Unsurprisingly, precision and recall are a perfect 1 for class 0, as there are no confusions with this class. For class 7, on the other hand, precision is 1 because no other class was mistakenly classified as 7, while for class 6, there are no false negatives, so the recall is 1. We can also see that the model has particular difficulties with classes 8 and 3.

The most commonly used metric for imbalanced datasets in the multiclass setting is the multiclass version of the *f*-score. The idea behind the multiclass *f*-score is to compute one binary *f*-score per class, with that class being the positive class and the other classes making up the negative classes. Then, these per-class *f*-scores are averaged using one of the following strategies:

- `"macro"` averaging computes the unweighted per-class *f*-scores. This gives equal weight to all classes, no matter what their size is.
- `"weighted"` averaging computes the mean of the per-class *f*-scores, weighted by their support. This is what is reported in the classification report.
- `"micro"` averaging computes the total number of false positives, false negatives, and true positives over all classes, and then computes precision, recall, and *f*-score using these counts.

If you care about each *sample* equally much, it is recommended to use the `"micro"` average f_1-score; if you care about each *class* equally much, it is recommended to use the `"macro"` average f_1-score:

In[66]:

```
print("Micro average f1 score: {:.3f}".format
      (f1_score(y_test, pred, average="micro")))
print("Macro average f1 score: {:.3f}".format
      (f1_score(y_test, pred, average="macro")))
```

Out[66]:

```
Micro average f1 score: 0.953
Macro average f1 score: 0.954
```

Regression Metrics

Evaluation for regression can be done in similar detail as we did for classification—for example, by analyzing overpredicting the target versus underpredicting the target. However, in most applications we've seen, using the default R^2 used in the `score` method of all regressors is enough. Sometimes business decisions are made on the basis of mean squared error or mean absolute error, which might give incentive to tune models using these metrics. In general, though, we have found R^2 to be a more intuitive metric to evaluate regression models.

Using Evaluation Metrics in Model Selection

We have discussed many evaluation methods in detail, and how to apply them given the ground truth and a model. However, we often want to use metrics like AUC in model selection using `GridSearchCV` or `cross_val_score`. Luckily `scikit-learn` provides a very simple way to achieve this, via the `scoring` argument that can be used in both `GridSearchCV` and `cross_val_score`. You can simply provide a string describing the evaluation metric you want to use. Say, for example, we want to evaluate the SVM classifier on the "nine vs. rest" task on the `digits` dataset, using the AUC score. Changing the score from the default (accuracy) to AUC can be done by providing `"roc_auc"` as the `scoring` parameter:

In[67]:

```
# default scoring for classification is accuracy
print("Default scoring: {}".format(
    cross_val_score(SVC(), digits.data, digits.target == 9)))
# providing scoring="accuracy" doesn't change the results
explicit_accuracy =  cross_val_score(SVC(), digits.data, digits.target == 9,
                                     scoring="accuracy")
print("Explicit accuracy scoring: {}".format(explicit_accuracy))
roc_auc =  cross_val_score(SVC(), digits.data, digits.target == 9,
                           scoring="roc_auc")
print("AUC scoring: {}".format(roc_auc))
```

Out[67]:

```
Default scoring: [ 0.9  0.9  0.9]
Explicit accuracy scoring: [ 0.9  0.9  0.9]
AUC scoring: [ 0.994  0.99   0.996]
```

Similarly, we can change the metric used to pick the best parameters in `GridSearchCV`:

In[68]:

```
X_train, X_test, y_train, y_test = train_test_split(
    digits.data, digits.target == 9, random_state=0)

# we provide a somewhat bad grid to illustrate the point:
param_grid = {'gamma': [0.0001, 0.01, 0.1, 1, 10]}
# using the default scoring of accuracy:
grid = GridSearchCV(SVC(), param_grid=param_grid)
grid.fit(X_train, y_train)
print("Grid-Search with accuracy")
print("Best parameters:", grid.best_params_)
print("Best cross-validation score (accuracy)): {:.3f}".format(grid.best_score_))
print("Test set AUC: {:.3f}".format(
    roc_auc_score(y_test, grid.decision_function(X_test))))
print("Test set accuracy: {:.3f}".format(grid.score(X_test, y_test)))
```