Multiclass Classification

Monday, March 23, 2020 12:22 PM

Note Written by: Zulfadli Zainal

Binary Classifiers: Distinguished between 2 classes.
Multiclass Classifier: Distinguished more than 2 classes.

Some algorithm able to handle multiclass directly, eg: Random Forest, Naïve Bayes
Others are strictly used for binary classifier, eg: SVM classifier, Linear classifier

However, there are **various strategies that you can use to perform multiclass classification using multiple binary classifiers.**

Strategy 1: One vs All Strategy/One versus The Rest Strategy (OvA)

For example, one way to create a system that can classify the digit images into 10 classes (from 0 to 9) is to train 10 binary classifiers, one for each digit (a 0-detector, a 1-detector, a 2-detector, and so on). Then when you want to classify an image, you get the decision score from each classifier for that image and you select the class whose classifier outputs the highest score. This is called the one-versus-all (OvA) strategy (also called one-versus-the-rest).

Strategy 2: One vs One Strategy (OvO)

Another strategy is to train a binary classifier for every pair of digits: one to distinguish 0s and 1s, another to distinguish 0s and 2s, another for 1s and 2s, and so on. This is called the one-versus-one (OvO) strategy. If there are N classes, you need to train $N \times (N - 1) / 2$ classifiers. For the MNIST problem, this means training 45 binary classifiers! When you want to classify an image, you have to run the image through all 45 classifiers and see which class wins the most duels. The main advantage of OvO is that each classifier only needs to be trained on the part of the training set for the two classes that it must distinguish.

Example:

If Scikit Learn detect you try to use binary classifier for multiclass classification task: It will automatically assume to use OvA strategy.

Predict with SGDClassifiers:

```
from sklearn.multiclass import OneVsOneClassifier
```

```
# Try to predict some digit with SGD Classifier

sgd_clf.fit(X_train, y_train)
print(sgd_clf.predict([some_digit]))

# Calculate scores for decision function

some_digit_scores = sgd_clf.decision_function([some_digit])
print(some_digit_scores)

# Find highest score

print(np.argmax(some_digit_scores))

# OVO strategy based on SGD Classifier

ovo_clf = OneVsOneClassifier(sgd_clf)
ovo_clf.fit(X_train, y_train)
print(ovo_clf.predict([some_digit]))
print(len(ovo_clf.estimators_))
```

```
from sklearn.multiclass import OneVsOneClassifier
```

Try to predict some digit with SGD Classifier

```
sgd_clf.fit(X_train, y_train)
print(sgd_clf.predict([some_digit]))

# Calculate scores for decision function

some_digit_scores = sgd_clf.decision_function([some_digit])
print(some_digit_scores)
```

[Result]

```
[[-36139.11209989 -11178.83438209 -15189.4708343   -1097.4968088
  -1871.53675105 -10339.23519239 -44572.14866888  -8840.06065921
  -2917.04470425  -4085.28264179]]
```

```
# Find highest score

print(np.argmax(some_digit_scores))
```

[Result]

3

Forcing ScikitLearn to use OvO startegy:

```
# OVO strategy based on SGD Classifier

ovo_clf = OneVsOneClassifier(sgd_clf)
ovo_clf.fit(X_train, y_train)
print(ovo_clf.predict([some_digit]))
print(len(ovo_clf.estimators_))
```

[Result]

Array [4]

45

Predict with Random Forest:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score
from sklearn.preprocessing import StandardScaler
```

```
# Training Random forest

forest_clf.fit(X_train, y_train)
print(forest_clf.predict([some_digit]))

# Predict probability

print(forest_clf.predict_proba([some_digit]))

# Calculate Cross Validation Score

print(cross_val_score(sgd_clf, X_train, y_train, cv=3, scoring='accuracy'))

# Scaling input to improve accuracy

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train.astype(np.float64))
print(cross_val_score(sgd_clf, X_train_scaled, y_train, cv=3, scoring='accuracy'))
```

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score
from sklearn.preprocessing import StandardScaler
```

```
# Training Random forest

forest_clf.fit(X_train, y_train)
print(forest_clf.predict([some_digit]))
```

[Result]
45

```
# Predict probability

print(forest_clf.predict_proba([some_digit]))
```

[Result]
[9]

```
# Calculate Cross Validation Score

print(cross_val_score(sgd_clf, X_train, y_train, cv=3, scoring='accuracy'))
```

[Result]
[[0. 0. 0. 0. 0.07 0. 0. 0. 0.01 0.92]]

```
# Scaling input to improve accuracy

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train.astype(np.float64))
print(cross_val_score(sgd_clf, X_train_scaled, y_train, cv=3, scoring='accuracy'))
```

[Result]
[0.87935 0.8784 0.86115]