Деревья решений в Python с Scikit-Learn

Автор: Scott Robinson < https://pythobyte.com/author/scott_robinson/>

07.04.2021 < https://pythobyte.com/decision-trees-in-python-with-scikit-learn-9e8b0826/>

Автор оригинала: Scott Robinson < https://stackabuse.com/decision-trees-in-python-with-scikit-learn/> .

Деревья решений в Python с Scikit-Learn

Вступление

Дерево решений – это один из наиболее часто и широко используемых алгоритмов контролируемого машинного обучения, который может выполнять как регрессионные, так и классификационные задачи. Интуиция, лежащая в основе алгоритма дерева решений, проста, но в то же время очень мощна.

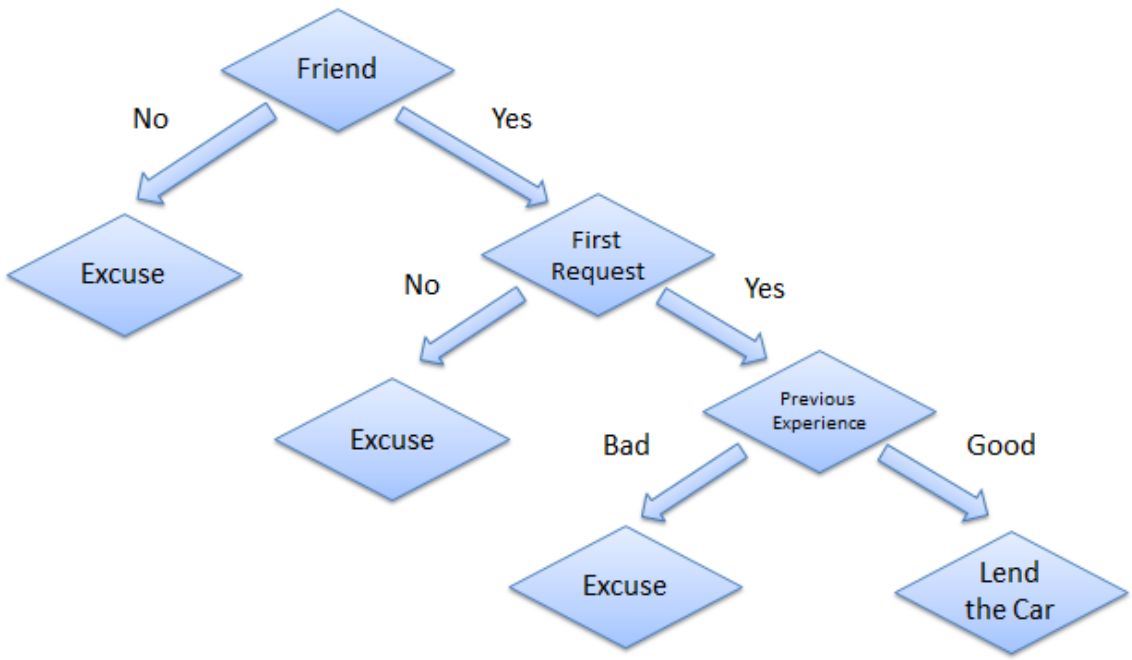
Для каждого атрибута в наборе данных алгоритм decision tree < https://en.wikipedia.org/wiki/Decision_tree_learning> формирует узел, где наиболее важный атрибут помещается в корневой узел. Для оценки мы начинаем с корневого узла и продвигаемся вниз по дереву, следуя за соответствующим узлом, который удовлетворяет нашему условию или “решению”. Этот процесс продолжается до тех пор, пока не будет достигнут конечный узел, содержащий предсказание или результат дерева решений.

Поначалу это может показаться немного сложным, но вы, вероятно, не осознаете, что всю свою жизнь использовали деревья решений для принятия решений, даже не подозревая об этом. Представьте себе ситуацию, когда человек просит вас одолжить ему машину на день, и вы должны принять решение, одолжить ему машину или нет. Есть

несколько факторов, которые помогают определить ваше решение, некоторые из которых были перечислены ниже:

1. Является ли этот человек близким другом или просто знакомым? Если человек просто знакомый, то отклоните просьбу; если человек друг, то переходите к следующему шагу.
2. Человек просит машину в первый раз? Если это так, одолжите им машину, в противном случае переходите к следующему шагу.
3. Была ли машина повреждена в прошлый раз, когда они возвращали машину? Если да, отклоните просьбу; если нет, одолжите им машину.

Дерево решений для вышеупомянутого сценария выглядит следующим образом:



Преимущества деревьев решений

Использование деревьев решений для прогнозного анализа имеет ряд преимуществ:

1. Деревья решений могут быть использованы для прогнозирования как непрерывных, так и дискретных значений, т. е. они хорошо работают как для задач регрессии, так и для задач классификации.
2. Они требуют относительно меньших усилий для обучения алгоритма.
3. Они могут быть использованы для классификации нелинейно разделимых данных.
4. Они очень быстры и эффективны по сравнению с KNN и другими алгоритмами классификации.

Реализация деревьев решений с помощью Python Scikit Learn

В этом разделе мы реализуем алгоритм дерева решений с помощью библиотеки Python **Scikit-Learn < http://scikit-learn.org/stable/>** . В следующих примерах мы будем решать как классификационные, так и регрессионные задачи с использованием дерева решений.

Примечание : Задачи классификации и регрессии были выполнены в записной книжке Jupyter IPython.

1. Дерево решений для классификации

В этом разделе мы будем предсказывать, является ли банкнота подлинной или поддельной в зависимости от четырех различных атрибутов изображения банкноты. К атрибутам относятся дисперсия вейвлет-преобразованного изображения, эксцесс изображения, энтропия и асимметрия изображения.

Набор данных

Набор данных для этой задачи можно загрузить по этой ссылке:

Набор данных для этой задачи можно загрузить по этой ссылке:

Для получения более подробной информации об этом наборе данных ознакомьтесь с **UCI ML repo < https://archive.ics.uci.edu/ml/datasets/banknote+authentication>** для этого набора данных.

Остальные шаги по реализации этого алгоритма в Scikit-Learn идентичны любой типичной задаче машинного обучения: мы импортируем библиотеки и наборы данных, выполняем некоторый анализ данных, разделяем данные на обучающие и тестовые наборы, обучаем алгоритм, делаем прогнозы и, наконец, оцениваем производительность алгоритма на нашем наборе данных.

Импорт библиотек

Следующий сценарий импортирует необходимые библиотеки:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

Импорт набора данных

Импорт набора данных

```
dataset = pd.read_csv("D:/Datasets/bill_authentication.csv")
```

Импорт набора данных

Анализ данных

Выполните следующую команду, чтобы увидеть количество строк и столбцов в нашем наборе данных:

```
dataset.shape
```

Результат покажет “(1372,5)”, что означает, что наш набор данных имеет 1372 записи и 5 атрибутов.

Выполните следующую команду для проверки первых пяти записей набора данных:

```
dataset.head()
```

Результат будет выглядеть следующим образом:

0	-0.44699	-2.8073	8.6661	3.62160	0
0	-1.46210	-2.4586	8.1674	4.54590	1
0	0.10645	1.9242	-2.6383	3.86600	2
0	-3.59440	-4.0112	9.5228	3.45660	3
0	-0.98880	4.5718	-4.4552	0.32924	4

Подготовка данных

В этом разделе мы разделим наши данные на атрибуты и метки, а затем разделим полученные данные на обучающие и тестовые наборы. Таким образом, мы можем обучить наш алгоритм на одном наборе данных, а затем протестировать его на совершенно другом наборе данных, который алгоритм еще не видел. Это дает вам более точное представление о том, как на самом деле будет работать ваш обученный алгоритм.

Чтобы разделить данные на атрибуты и метки, выполните следующий код:

```
X = dataset.drop('Class', axis=1)
y = dataset['Class']
```

Здесь переменная `X` содержит все столбцы из набора данных, кроме столбца “Класс”, который является меткой. Переменная `y` содержит значения из столбца “Класс”. Переменная `X` – это наш набор атрибутов, а переменная `y` содержит соответствующие метки.

Заключительный этап предварительной обработки состоит в том, чтобы разделить наши данные на обучающие и тестовые наборы. Библиотека `model_selection` Scikit-Learn содержит метод `train_test_split` , который мы будем использовать для случайного разделения данных на обучающие и тестовые наборы. Для этого выполните следующий код:

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

В приведенном выше коде параметр `test_size` указывает отношение тестового набора, которое мы используем для разделения 20% данных в тестовом наборе и 80% для обучения.

Обучение и составление прогнозов

После того, как данные были разделены на обучающие и тестовые наборы, последний шаг состоит в том, чтобы обучить алгоритм дерева решений на этих данных и сделать прогнозы. Scikit-Learn содержит библиотеку `tree` , которая содержит встроенные классы/методы для различных алгоритмов дерева решений. Поскольку мы собираемся выполнить здесь задачу классификации, мы будем использовать класс `DecisionTreeClassifier` для этого примера. Метод `fit` этого класса вызывается для обучения алгоритма на обучающих данных, которые передаются в качестве параметра методу `fit` . Выполните следующий сценарий для обучения алгоритма:

```
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier()
classifier.fit(X_train, y_train)
```

Теперь, когда наш классификатор обучен, давайте сделаем прогнозы по тестовым данным. Для составления прогнозов используется метод `predict` класса `Decision Tree Classifier|/. Взгляните на следующий код для использования:`

```
y_pred = classifier.predict(X_test)
```

Оценка алгоритма

На данный момент мы обучили наш алгоритм и сделали некоторые прогнозы. Теперь посмотрим, насколько точен наш алгоритм. Для задач классификации обычно используются такие метрики, как **матрица путаницы < https://en.wikipedia.org/wiki/Confusion_matrix>** , точность, отзыв и **оценка F1 < https://en.wikipedia.org/wiki/F1_score>** . К счастью для нас, библиотека Scikit=-Learn `metrics` содержит методы `classification_report` и `confusion_matrix` , которые могут быть использованы для расчета этих метрик для нас:

```
from sklearn.metrics import classification_report, confusion_
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

Это приведет к следующей оценке:

```
[[142    2]
    2  129]]
              precision    recall  f1-score   support

           0       0.99      0.99      0.99       144
           1       0.98      0.98      0.98       131
```

```
avg / total       0.99      0.99      0.99       275
```

Из матрицы путаницы вы можете видеть, что из 275 тестовых экземпляров наш алгоритм неправильно классифицировал только 4. Это 98,5 % точности. Не так уж и плохо!

2. Дерево решений для регрессии

Процесс решения регрессионной задачи с деревом решений с помощью Scikit Learn очень похож на процесс классификации. Однако для регрессии мы используем класс DecisionTreeRegressor древовидной библиотеки. Кроме того, оценочные показатели регрессии отличаются от показателей классификации. В остальном процесс почти такой же.

Набор данных

Набор данных, который мы будем использовать для этого раздела, такой же, как и в статье о линейной регрессии. Мы будем использовать этот набор данных, чтобы попытаться предсказать потребление газа (в миллионах галлонов) в 48 штатах США на основе налога на газ (в центах), дохода на душу населения (в долларах), мощеных автомагистралей (в милях) и доли населения с водительскими правами.

Набор данных доступен по этой ссылке:

Набор данных доступен по этой ссылке:

Подробности набора данных можно найти в **оригинальном источнике < http://people.sc.fsu.edu/~jburkardt/datasets/regression/x16.txt>** .

Первые два столбца в приведенном выше наборе данных не содержат никакой полезной информации, поэтому они были удалены из файла набора данных.

Теперь давайте применим наш алгоритм дерева решений к этим данным, чтобы попытаться предсказать потребление газа на основе этих данных.

Импорт библиотек

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

Импорт набора данных

```
dataset = pd.read_csv('D:\Datasets\petrol_consumption.csv')
```

Анализ данных

Мы снова будем использовать функцию `head` фрейма данных, чтобы увидеть, как на самом деле выглядят наши данные:

```
dataset.head()
```

Вывод выглядит следующим образом:

9.0	0	3571	1976	541	0.525
9.0	1	4092	1250	524	0.572
9.0	2	3865	1586	561	0.580
7.5	3	4870	2351	414	0.529
8.0	4	4399	431	410	0.544

Вывод выглядит следующим образом:

```
dataset.describe()
```

48.000000	Вывод выглядит следующим образом:	48.000000	48.000000	48.000000	48.000000
7.668333	Вывод выглядит следующим образом:	4241.833333	5565.416667	576.770833	0.570333
0.950770	станд	573.623768	3491.507166	111.885816	0.055470
5.000000	минута	3063.000000	431.000000	344.000000	0.451000
7.000000	25%	3739.000000	3110.250000	509.500000	0.529750
7.500000	50%	4298.000000	4735.500000	568.500000	0.564500
8.125000	75%	4578.750000	7156.000000	632.750000	0.595250
10.000000	максимум	5342.000000	17782.000000	986.000000	0.724000

максимум

максимум

максимум

```
X = dataset.drop('Petrol_Consumption', axis=1)
y = dataset['Petrol_Consumption']
```

Здесь переменная `X` содержит все столбцы из набора данных, кроме столбца ‘Petrol_Consumption’, который является меткой. Переменная `y` содержит значения из столбца ‘Petrol_Consumption’, что означает, что переменная `X` содержит набор атрибутов, а переменная `y` содержит соответствующие метки.

Здесь переменная || X || содержит все столбцы из набора данных, кроме столбца ‘Petrol_Consumption’, который является меткой. Переменная || y || содержит значения из столбца ‘Petrol_Consumption’, что означает, что переменная || X || содержит набор атрибутов, а переменная || y || содержит соответствующие метки.

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

Здесь переменная || X || содержит все столбцы из набора данных, кроме столбца ‘Petrol_Consumption’, который является меткой. Переменная || y || содержит значения из столбца ‘Petrol_Consumption’, что означает, что переменная || X || содержит набор атрибутов, а переменная || y || содержит соответствующие метки.

Здесь переменная `X` содержит все столбцы из набора данных, кроме столбца ‘Petrol_Consumption’, который является меткой. Переменная `y` содержит значения из столбца ‘Petrol_Consumption’, что означает, что переменная

Здесь переменная `X` содержит все столбцы из набора данных, кроме столбца ‘Petrol_Consumption’, который является меткой. Переменная `y` содержит значения из столбца ‘Petrol_Consumption’, что означает, что переменная

```
from sklearn.tree import DecisionTreeRegressor
regressor = DecisionTreeRegressor()
regressor.fit(X_train, y_train)
```

Здесь переменная `X` содержит все столбцы из набора данных, кроме столбца ‘Petrol_Consumption’, который является меткой. Переменная

```
y_pred = regressor.predict(X_test)
```

Теперь давайте сравним некоторые из наших прогнозируемых значений с фактическими значениями и

```
df=pd.DataFrame({'Actual':y_test, 'Predicted':y_pred})
df
```

Теперь давайте сравним некоторые из наших прогнозируемых значений с фактическими значениями и

699	631.0	41
561	524.0	2
525	510.0	12
640	704.0	36
648	524.0	38
498	510.0	9
460	510.0	24
508	603.0	13
644	631.0	35

Теперь давайте сравним некоторые из наших прогнозируемых значений с фактическими значениями и

Теперь давайте сравним некоторые из наших прогнозируемых значений с фактическими значениями и

Для оценки эффективности регрессионного алгоритма обычно используются следующие метрики: **средняя абсолютная ошибка < https://en.wikipedia.org/wiki/Mean_absolute_error>** , **среднеквадратичная ошибка < https://en.wikipedia.org/wiki/Mean_squared_error>** и **< https://en.wikipedia.org/wiki/Root-mean-square_deviation>**

```
from sklearn import metrics
print('Mean Absolute Error:', metrics.mean_absolute_error(y_t
print('Mean Squared Error:', metrics.mean_squared_error(y_tes
print('Root Mean Squared Error:', np.sqrt(metrics.mean_square
```

Для оценки эффективности регрессионного алгоритма обычно используются следующие метрики: || средняя абсолютная ошибка||, || среднеквадратичная ошибка || и ||

```
Mean Absolute Error: 54.7
Mean Squared Error: 4228.9
Root Mean Squared Error: 65.0299930801
```

Средняя абсолютная ошибка для нашего алгоритма составляет 54,7, что составляет менее 10 процентов от среднего значения всех значений в столбце “Petrol_Consumption”.

Средняя абсолютная ошибка для нашего алгоритма составляет 54,7, что составляет менее 10 процентов от среднего значения всех значений в столбце “Petrol_Consumption”.

Хотите узнать больше о Scikit-Learn и других полезных алгоритмах машинного обучения? Я бы рекомендовал проверить некоторые более подробные ресурсы, такие как онлайн-курс:

Вывод

В этой статье мы показали, как можно использовать популярную библиотеку Scikit-Learn Python для использования деревьев решений как для задач классификации, так и для задач регрессии. Будучи довольно простым алгоритмом сам по себе, реализация деревьев решений с помощью Scikit-Learn еще проще.

Читайте Ещё По Теме:

- **Алгоритм Витерби: Реализация на Python < https://pythobyte.com/viterbi-algorithm-python-70164/>**
- **Руководство инсайдера по алгоритму A* в Python < https://pythobyte.com/a-star-algorithm-python-93427/>**
- **TimSort: Алгоритм и реализация в Python < https://pythobyte.com/python-timsort-34500/>**
- **Битоническая сортировка: Алгоритм и реализация в Python < https://pythobyte.com/bitonic-sort-python-28918/>**
- **Двоичный алгоритм поиска в Python < https://pythobyte.com/daily-python-puzzle-bsearch-631791bc/>**

- Quicksort в Python < https://pythobyte.com/quicksort-in-python-333e847a/>
- Алгоритм K-ближайших соседей в Python и Scikit-Learn < https://pythobyte.com/k-nearest-neighbors-algorithm-in-python-and-scikit-learn-31cdde4d/>

Вверх ↑