Out[62]:

```
gamma = 1.00  accuracy = 0.90  AUC = 0.50
gamma = 0.05  accuracy = 0.90  AUC = 0.90
gamma = 0.01  accuracy = 0.90  AUC = 1.00
```

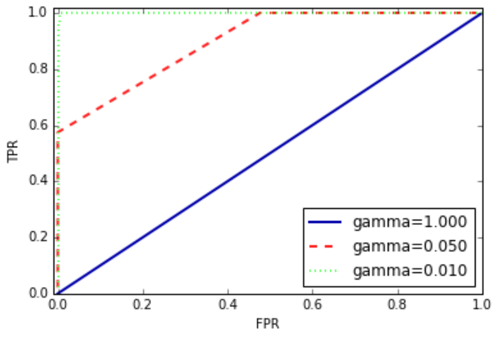

Figure 5-17. Comparing ROC curves of SVMs with different settings of gamma

The accuracy of all three settings of `gamma` is the same, 90%. This might be the same as chance performance, or it might not. Looking at the AUC and the corresponding curve, however, we see a clear distinction between the three models. With `gamma=1.0`, the AUC is actually at chance level, meaning that the output of the `decision_function` is as good as random. With `gamma=0.05`, performance drastically improves to an AUC of 0.5. Finally, with `gamma=0.01`, we get a perfect AUC of 1.0. That means that all positive points are ranked higher than all negative points according to the decision function. In other words, with the right threshold, this model can classify the data perfectly![5] Knowing this, we can adjust the threshold on this model and obtain great predictions. If we had only used accuracy, we would never have discovered this.

For this reason, we highly recommend using AUC when evaluating models on imbalanced data. Keep in mind that AUC does not make use of the default threshold, though, so adjusting the decision threshold might be necessary to obtain useful classification results from a model with a high AUC.

Metrics for Multiclass Classification

Now that we have discussed evaluation of binary classification tasks in depth, let's move on to metrics to evaluate multiclass classification. Basically, all metrics for multiclass classification are derived from binary classification metrics, but averaged

5 Looking at the curve for `gamma=0.01` in detail, you can see a small kink close to the top left. That means that at least one point was not ranked correctly. The AUC of 1.0 is a consequence of rounding to the second decimal point.

over all classes. Accuracy for multiclass classification is again defined as the fraction of correctly classified examples. And again, when classes are imbalanced, accuracy is not a great evaluation measure. Imagine a three-class classification problem with 85% of points belonging to class A, 10% belonging to class B, and 5% belonging to class C. What does being 85% accurate mean on this dataset? In general, multiclass classification results are harder to understand than binary classification results. Apart from accuracy, common tools are the confusion matrix and the classification report we saw in the binary case in the previous section. Let's apply these two detailed evaluation methods on the task of classifying the 10 different handwritten digits in the `digits` dataset:

In[63]:

```
from sklearn.metrics import accuracy_score
X_train, X_test, y_train, y_test = train_test_split(
    digits.data, digits.target, random_state=0)
lr = LogisticRegression().fit(X_train, y_train)
pred = lr.predict(X_test)
print("Accuracy: {:.3f}".format(accuracy_score(y_test, pred)))
print("Confusion matrix:\n{}".format(confusion_matrix(y_test, pred)))
```

Out[63]:

```
Accuracy: 0.953
Confusion matrix:
[[37  0  0  0  0  0  0  0  0  0]
 [ 0 39  0  0  0  0  2  0  2  0]
 [ 0  0 41  3  0  0  0  0  0  0]
 [ 0  0  1 43  0  0  0  0  0  1]
 [ 0  0  0  0 38  0  0  0  0  0]
 [ 0  1  0  0  0 47  0  0  0  0]
 [ 0  0  0  0  0  0 52  0  0  0]
 [ 0  1  0  1  1  0  0 45  0  0]
 [ 0  3  1  0  0  0  0  0 43  1]
 [ 0  0  0  1  0  1  0  0  1 44]]
```

The model has an accuracy of 95.3%, which already tells us that we are doing pretty well. The confusion matrix provides us with some more detail. As for the binary case, each row corresponds to a true label, and each column corresponds to a predicted label. You can find a visually more appealing plot in Figure 5-18:

In[64]:

```
scores_image = mglearn.tools.heatmap(
    confusion_matrix(y_test, pred), xlabel='Predicted label',
    ylabel='True label', xticklabels=digits.target_names,
    yticklabels=digits.target_names, cmap=plt.cm.gray_r, fmt="%d")
plt.title("Confusion matrix")
plt.gca().invert_yaxis()
```

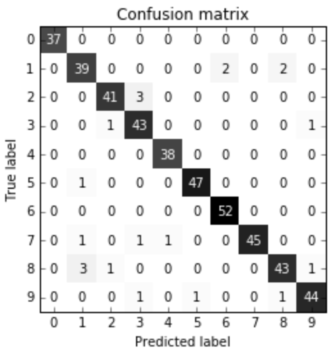

Figure 5-18. Confusion matrix for the 10-digit classification task

For the first class, the digit 0, there are 37 samples in the class, and all of these samples were classified as class 0 (there are no false negatives for class 0). We can see that because all other entries in the first row of the confusion matrix are 0. We can also see that no other digits were mistakenly classified as 0, because all other entries in the first column of the confusion matrix are 0 (there are no false positives for class 0). Some digits were confused with others, though—for example, the digit 2 (third row), three of which were classified as the digit 3 (fourth column). There was also one digit 3 that was classified as 2 (third column, fourth row) and one digit 8 that was classified as 2 (thrid column, fourth row).

With the `classification_report` function, we can compute the precision, recall, and *f*-score for each class:

In[65]:

```
print(classification_report(y_test, pred))
```

Out[65]:

```
             precision    recall  f1-score   support

          0       1.00      1.00      1.00        37
          1       0.89      0.91      0.90        43
          2       0.95      0.93      0.94        44
          3       0.90      0.96      0.92        45
          4       0.97      1.00      0.99        38
          5       0.98      0.98      0.98        48
          6       0.96      1.00      0.98        52
          7       1.00      0.94      0.97        48
          8       0.93      0.90      0.91        48
          9       0.96      0.94      0.95        47

avg / total       0.95      0.95      0.95       450
```

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.