Performance Measures

Monday, March 2, 2020 4:15 PM

Note Written by: Zulfadli Zainal

Performance measure for classification is much more complex than regressor performance measures.

Measuring Accuracy Using Cross-Validation

A good way to evaluate a model is to use cross-validation, just as you did in Chapter 2.

Implementing Cross Validation: The following code roughly done the same thing as the preceding cross_val_score() code, and prints the same result:

Implementing Cross-Validation

Occasionally you will need more control over the cross-validation process than what `cross_val_score()` and similar functions provide. In these cases, you can implement cross-validation yourself; it is actually fairly straightforward. The following code does roughly the same thing as the preceding `cross_val_score()` code, and prints the same result:

```
from sklearn.model_selection import StratifiedKFold
from sklearn.base import clone

skfolds = StratifiedKFold(n_splits=3, random_state=42)

for train_index, test_index in skfolds.split(X_train, y_train_5):
    clone_clf = clone(sgd_clf)
    X_train_folds = X_train[train_index]
    y_train_folds = (y_train_5[train_index])
    X_test_fold = X_train[test_index]
    y_test_fold = (y_train_5[test_index])

    clone_clf.fit(X_train_folds, y_train_folds)
    y_pred = clone_clf.predict(X_test_fold)
    n_correct = sum(y_pred == y_test_fold)
    print(n_correct / len(y_pred))  # prints 0.9502, 0.96565 and 0.96495
```

The `StratifiedKFold` class performs stratified sampling (as explained in Chapter 2) to produce folds that contain a representative ratio of each class. At each iteration the code creates a clone of the classifier, trains that clone on the training folds, and makes predictions on the test fold. Then it counts the number of correct predictions and outputs the ratio of correct predictions.

Now, lets measure our SGD Classifier model accuracy using cross_val_score()

```
from sklearn.model_selection import cross_val_score
```

```
#Evaluate model using Cross Evaluation
print(cross_val_score(sgd_clf, X_train, y_train, cv=3, scoring='accuracy'))
```


```
from sklearn.model_selection import cross_val_score

#Evaluate model using Cross Evaluation
print(cross_val_score(sgd_clf, X_train, y_train, cv=3, scoring='accuracy'))
```


Result:

[0.87035 0.86345 0.85575]

The accuracy is around 85% - 87%

Lets try to compare the accuracy with more simple classifier - And test our number 9 sample accuracy.

```
from sklearn.base import BaseEstimator
```

```
# Try to measure the accuracy base on simple classifier model


class Never9Classifier(BaseEstimator):
    def fit(self, X, y=None):
        pass

    def predict(self, X):
        return np.zeros((len(X), 1), dtype=bool)


never_9_clf = Never9Classifier()
print(cross_val_score(never_9_clf, X_train, y_train_9, cv=3, scoring='accuracy'))
```


```
from sklearn.base import BaseEstimator

# Try to measure the accuracy base on simple classifier model

class Never9Classifier(BaseEstimator):
    def fit(self, X, y=None):
        pass
    def predict(self, X):
        return np.zeros((len(X), 1), dtype=bool)

never_9_clf = Never9Classifier()
print(cross_val_score(never_9_clf, X_train, y_train_9, cv=3, scoring='accuracy'))
```


Result:

[[0.90255 0.9011 0.8989]]

The accuracy is around 89% - 90%

The accuracy is different. This is because the number of datasets that containing 9 is small compared to the total datasets. This demonstrates why accuracy is generally not the preferred performance measure for classifiers, especially when you are dealing with unbalanced datasets (i.e., when some classes are much more frequent than others).

Confusion Matrix

Another way to evaluate performance of classifier -> Use confusion Matrix

Mechanism: The general idea is to count the number of times instances of class A are classified as class B.

To compute:

1. You need to have set of predictions
2. This need to be compared with actual targets
3. We can use cross validation function (cross_validation_predict) to apply this.

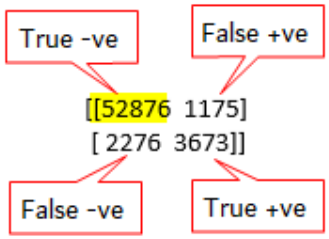
```
y_train_pred = cross_val_predict(sgd_clf, X_train, y_train_9, cv=3)

print(confusion_matrix(y_train_9, y_train_pred))
```


```
y_train_pred = cross_val_predict(sgd_clf, X_train, y_train_9, cv=3)
print(confusion_matrix(y_train_9, y_train_pred))
```


Result:

[[52876 1175]
[2276 3673]]



52876: Correctly classified as non 9
1175: Wrongly classified as 9
2276: Wrongly classified as non 9
3673: Correctly classified as 9

If perfect predictions, it will look like this:

[[54579, 0],
[0, 5421]]

Confusion Matrix -> Give you a lot of information (But complicated and confusing) -> Overcome this using Precision!

Equation 3-1. Precision

$$\text{precision} = \frac{TP}{TP + FP}$$

TP -> True Positive
FP -> False Positive

However if the number of sample is very small (Eg: 1/1), you can still get 100% precision. To overcome this issue, usually precision is being used together with Recall.

Equation 3-2. Recall

$$\text{recall} = \frac{TP}{TP + FN}$$

FN -> False Negative

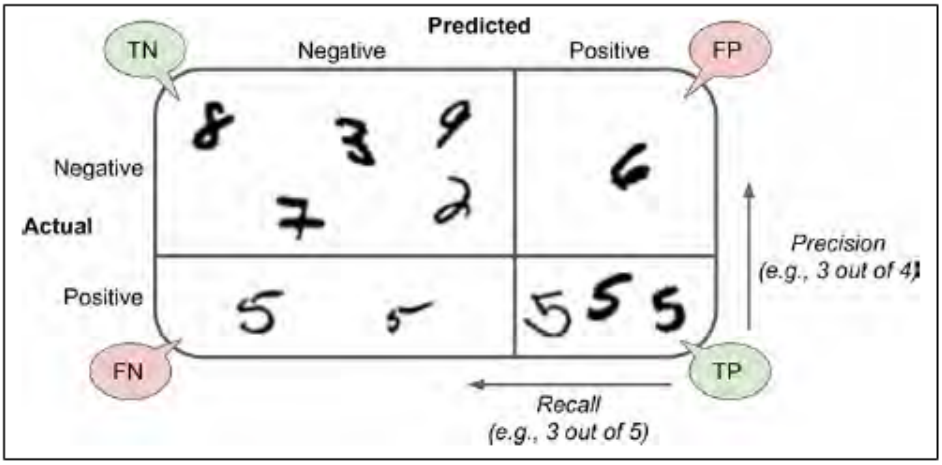


Figure 3-2. An illustrated confusion matrix