Перекрестная проверка и поиск сетки для выбора модели в Python

Автор: Usman Malik < https://pythobyte.com/author/8950410092247140337/>

10.04.2021 < https://pythobyte.com/cross-validation-and-grid-search-for-model-selection-in-python-11883b5c/>

Автор оригинала: Usman Malik < https://stackabuse.com/cross-validation-and-grid-search-for-model-selection-in-python/> .

Перекрестная проверка и поиск сетки для выбора модели в Python

Вступление

Типичный процесс машинного обучения включает в себя обучение различных моделей на базе набора данных и выбор модели с наилучшей производительностью. Однако оценка производительности алгоритма не всегда является прямой задачей. Существует несколько факторов, которые могут помочь вам определить, какой алгоритм работает лучше всего. Одним из таких факторов является производительность набора перекрестной валидации, а другим – выбор параметров алгоритма.

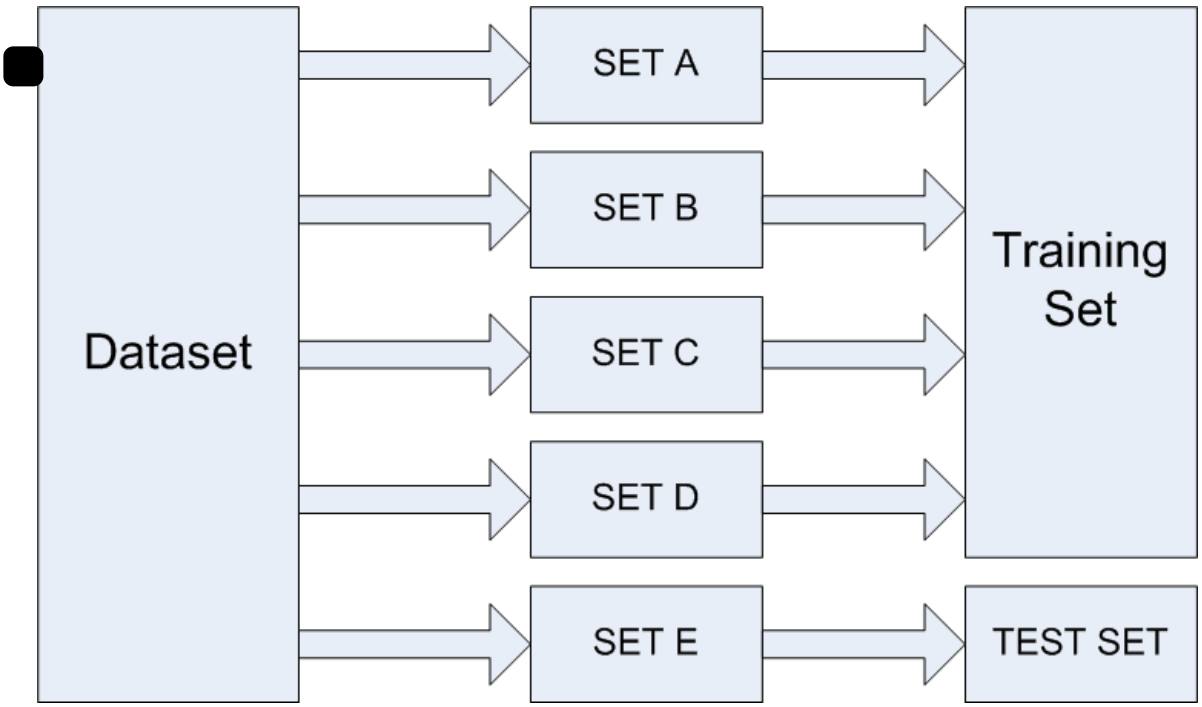
В этой статье мы подробно рассмотрим эти два фактора. Сначала мы изучим, что такое кросс-валидация, зачем она нужна и как ее выполнять с помощью библиотеки Python Scikit-Learn < http://scikit-learn.org/stable/> . Затем мы перейдем к алгоритму поиска сетки и посмотрим, как его можно использовать для автоматического выбора наилучших параметров для алгоритма. < https://en.wikipedia.org/wiki/Hyperparameter_optimization#Grid_search>

Перекрестная валидация

Обычно в процессе машинного обучения данные делятся на обучающие и тестовые наборы; затем обучающий набор используется для обучения модели, а тестовый набор-для оценки производительности модели. Однако такой подход может привести к проблемам дисперсии. Проще говоря, проблема дисперсии относится к сценарию, в котором наша точность, полученная на одном тесте, сильно отличается от точности, полученной на другом тестовом наборе с использованием того же алгоритма.

Решение этой проблемы заключается в использовании **K-кратной перекрестной проверки < https://en.wikipedia.org/wiki/Cross-validation_(statistics)#k-fold_cross-validation>** для оценки производительности, где K-любое число. Процесс K-кратной перекрестной проверки прост. Вы делите данные на K складок. Из K складок наборы K-1 используются для обучения, в то время как остальные наборы используются для тестирования. Алгоритм обучается и тестируется K раз, каждый раз новый набор используется в качестве тестового набора, а остальные наборы используются для обучения. Наконец, результатом K-кратной перекрестной проверки является среднее значение результатов, полученных на каждом наборе.

Предположим, мы хотим выполнить 5-кратную перекрестную проверку. Для этого данные делятся на 5 наборов, например, мы называем их НАБОРОМ A, НАБОРОМ B, НАБОРОМ C, НАБОРОМ D и НАБОРОМ E. Алгоритм обучается и тестируется K раз. В первом сгибе НАБОР A и НАБОР D используются в качестве обучающего набора, а НАБОР E используется в качестве тестового набора, как показано на рисунке ниже:



Во втором сгибе НАБОР A, НАБОР B, НАБОР C и НАБОР E используются для обучения, а НАБОР D используется в качестве тестирования. Процесс продолжается до тех пор, пока каждый набор не будет использован по крайней мере один раз для обучения и один раз для тестирования. Конечный результат-это среднее значение результатов, полученных с использованием всех сгибов. Таким образом, мы можем избавиться от дисперсии. Используя стандартное отклонение результатов, полученных из каждой складки, мы фактически можем найти дисперсию в общем результате.

Перекрестная проверка с помощью Scikit-Learn

В этом разделе мы будем использовать перекрестную проверку для оценки производительности алгоритма случайного леса для классификации. Проблема, которую мы собираемся решить, состоит в том, чтобы предсказать качество вина на основе 12 атрибутов. Подробная информация о наборе данных доступна по следующей ссылке:

В этом разделе мы будем использовать перекрестную проверку для оценки производительности алгоритма случайного леса для классификации. Проблема, которую мы собираемся решить, состоит в том, чтобы предсказать качество вина на основе 12 атрибутов.

Подробная информация о наборе данных доступна по следующей ссылке:

В этой статье мы используем только данные по красному вину.

Выполните следующие действия, чтобы реализовать перекрестную проверку с помощью Scikit-Learn:

1. Импорт Необходимых Библиотек

Следующий код импортирует несколько необходимых библиотек:

```
import pandas as pd
import numpy as np
```

2. Импорт набора данных

Загрузите набор данных, который доступен в Интернете по этой ссылке:

Загрузите набор данных, который доступен в Интернете по этой ссылке:

Как только мы загрузили его, мы поместили файл в папку “Datasets” нашего диска “D” ради этой статьи. Имя набора данных – “winequality-red.csv”. Обратите внимание, что вам нужно будет изменить путь к файлу, чтобы он соответствовал местоположению, в котором вы сохранили файл на своем компьютере.

Выполните следующую команду для импорта набора данных:

```
dataset = pd.read_csv(r"D:/Datasets/winequality-red.csv", sep
```

Набор данных был разделен точкой с запятой, поэтому мы передали атрибут “;” параметру “sep”, чтобы pandas мог правильно разобрать

файл.

3. Анализ Данных

Выполните следующий сценарий, чтобы получить обзор данных:

```
dataset.head()
```

Вывод выглядит следующим образом:

3.51	9.4	0.9978	5	0.076	0.56	0	0.00	7.4	1.9	0.70	11.0	34.0
3.20	9.8	0.9968	5	0.098	0.68	1	0.00	7.8	2.6	0.88	25.0	67.0
3.26	9.8	0.9970	5	0.092	0.65	2	0.04	7.8	2.3	0.76	15.0	54.0
3.16	9.8	0.9980	6	0.075	0.58	3	0.56	11.2	1.9	0.28	17.0	60.0
3.51	9.4	0.9978	5	0.076	0.56	4	0.00	7.4	1.9	0.70	11.0	34.0

4. Предварительная обработка данных

Выполните следующий сценарий для разделения данных на наборы меток и объектов.

```
X = dataset.iloc[:, 0:11].values
y = dataset.iloc[:, 11].values
```

Поскольку мы используем перекрестную проверку, нам не нужно делить наши данные на обучающие и тестовые наборы. Нам нужны

все данные в обучающем наборе, чтобы мы могли применить к ним перекрестную проверку. Самый простой способ сделать это- установить значение параметра `test_size` равным 0. Это вернет все данные в обучающем наборе следующим образом:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, tes
```

5. Масштабирование данных

Если вы посмотрите на набор данных, то заметите, что он плохо масштабируется. Например, столбцы “летучая кислотность” и “лимонная кислота” имеют значения от 0 до 1, в то время как большинство остальных столбцов имеют более высокие значения. Поэтому, прежде чем обучать алгоритм, нам нужно будет уменьшить масштаб наших данных.

Здесь мы будем использовать `Стандартный скалярный` класс.

```
from sklearn.preprocessing import StandardScaler
feature_scaler = StandardScaler()
X_train = feature_scaler.fit_transform(X_train)
X_test = feature_scaler.transform(X_test)
```

6. Обучение и перекрестная валидация

Первый шаг на этапе обучения и перекрестной проверки прост. Вам просто нужно импортировать класс алгоритма из библиотеки `sklearn` , как показано ниже:

```
from sklearn.ensemble import RandomForestClassifier
classifier = RandomForestClassifier(n_estimators=300, random_
```

Затем для реализации перекрестной проверки можно использовать метод `cross_val_score` библиотеки `sklearn.model_selection` . Функция `cross_val_score` возвращает точность для всех сгибов. Значения для 4 параметров должны быть переданы в класс `cross_val_score` . Первый параметр-это оценщик, который в основном определяет алгоритм, который вы хотите использовать для перекрестной проверки. Второй и третий параметры, `X` и `y` , содержат данные `X_train` и `y_train` , то есть объекты и метки. Наконец, количество сгибов передается параметру `cv` , как показано в следующем коде:

```
from sklearn.model_selection import cross_val_score
all_accuracies = cross_val_score(estimator=classifier, X=X_tr
```

После того как вы выполнили это, давайте просто напечатаем точность, возвращенную для пяти сгибов методом `cross_walk_score` , вызвав `print` on `all_accuracies` .

```
print(all_accuracies)
```

Выход:

```
[ 0.72360248  0.68535826  0.70716511  0.68553459  0.68454259
```

Чтобы найти среднее значение всех точностей, просто используйте метод `mean()` объекта, возвращаемого методом `cross_val_score` , как показано ниже:

```
print(all_accuracies.mean())
```

Среднее значение составляет 0,6972, или 69,72%.

Наконец, давайте найдем стандартное отклонение данных, чтобы увидеть степень дисперсии результатов, полученных нашей моделью. Для этого вызовите метод `std()` для объекта `all_accuracies` .

```
print(all_accuracies.std())
```

Результат: 0,01572, что составляет 1,57%. Это чрезвычайно мало, что означает, что наша модель имеет очень низкую дисперсию, что на самом деле очень хорошо, поскольку это означает, что предсказание, которое мы получили на одном тестовом наборе, не случайно. Скорее всего, модель будет работать более или менее одинаково на всех тестовых наборах.

Поиск по сетке для выбора параметров

Модель машинного обучения имеет два типа параметров. Первый тип параметров-это параметры, которые изучаются с помощью модели машинного обучения, в то время как второй тип параметров-это гиперпараметры, которые мы передаем в модель машинного обучения.

В последнем разделе при прогнозировании качества вина мы использовали алгоритм Случайного леса. Количество оценок, которые мы использовали для алгоритма, составило 300. Аналогично в алгоритме KNN мы должны указать значение K, а для алгоритма SVM мы должны указать тип ядра. Эти оценки – значение K и Ядро – являются всеми типами гиперпараметров.

Обычно мы случайным образом устанавливаем значение для этих гиперпараметров и видим, какие параметры приводят к лучшей производительности. Однако случайный выбор параметров для алгоритма может быть исчерпывающим.

Кроме того, нелегко сравнивать производительность различных алгоритмов путем случайной установки гиперпараметров, потому что один алгоритм может работать лучше другого с различным набором

параметров. И если параметры изменяются, алгоритм может работать хуже, чем другие алгоритмы.

Поэтому вместо случайного выбора значений параметров лучше было бы разработать алгоритм, который автоматически находит наилучшие параметры для конкретной модели. Одним из таких алгоритмов является поиск по сетке.

Поиск по сетке с помощью Scikit-Learn

Реализуем алгоритм поиска сетки на примере. Сценарий в этом разделе должен быть запущен после сценария, который мы создали в предыдущем разделе.

Для реализации алгоритма поиска сетки нам необходимо импортировать класс `GridSearchCV` из библиотеки `sklearn.model_selection` .

Первый шаг, который вам нужно выполнить, – это создать словарь всех параметров и соответствующий им набор значений, которые вы хотите проверить на лучшую производительность. Имя элементов словаря соответствует имени параметра, а значение-списку значений для этого параметра.

Давайте создадим словарь параметров и соответствующих им значений для нашего алгоритма случайного леса. Подробная информация обо всех параметрах алгоритма случайного леса доступна в **Scikit-Learn docs < http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html#sklearn.ensemble.RandomForestClassifier>** .

Для этого выполните следующий код:

```
grid_param = {
    'n_estimators': [100, 300, 500, 800, 1000],
    'criterion': ['gini', 'entropy'],
    'bootstrap': [True, False]
}
```

Внимательно посмотрите на приведенный выше код. Здесь мы создаем `grid_param` словарь с тремя параметрами `n_estimators` , `criterion` и `bootstrap` . Значения параметров, которые мы хотим опробовать, передаются в списке. Например, в приведенном выше сценарии мы хотим найти, какое значение (из 100, 300, 500, 800 и 1000) обеспечивает наибольшую точность.

Аналогично, мы хотим найти, какое значение приводит к наибольшей производительности для параметра `критерия` : “джини” или “энтропия”? Алгоритм поиска сетки в основном пробует все возможные комбинации значений параметров и возвращает комбинацию с наибольшей точностью. Например, в приведенном выше случае алгоритм проверит 20 комбинаций (5 x 2 x).

Алгоритм поиска сетки может быть очень медленным из-за потенциально огромного количества тестируемых комбинаций. Кроме того, перекрестная проверка еще больше увеличивает время выполнения и сложность.

После создания словаря параметров следующим шагом является создание экземпляра класса `GridSearchCV` . Вам нужно передать значения для параметра `estimator` , который в основном является алгоритмом, который вы хотите выполнить. Параметр `param_grid` принимает словарь параметров, который мы только что создали в качестве параметра, параметр `scoring` принимает показатели производительности, параметр `cv` соответствует количеству сгибов, которое в нашем случае равно 5, и, наконец, параметр `n_jobs` относится к числу процессоров, которые вы хотите использовать для выполнения. Значение -1 для параметра `n_jobs` означает, что используются все доступные вычислительные мощности. Это может быть удобно, если у вас есть большое количество данных.

Взгляните на следующий код:

```
gd_sr = GridSearchCV(estimator=classifier,
                     param_grid=grid_param,
                     scoring='accuracy',
```

```
                    cv=5,
                    n_jobs=-1)
```

После инициализации класса `GridSearchCV` последним шагом является вызов метода `fit` класса и передача ему обучающего и тестового набора, как показано в следующем коде:

```
gd_sr.fit(X_train, y_train)
```

Этот метод может занять некоторое время, потому что у нас есть 20 комбинаций параметров и 5-кратная перекрестная проверка. Поэтому алгоритм будет выполняться в общей сложности 100 раз.

Как только метод завершит выполнение, следующим шагом будет проверка параметров, возвращающих наибольшую точность. Для этого выведите атрибут `sr.best_params_` объекта `GridSearchCV` , как показано ниже:

```
best_parameters = gd_sr.best_params_
print(best_parameters)
```

Выход:

```
{'bootstrap': True, 'criterion': 'gini', 'n_estimators': 1000
```

Результат показывает, что наибольшая точность достигается, когда `n_estimators` равны 1000, `bootstrap` является `Истинным` и `критерием` является “джини”.

Примечание : Было бы неплохо добавить больше оценок и посмотреть, увеличится ли производительность еще больше, так как было выбрано

самое высокое допустимое значение `n_estimators` .

Последним и завершающим шагом алгоритма поиска сетки является поиск точности, полученной с использованием наилучших параметров. Ранее мы имели среднюю точность 69,72% с 300 `n_estimators` .

Чтобы найти наилучшую достигнутую точность, выполните следующий код:

```
best_result = gd_sr.best_score_
print(best_result)
```

Достигнутая точность составляет: 0,6985 из 69,85%, что лишь немного лучше, чем 69,72%. Чтобы улучшить это еще больше, было бы неплохо проверить значения других параметров алгоритма случайного леса, таких как `max_features` , `max_depth` , `max_leaf_nodes` и т. Д. чтобы увидеть, улучшится ли точность еще больше или нет.

Вывод

В этой статье мы изучили два очень часто используемых метода оценки производительности и выбора модели алгоритма. K-Кратная перекрестная проверка может быть использована для оценки производительности модели путем обработки дисперсионной задачи результирующего набора. Кроме того, для определения наилучшего алгоритма и наилучших параметров мы можем использовать алгоритм поиска сетки.