Model selection with Scikit-Learn

Training error

```
[22]: from sklearn.neighbors import KNeighborsClassifier
      from sklearn.metrics import zero_one_loss
      from sklearn.datasets import make_blobs

      # Blob data
      X, y = make_blobs(n_samples=1000, centers=20, random_state=123)
      labels = ["b", "r"]
      y = np.take(labels, (y < 10)) # Relabels numeric values to b,r

      clf = KNeighborsClassifier()
      clf.fit(X, y)
      print("Training error =", zero_one_loss(y, clf.predict(X)))
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=5, p=2,
           weights='uniform')


Training error = 0.108
```

Test error

Issue: the training error is a **biased** estimate of the generalization error.

Solution: Divide data into two disjoint parts called training and test sets (usually using 70% for training and 30% for test).

- Use the training set for fitting the model;
- Use the test set for evaluation only, thereby yielding an unbiased estimate.
- **The same data should not be used both for training and evaluation.**

```
[23]: from sklearn.model_selection import train_test_split
      X_train, X_test, y_train, y_test = train_test_split(X, y)
      clf = KNeighborsClassifier()
      clf.fit(X_train, y_train)
      print("Training error =", zero_one_loss(y_train, clf.predict(X_train)))
      print("Test error =", zero_one_loss(y_test, clf.predict(X_test)))
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=5, p=2,
           weights='uniform')


Training error = 0.0973333333333
Test error = 0.172
```

Cross-validation

Issue:

- When data is small, training on 70% of the data may lead to a model that is significantly different from a model that would have been learned on the entire set.
- Yet, increasing the size of the training set (resp. decreasing the size of the test set), might lead to an inaccurate estimate of the generalization error.

Solution: K-Fold cross-validation.

- Split data into K small disjoint folds.
- Train on K-1 folds, evaluate the test error one the held-out fold.
- Repeat for all combinations and average the K estimates of the generalization error.

[24]:
```
from sklearn.cross_validation import KFold

scores = []

for train, test in KFold(n=len(X), n_folds=5, random_state=42):
    X_train, y_train = X[train], y[train]
    X_test, y_test = X[test], y[test]
    clf = KNeighborsClassifier().fit(X_train, y_train)
    scores.append(zero_one_loss(y_test, clf.predict(X_test)))

print("CV error = %f +-%f" % (np.mean(scores), np.std(scores)))
```

```
CV error = 0.163000 +-0.010770
```

[25]:
```
# Shortcut
from sklearn.cross_validation import cross_val_score
scores = cross_val_score(KNeighborsClassifier(), X, y,
                         cv=KFold(n=len(X), n_folds=5, random_state=42),
                         scoring="accuracy")
print("CV error = %f +-%f" % (1. - np.mean(scores), np.std(scores)))
```

```
CV error = 0.163000 +-0.010770
```

Metrics

Default score

Estimators come with a built-in default evaluation score

- Accuracy for classification
- R2 score for regression

[26]:
```
y_train = (y_train == "r")
y_test = (y_test == "r")
clf = KNeighborsClassifier(n_neighbors=5)
clf.fit(X_train, y_train)
print("Default score =", clf.score(X_test, y_test))
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=5, p=2,
           weights='uniform')


Default score = 0.84
```

Accuracy

Definition: The accuracy is the proportion of correct predictions.

```
[27]: from sklearn.metrics import accuracy_score
      print("Accuracy =", accuracy_score(y_test, clf.predict(X_test)))

Accuracy = 0.84
```

Precision, recall and F-measure

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

$$F = \frac{2 * Precision * Recall}{Precision + Recall}$$

```
[28]: from sklearn.metrics import precision_score
      from sklearn.metrics import recall_score
      from sklearn.metrics import fbeta_score
      print("Precision =", precision_score(y_test, clf.predict(X_test)))
      print("Recall =", recall_score(y_test, clf.predict(X_test)))
      print("F =", fbeta_score(y_test, clf.predict(X_test), beta=1))

Precision = 0.811881188119
Recall = 0.863157894737
F = 0.836734693878
```

ROC AUC

Definition: Area under the curve of the false positive rate (FPR) against the true positive rate (TPR) as the decision threshold of the classifier is varied.

```
[29]: from sklearn.metrics import get_scorer
      roc_auc_scorer = get_scorer("roc_auc")
      print("ROC AUC =", roc_auc_scorer(clf, X_test, y_test))

      from sklearn.metrics import roc_curve
      fpr, tpr, thresholds = roc_curve(y_test, clf.predict_proba(X_test)[:, 1])
      plt.plot(fpr, tpr)
      plt.xlabel("FPR")
      plt.ylabel("TPR")
      plt.show()
```