cats and dogs. The rescue shelter has a questionnaire for people who do not know want a cat or a dog. From the questionnaire, the assigned agent will be able to determine if the person will like cats or dogs more. The shelter wants fewer people to return their original adopted pet. Accuracy, Recall, F1-Score, AccuracyHow do you predict that yes the person likes dogs, how often is it really right? When in fact it is yes the person likes dogs, how often do you predict correctly? Weighted average between precision and memory. Useful when it comes to unbalanced samples. The sum of true positives and true negatives divided by the total number of samples. This is only accurate if the model is balanced. It will give inaccurate results if there is a class imbalance. Let's compare this with the example of logistic regression confusion matrixThe example of the Confusion Matrix Decision Tree Example of the confusion matrixBased in the above performance metrics, I will choose overall accuracy. Since our data is balanced, which means a division between 50/50 true and negative samples, I choose accuracy. The best model that gives me the best precision is the logistic regression model. Useful Links / Resources:- Classification Report- Type I and II errors- Precision and Recall- Model Selection: Accuracy, Precision, Recall or F1?- What's the deal with Accuracy, Precision, Recall and F1?- Simple guide to confusion matrix terminology Stanford University Istanbul Aydin University Stanford University RMIT University Faculty of Medicine and Pharmacy of Rabat This is a simple example of classification_report in sklearn 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 # # 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 # #avg / total 0.70 0.60 0.61 5 I want to have access to avg/total row. For example, I want to extract the f1 score from the report, which is 0.61. How can I access the number in classification_report? Calculate accuracy, remember, F-measure and support for each class Accuracy is the tp/ (tp + fp) ratio where tp is the number of true positives and fp the number of false positives. Accuracy is intuitively the ability of classifiers not to label as positive a sample that is negative. The recovery is the tp/(tp+fn) ratio where the TP is the number of true positives and the number of false negatives. The recall is intuitively the ability of the classifiers to find all the positive samples. The F-beta score can be interpreted as a weighted harmonic average of accuracy and memory, where an F-beta score reaches its best value at 1 and the worst score at 0. F-beta scoring weights are remembered more than accuracy by a beta factor. beta == 1.0 means remembering and are equally important. Support is the number of occurrences of each class in y_true. If pos_label is none and binary this function returns the average accuracy, memory and measure F if the average is one of the micro, macro, weighted or samples. Read more in the User Guide. Parameters y_true1d such as array, or array of indicators of labels / array scarce Target values (correct). y_pred1d as an array, or matrix of indicators of labels / matrix scarceEstimated targets as returned by a classifiers. betafloat, 1.0 by defaultThe strength of recovery versus accuracy in the F. tag list score, optionalThe set of tags to include when average != 'binary', and its order if average is None. Tags present in the data can be excluded, for example, to calculate a multiclass mean by ignoring a majority negative class, while the tags not present in the data will result in 0 components in an average macro. For multilaphile goals, labels are column indexes. By default, all labels y_true and y_pred are used in order order. pos_labelstr or int, 1 by defaultThe class to report if average='binary' and the data are binary. If the data is multiclass or multi-roof, this will be ignored; set tags=[pos_label] and average != 'binary' will only report ratings for this tag. average, [None (default), 'binary', 'micro', 'macro', 'samples', 'weighted']If none, the scores of each class are returned. Otherwise, this determines the average type performed on the data: 'binary':Only the report results for the class specified by pos_label. This applies only if the y_ {true,pred}) are binary. 'micro':Calculate metrics globally counting the total true positives, false negatives and false positives. 'macro': Calculate the metrics for each label and find its weightless average. This does not take into account the imbalance of labels. 'Weighted': Calculate the metrics for each label and find the average weighted by support (the number of actual instances for each label). This alters the 'macro' to explain the tag imbalance; can result in an F-score that is not between accuracy and memory. 'samples':Calculate metrics for each instance, and find its average (only significant for multilaphile classification where this differs from accuracy_score). warn_fortuple or set, for internal use This determines what warnings will be made in case this feature is being used to return only one of its metrics. sample_weightarray (n_samples),default=NoneSample pesos. zero_divisionadvertir, 0 or 1, default=warn Sets the value to return when there is zero division: remember: when there are no positive precision labels: when there are no positive f-score predictions: whether set to warn, this acts as 0, but warnings are also added. Returns precision lafloate (if average is none) or a variety of floats, shape = returnedafloat (if the average is none) or the float array, form = [n_unique_labels] fbeta_scorefloat (if the average is none) or float array, form = [n_unique_labels] supportNone (if the average is none) or int variety, form = [n_unique_labels] The number of of each label in y_true. Notes When true positive + false positive == 0, accuracy is not defined; When the true positive + false negative == 0, the record is not defined. In these cases, by default, the metric will be set to 0, just like the f-score, and undefined warming will be raised. This behavior can be modified with zero_division. References 1 Wikipedia entry for accuracy and remember 2 Wikipedia entry for F1 score 3 Discriminatory methods for multilabeled advances in knowledge discovery and data mining classification (2004), pp. 22-30 by Shantanu Godbole, Sunita Sarawagi Examples >>> numpy import as np >>> from sklearn.metrics amount precision_recall_fscore_support >>> y_true = np.array(['cat' , 'dog', 'pig', 'cat', 'dog', 'pig']) >>> y_pred= np.array(['cat', 'pig', 'dog', 'cat', 'cat']) >>> precision_recall_fscore_support(y_true, y_pred, media='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 calculate precisions by label, remember, F1-scores and media instead of average : >>> 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]))