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)))
```

Out[68]:

```
Grid-Search with accuracy
Best parameters: {'gamma': 0.0001}
Best cross-validation score (accuracy)): 0.970
Test set AUC: 0.992
Test set accuracy: 0.973
```

In[69]:

```
# using AUC scoring instead:
grid = GridSearchCV(SVC(), param_grid=param_grid, scoring="roc_auc")
grid.fit(X_train, y_train)
print("\nGrid-Search with AUC")
print("Best parameters:", grid.best_params_)
print("Best cross-validation score (AUC): {:.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)))
```

Out[69]:

```
Grid-Search with AUC
Best parameters: {'gamma': 0.01}
Best cross-validation score (AUC): 0.997
Test set AUC: 1.000
Test set accuracy: 1.000
```

When using accuracy, the parameter `gamma=0.0001` is selected, while `gamma=0.01` is selected when using AUC. The cross-validation accuracy is consistent with the test set accuracy in both cases. However, using AUC found a better parameter setting in terms of AUC and even in terms of accuracy.[6]

The most important values for the `scoring` parameter for classification are `accuracy` (the default); `roc_auc` for the area under the ROC curve; `average_precision` for the area under the precision-recall curve; `f1`, `f1_macro`, `f1_micro`, and `f1_weighted` for the binary f_1-score and the different weighted variants. For regression, the most commonly used values are `r2` for the R^2 score, `mean_squared_error` for mean squared error, and `mean_absolute_error` for mean absolute error. You can find a full list of supported arguments in the documentation or by looking at the `SCORER` dictionary defined in the `metrics.scorer` module:

6 Finding a higher-accuracy solution using AUC is likely a consequence of accuracy being a bad measure of model performance on imbalanced data.

In[70]:

```
from sklearn.metrics.scorer import SCORERS
print("Available scorers:\n{}".format(sorted(SCORERS.keys())))
```

Out[70]:

```
Available scorers:
['accuracy', 'adjusted_rand_score', 'average_precision', 'f1', 'f1_macro',
 'f1_micro', 'f1_samples', 'f1_weighted', 'log_loss', 'mean_absolute_error',
 'mean_squared_error', 'median_absolute_error', 'precision', 'precision_macro',
 'precision_micro', 'precision_samples', 'precision_weighted', 'r2', 'recall',
 'recall_macro', 'recall_micro', 'recall_samples', 'recall_weighted', 'roc_auc']
```

Summary and Outlook

In this chapter we discussed cross-validation, grid search, and evaluation metrics, the cornerstones of evaluating and improving machine learning algorithms. The tools described in this chapter, together with the algorithms described in Chapters 2 and 3, are the bread and butter of every machine learning practitioner.

There are two particular points that we made in this chapter that warrant repeating, because they are often overlooked by new practitioners. The first has to do with cross-validation. Cross-validation or the use of a test set allow us to evaluate a machine learning model as it will perform in the future. However, if we use the test set or cross-validation to select a model or select model parameters, we "use up" the test data, and using the same data to evaluate how well our model will do in the future will lead to overly optimistic estimates. We therefore need to resort to a split into training data for model building, validation data for model and parameter selection, and test data for model evaluation. Instead of a simple split, we can replace each of these splits with cross-validation. The most commonly used form (as described earlier) is a training/test split for evaluation, and using cross-validation on the training set for model and parameter selection.

The second point has to do with the importance of the evaluation metric or scoring function used for model selection and model evaluation. The theory of how to make business decisions from the predictions of a machine learning model is somewhat beyond the scope of this book.[7] However, it is rarely the case that the end goal of a machine learning task is building a model with a high accuracy. Make sure that the metric you choose to evaluate and select a model for is a good stand-in for what the model will actually be used for. In reality, classification problems rarely have balanced classes, and often false positives and false negatives have very different consequences.

7 We highly recommend Foster Provost and Tom Fawcett's book *Data Science for Business* (O'Reilly) for more information on this topic.