Классификаторы градиентного бустинга в Python с помощью Scikit-Learn

Автор: Dan Nelson < https://pythobyte.com/author/30389147019092124 77/>

14.04.2021 < https://pythobyte.com/gradient-boosting-classifiers-in-python-with-scikit-learn-6637b9fe/>

Автор оригинала: Dan Nelson < https://stackabuse.com/gradient-boosting-classifiers-in-python-with-scikit-learn/ >.

Вступление

Классификаторы градиентного повышения < https://en.wikipedia.org/wiki/Gradient_boosting> представляют собой группу алгоритмов машинного обучения, которые объединяют множество слабых моделей обучения вместе для создания сильной прогностической модели. Деревья решений обычно используются при выполнении градиентного бустинга. Модели градиентного бустинга становятся популярными из-за их эффективности при классификации сложных наборов данных и в последнее время используются для победы во многих конкурсах Kaggle < https://www.kaggle.com/> data science.

Библиотека машинного обучения Python, Scikit-Learn < https://scikit-learn.org/stable/> , поддерживает различные реализации классификаторов градиентного бустинга, включая XGBoost < https://xgboost.readthedocs.io/en/latest/> .

В этой статье мы рассмотрим теорию, лежащую в основе моделей/ классификаторов градиентного бустинга, и рассмотрим два различных способа проведения классификации с помощью классификаторов градиентного бустинга в Scikit-Learn.

Определение Терминов

Давайте начнем с определения некоторых терминов, связанных с классификаторами машинного обучения и градиентного повышения.

Прежде всего, что такое классификация? В машинном обучении существует два типа задач **контролируемого обучения < https://www.geeksforgeeks.org/regression-classification-supervised-machine-learning/>** : *классификация* и *регрессия* .

Классификация относится к задаче придания алгоритму машинного обучения признаков и того, чтобы алгоритм поместил экземпляры/ точки данных в один из многих *дискретных* классов. Классы категоричны по своей природе, экземпляр не может быть классифицирован как частично один класс и частично другой. Классическим примером задачи классификации является классификация электронных писем как “спам” или “не спам” – нет никакой “немного спамовой” электронной почты.

Регрессии выполняются, когда выход модели машинного обучения является реальным значением или непрерывным значением. Таким примером таких непрерывных значений будет “вес” или “длина”. Примером регрессионной задачи является прогнозирование возраста человека на основе таких характеристик, как рост, вес, доход и т. Д.

Классификаторы градиентного бустинга – это специфические типы алгоритмов, которые используются для задач классификации, как следует из названия.

Функции – это входные данные, которые задаются алгоритму машинного обучения, входные данные, которые будут использоваться для вычисления выходного значения. В математическом смысле объекты набора данных являются переменными, используемыми для решения уравнения. Другая часть уравнения-это *label* или target, которые являются классами, в которые будут классифицированы экземпляры. Поскольку метки содержат целевые значения для классификатора машинного обучения, при обучении классификатора следует разделить данные на обучающие и тестовые наборы. Обучающий набор будет иметь цели/метки, в то время как тестовый набор не будет содержать этих значений.

Scikit-Learn, или “sklearn”, – это библиотека машинного обучения, созданная для Python, предназначенная для ускорения задач машинного обучения за счет упрощения реализации алгоритмов машинного обучения. Он имеет простые в использовании функции, помогающие разбивать данные на обучающие и тестовые наборы, а также обучать модель, делать прогнозы и оценивать ее.

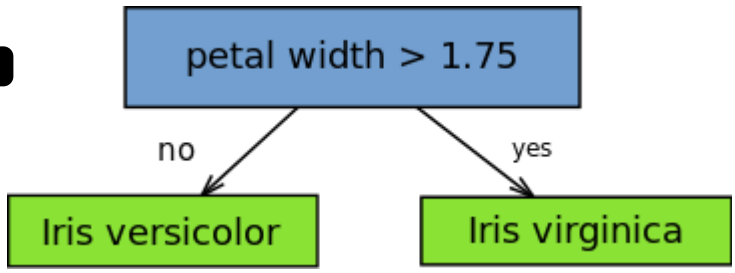
Как появился Градиентный Бустинг

Идея “градиентного бустинга” состоит в том, чтобы взять слабую гипотезу или слабый алгоритм обучения и внести в него ряд изменений, которые улучшат силу гипотезы/ученика. Этот тип повышения гипотезы основан на идее **Вероятности Приблизительно правильного обучения < https://en.wikipedia.org/wiki/Probably_approximately_correct_learning>** (PAC).

Этот метод обучения PAC исследует проблемы машинного обучения, чтобы интерпретировать, насколько они сложны, и аналогичный метод применяется к *Повышение гипотезы* .

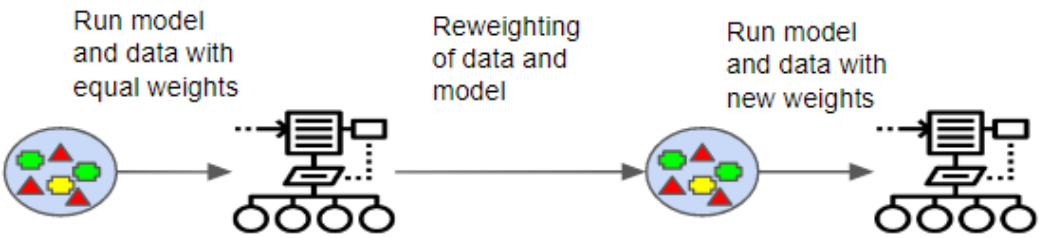
При повышении гипотезы вы просматриваете все наблюдения, на которых обучается алгоритм машинного обучения, и оставляете только те наблюдения, которые метод машинного обучения успешно классифицировал, вычеркивая другие наблюдения. Новый слабый ученик создается и тестируется на наборе данных, которые были плохо классифицированы, а затем сохраняются только примеры, которые были успешно классифицированы.

Эта идея была реализована в алгоритме **Adaptive Boosting < https://towardsdatascience.com/boosting-algorithm-adaboost-b6737a9ee60c>** (*AdaBoost*). Для AdaBoost многие слабые ученики создаются путем инициализации многих алгоритмов дерева решений, которые имеют только одно разделение, например “пень” на изображении ниже.



Экземпляры/наблюдения в обучающем наборе взвешиваются алгоритмом, и больший вес присваивается экземплярам, которые трудно классифицировать. Более слабые ученики добавляются в систему последовательно, и они назначаются на самые сложные учебные экземпляры.

В АдаБусте предсказания делаются большинством голосов, причем примеры классифицируются в соответствии с тем, какой класс получает наибольшее количество голосов от слабых учеников.



Классификаторы градиентного бустинга-это метод АдаБустинга в сочетании с взвешенной минимизацией, после чего классификаторы и взвешенные входные данные пересчитываются. Цель классификаторов градиентного бустинга состоит в том, чтобы минимизировать потери или разницу между фактическим значением класса обучающего примера и прогнозируемым значением класса. Не требуется понимать процесс уменьшения потерь классификаторов, но он работает аналогично **градиентному спуску < https://en.wikipedia.org/wiki/Gradient_descent>** в нейронной сети.

Были внесены уточнения в этот процесс и созданы **Градиентные форсирующие машины < https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3885826/>** .

В случае машин с градиентным повышением каждый раз, когда в модель добавляется новый слабый ученик, веса предыдущих учеников замораживаются или цементируются на месте, оставаясь неизменными по мере введения новых слоев. Это отличается от подходов, используемых в AdaBoosting, где значения корректируются при добавлении новых учеников.

Мощь машин градиентного бустинга проистекает из того факта, что они могут быть использованы не только для задач бинарной классификации, но и для задач многоклассовой классификации и даже регрессионных задач.

Теория Градиентного Усиления

Классификатор градиентного повышения зависит от **функции потерь < https://en.wikipedia.org/wiki/Loss_function>** . Можно использовать пользовательскую функцию потерь, и многие стандартизированные функции потерь поддерживаются классификаторами градиентного повышения, но функция потерь должна быть дифференцируемой.

Алгоритмы классификации часто используют логарифмические потери, в то время как алгоритмы регрессии могут использовать квадраты ошибок. Системы градиентного бустинга не должны выводить новую функцию потерь каждый раз, когда добавляется алгоритм бустинга, скорее любая дифференцируемая функция потерь может быть применена к системе.

Системы градиентного бустинга имеют еще две необходимые части: слабый обучаемый и аддитивный компонент. Системы градиентного бустинга используют деревья решений в качестве своих слабых учеников. Деревья регрессии используются для слабых учеников, и эти деревья регрессии выводят реальные значения. Поскольку выходные данные являются реальными значениями, по мере добавления новых учащихся в модель выходные данные деревьев регрессии могут быть добавлены вместе для исправления ошибок в прогнозах.

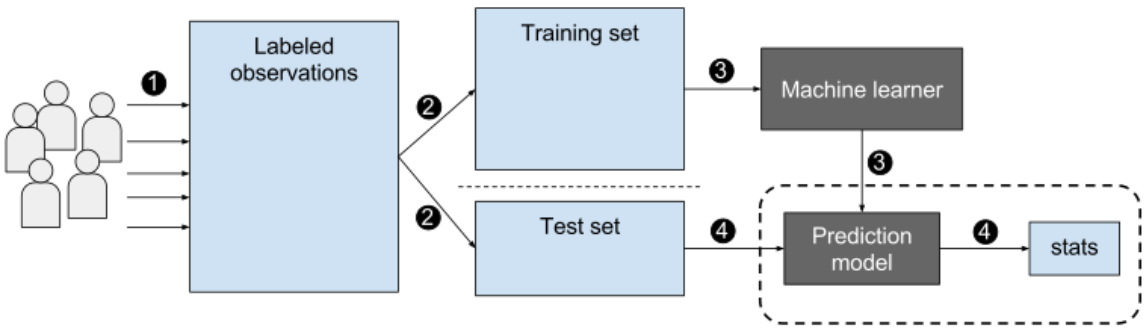
Аддитивная составляющая модели градиентного бустинга исходит из того, что деревья добавляются в модель с течением времени, и когда

это происходит, существующие деревья не манипулируются, их значения остаются фиксированными.

Процедура, аналогичная градиентному спуску, используется для минимизации ошибки между заданными параметрами. Это делается путем взятия расчетной потери и выполнения градиентного спуска, чтобы уменьшить эту потерю. После этого параметры дерева модифицируются, чтобы уменьшить остаточные потери.

Затем выходные данные нового дерева добавляются к выходным данным предыдущих деревьев, используемых в модели. Этот процесс повторяется до тех пор, пока не будет достигнуто заранее заданное количество деревьев или потери не будут уменьшены ниже определенного порога.

Шаги к градиентному бустингу



Чтобы реализовать классификатор градиентного бустинга, нам потребуется выполнить ряд различных шагов. Нам нужно будет:

- Подогнать под модель
- Настройка параметров модели и гиперпараметров
- Делайте прогнозы
- Интерпретируйте результаты

Подгонка моделей с помощью Scikit-Learn довольно проста, так как обычно нам просто нужно вызвать команду `fit()` после настройки модели.

Однако настройка гиперпараметров модели требует от нас активного принятия решений. Существуют различные аргументы/ гиперпараметры, которые мы можем настроить, чтобы попытаться получить наилучшую точность для модели. Один из способов сделать это-изменить скорость обучения модели. Мы хотим проверить производительность модели на обучающем наборе при различных скоростях обучения, а затем использовать наилучшую скорость обучения для прогнозирования.

Предсказания могут быть сделаны в Scikit-Learn очень просто с помощью функции `predict()` после подгонки классификатора. Вы захотите спрогнозировать характеристики тестового набора данных, а затем сравнить прогнозы с фактическими метками. Процесс оценки классификатора обычно включает в себя проверку точности классификатора, а затем настройку параметров/гиперпараметров модели до тех пор, пока классификатор не достигнет точности, удовлетворяющей пользователя.

Различные Улучшенные Классификаторы Градиентного Бустинга

Из-за того, что алгоритмы повышения оценки могут легко переоснащаться на наборе обучающих данных, различные ограничения или **методы регуляризации < https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html/>** могут быть использованы для повышения производительности алгоритма и борьбы с переоснащением. Наказанное обучение, ограничения дерева, рандомизированная выборка и усадка могут быть использованы для борьбы с переобучением.

Наказанное Обучение

Некоторые ограничения могут быть использованы для предотвращения переоснащения, в зависимости от структуры дерева решений. Тип дерева решений, используемого в градиентном бустинге, – это дерево регрессии, которое имеет числовые значения в виде листьев или весов. Эти значения веса могут быть регуляризованы с помощью различных методов регуляризации, таких как веса

регуляризации L1 или L2, которые штрафуют алгоритм радиантного повышения.

Ограничения дерева

Дерево решений может быть ограничено множеством способов, таких как ограничение глубины дерева, наложение ограничения на количество листьев или узлов дерева, ограничение количества наблюдений на разбиение и ограничение количества наблюдений, обученных. В общем, чем больше ограничений вы используете при создании деревьев, тем больше деревьев потребуется модели для правильного соответствия данным.

Случайная выборка/Стохастический Бустинг

Взятие случайных подвыборок обучающего набора данных, метод, называемый стохастическим градиентным повышением, также может помочь предотвратить переобучение. Этот метод существенно снижает силу корреляции между деревьями.

Существует несколько способов субсэмплирования набора данных, таких как субсэмплирование столбцов перед каждым разделением, субсэмплирование столбцов перед созданием дерева, а также субсэмплирование строк перед созданием дерева. В целом, субсэмплирование при больших скоростях, не превышающих 50% данных, кажется полезным для модели.

Усадка/Взвешенные Обновления

Поскольку предсказания каждого дерева суммируются вместе, вклад деревьев может быть подавлен или замедлен с помощью метода, называемого усадкой. “Скорость обучения” корректируется, и когда скорость обучения уменьшается, в модель необходимо добавить больше деревьев. Это делает его таким, что модель нуждается в более длительном обучении.

Существует компромисс между скоростью обучения и количеством необходимых деревьев, поэтому вам придется поэкспериментировать, чтобы найти лучшие значения для каждого из параметров, но

небольшие значения меньше 0,1 или значения от 0,1 до 0,3 часто работают хорошо.

XGBoost

XGBoost-это усовершенствованная и настроенная версия системы дерева решений с градиентным повышением, созданная с учетом производительности и скорости. XGBoost на самом деле означает “Экстремальный градиентный бустинг”, и это относится к тому факту, что алгоритмы и методы были настроены так, чтобы раздвинуть предел возможного для алгоритмов градиентного бустинга.

В следующем разделе мы сравним обычный повышающий классификатор и классификатор XGBoost.

Реализация Классификатора Градиентного Бустинга

Теперь мы рассмотрим реализацию простого классификатора градиентного бустинга и классификатора XGBoost. Начнем с простого повышающего классификатора.

Регулярный повышающий классификатор

Для начала нам нужно выбрать набор данных для работы, и в этом примере мы будем использовать набор данных Titanic. Вы можете скачать данные **здесь < https://www.kaggle.com/c/titanic/data>** .

Давайте начнем с импорта всех наших библиотек:

```
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_
from sklearn.ensemble import GradientBoostingClassifier
```

Теперь давайте загрузим наши тренировочные данные:

```
train_data = pd.read_csv("train.csv")
test_data = pd.read_csv("test.csv")
```

Возможно, нам потребуется выполнить некоторую предварительную обработку данных. Давайте установим индекс как `Идентификатор пассажира` , а затем выберем ваши функции и метки. Наши данные метки, данные `y` – это столбец `Survived` . Поэтому мы сделаем этот собственный фрейм данных, а затем удалим его из функций:

```
y_train = train_data["Survived"]
train_data.drop(labels="Survived", axis=1, inplace=True)
```

Теперь нам нужно создать объединенный новый набор данных:

```
full_data = train_data.append(test_data)
```

Давайте отбросим любые столбцы, которые не нужны или полезны для обучения, хотя вы можете оставить их и посмотреть, как они влияют на вещи:

```
drop_columns = ["Name", "Age", "SibSp", "Ticket", "Cabin", "
full_data.drop(labels=drop_columns, axis=1, inplace=True)
```

Любые текстовые данные должны быть преобразованы в числа, которые может использовать наша модель, поэтому давайте изменим это сейчас. Мы также заполним все пустые ячейки 0:

```
full_data = pd.get_dummies(full_data, columns=["Sex"])
full_data.fillna(value=0.0, inplace=True)
```

Давайте разделим данные на обучающие и тестовые наборы:

```
X_train = full_data.values[0:891]
X_test = full_data.values[891:]
```

Теперь мы будем масштабировать наши данные, создавая экземпляр масштаба и масштабируя его:

```
scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Теперь мы можем разделить данные на обучающие и тестовые наборы. Давайте также установим семя (чтобы вы могли реплицировать результаты) и выберем процент данных для тестирования.:

```
state = 12
test_size = 0.30

X_train, X_val, y_train, y_val = train_test_split(X_train, y_
    test_size=test_size, random_state=state)
```

Теперь мы можем попробовать установить разные скорости обучения, чтобы сравнить производительность классификатора при разных скоростях обучения.

```
lr_list = [0.05, 0.075, 0.1, 0.25, 0.5, 0.75, 1]
```

```
for learning_rate in lr_list:
    gb_clf = GradientBoostingClassifier(n_estimators=20, lear
    gb_clf.fit(X_train, y_train)

    print("Learning rate: ", learning_rate)
    print("Accuracy score (training): {0:.3f}".format(gb_clf.
    print("Accuracy score (validation): {0:.3f}".format(gb_cl
```

Давайте посмотрим, какова была производительность для разных уровней обучения:

```
Learning rate:  0.05
Accuracy score (training): 0.801
Accuracy score (validation): 0.731

Learning rate:  0.075
Accuracy score (training): 0.814
Accuracy score (validation): 0.731

Learning rate:  0.1
Accuracy score (training): 0.812
Accuracy score (validation): 0.724

Learning rate:  0.25
Accuracy score (training): 0.835
Accuracy score (validation): 0.750

Learning rate:  0.5
Accuracy score (training): 0.864
Accuracy score (validation): 0.772

Learning rate:  0.75
Accuracy score (training): 0.875
```

```
Accuracy score (validation): 0.754

Learning rate:  1
Accuracy score (training): 0.875
Accuracy score (validation): 0.739
```

Нас в основном интересует точность классификатора в наборе валидации, но похоже, что скорость обучения 0,5 дает нам лучшую производительность в наборе валидации и хорошую производительность в наборе обучения.

Теперь мы можем оценить классификатор, проверив его точность и создав матрицу путаницы. Давайте создадим новый классификатор и определим наилучшую скорость обучения, которую мы обнаружили.

```
gb_clf2 = GradientBoostingClassifier(n_estimators=20, learnir
gb_clf2.fit(X_train, y_train)
predictions = gb_clf2.predict(X_val)

print("Confusion Matrix:")
print(confusion_matrix(y_val, predictions))

print("Classification Report")
print(classification_report(y_val, predictions))
```

Вот результат работы нашего настроенного классификатора:

```
Confusion Matrix:
```

```
[[142  19]
 [ 42  65]]
Classification Report
              precision    recall  f1-score   support

           0       0.77      0.88      0.82       161
           1       0.77      0.61      0.68       107

    accuracy                           0.77       268
   macro avg       0.77      0.74      0.75       268
weighted avg       0.77      0.77      0.77       268
```

Классификатор XGBoost

Теперь мы поэкспериментируем с классификатором XGBoost.

Как и прежде, давайте начнем с импорта нужных нам библиотек.

```
from xgboost import XGBClassifier
```

Поскольку наши данные уже подготовлены, нам просто нужно подогнать классификатор к обучающим данным:

```
xgb_clf = XGBClassifier()
xgb_clf.fit(X_train, y_train)
```

Теперь, когда классификатор был приспособлен и обучен, мы можем проверить оценку, которую он получает на наборе проверки, используя команду `score` .

```
```

```
score = xgb_clf.score(X_val, y_val)
print(score)
```

Вот результат:

```
0.7761194029850746
```

Кроме того, вы можете предсказать данные `X_val` , а затем проверить точность с помощью `y_val` с помощью `accuracy_score` . Это должно дать вам такой же результат.

Сравнение точности XGBoost с точностью обычного градиентного классификатора показывает, что в этом случае результаты были очень похожи. Однако это не всегда так, и в различных обстоятельствах один из классификаторов может легко работать лучше, чем другой. Попробуйте варьировать аргументы в этой модели, чтобы увидеть, как результат отличается.

Вывод

Модели градиентного бустинга-это мощные алгоритмы, которые могут быть использованы как для классификационных, так и для регрессионных задач. Модели градиентного бустинга могут работать невероятно хорошо на очень сложных наборах данных, но они также склонны к переобучению, с которым можно бороться с помощью нескольких методов, описанных выше. Классификаторы градиентного бустинга также легко реализуются в Scikit-Learn.

Теперь, когда мы реализовали как обычный повышающий классификатор, так и классификатор XGBoost, попробуйте реализовать их оба на одном наборе данных и посмотрите, как сравнивается производительность этих двух классификаторов.

Если вы хотите узнать больше о теории градиентного бустинга, вы можете прочитать больше об этом **здесь <**
http://blog.kaggle.com/2017/01/23/a-kaggle-master-explains-gradient-

boosting/> . Вы также можете узнать больше о других классификаторах, поддерживаемых Scikit-Learn, чтобы сравнить их производительность. Подробнее о классификаторах Scikit-Learn здесь < https://scikit-learn.org/stable/auto_examples/classification/plot_classifier_comparison.html> .

Если вы хотите поиграть с кодом, он находится на GitHub < https://github.com/criticallycode/programming-tutorials/blob/master/boosting_classifier.py> !

Читайте Ещё По Теме:

- **Усиление градиента с использованием Python Xgboost < https://pythobyte.com/gradient-boosting-90d6af87/>**
- **Модель усаживания градиента – утепленная в Python < https://pythobyte.com/gradient-boosting-model-in-python-75749/>**
- **Окончательное руководство по рецидивирующим нейронным сетям в Python < https://pythobyte.com/the-ultimate-guide-to-recurrent-neural-networks-in-python-2dfb0b60/>**

Вверх ↑