



dzis_science 17 мая 2022 в 17:58

Категориальные признаки

🕒 8 мин 👁 42K

Python*, Data Mining*, Data Engineering*

Из песочницы

Не одним One-Hot единым...

Привет, хабр! Хотел бы сделать краткий экскурс про работу с категориальными признаками, который часто на просторах интернета обходят стороной. В данной статье я постараюсь расширить базовые понятия по данной тематике и иллюстрировать их примерами.

*Под **категориальными данными** мы понимаем данные, которые не имеют численного представления, они могут иметь как и два уникальных значения (бинарные признаки), так и более.*

Для работы с признаками надо произвести **кодирование категориальных признаков** - процедуру, которая представляет собой некоторое преобразование категориальных признаков в численное представление по некоторым оговоренным ранее правилам.

Данная тема для меня очень острая, так ей не так в большинстве курсов по DS либо не уделяют совсем, либо уделяют недостаточное количество времени.

Зачем это надо?

Как мы знаем из теории, простым линейным моделям для корректной работы в качестве входных данных мы должны передавать только **численные признаки, поэтому для них необходимость таких преобразований (категориальный признак -> некоторое численное представление) просто необходимо!**

Однако, опять же, возвращаясь к теории, мы знаем, что **решающие деревья могут работать с сырыми категориальными признаками, однако на практике имеет место следующее:**

Попробуем обучить решающее дерево для задачи **бинарной классификации** в реализации sklearn (из модуля tree) на игрушечном датасете:

- Импортируем метод DecisionTreeClassifier из sklearn.tree и конечно же подгрузим pandas:

```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
```

- Далее создадим игрушечный датасет

```
data = pd.DataFrame()
data['A'] = ['a', 'a', 'b', 'a']
data['B'] = ['b', 'b', 'a', 'b']
data['Class'] = [0, 0, 1, 0]
```

	A	B	Class
0	a	b	0
1	a	b	0
2	b	a	1
3	a	b	0

результат выполнения кода

- Обучим модель:

```
tree = DecisionTreeClassifier()
tree.fit(data[['A', 'B']], data['Class'])
```

```
-----  
ValueError                                Traceback (most recent call last)  
<ipython-input-9-deddefae7551> in <module>()  
    1 tree = DecisionTreeClassifier()  
----> 2 tree.fit(data[['A','B']], data['Class'])  
  
-----  
      4 frames -----  
/usr/local/lib/python3.7/dist-packages/pandas/core/generic.py in __array__(self, dtype)  
    1991  
    1992     def __array__(self, dtype: NpDtype | None = None) -> np.ndarray:  
-> 1993         return np.asarray(self._values, dtype=dtype)  
    1994  
    1995     def __array_wrap__(  
  
ValueError: could not convert string to float: 'a'
```

результат выполнения кода

Мы видим ошибку значений ValueError, которая говорит нам, что у нас на вход реализации в sklearn надо подавать строго численные признаки!

Таким образом, мы обосновали преобразование даже для тех моделей, которые в теории могут работать с сырыми категориальными признаками.

Какие есть категориальные кодировщики?

Как говорилось ранее Encoder'ы, они же кодировщики представляют собой некоторое **правило перевода категориальных признаков в численные**.

В первую очередь разберем наиболее популярные из них - **Label** и **One-Hot encoder'ы**.

Label Encoder

Данный тип кодирования является наиболее часто используемым, преобразование представляет собой однозначное соответствие число <-> уникальное значение категориального признака.

Первое (выбранное каким-то образом) уникальное значение кодируется нулем, второе единицей, и так далее, последнее кодируется числом, равным количеству уникальных значений минус единица.

Давайте посмотрим на примере:

- Предположим у нас есть категориальный признак бренда автомобиля, со значениями BMW, Mercedes, Nissan, Infinity, Audi, Volvo, Skoda.
- Создадим искусственный признак brand и закодируем его с помощью реализации sklearn.

```
data = pd.DataFrame()
data['brand'] = ['BMW', 'Mercedes', 'Nissan', 'Infinity', 'Audi', 'Volvo', 'Skoda']*5
```

- Для чистоты эксперимента перемешаем наши данные с помощью метода `sample()`, зафиксируем `random_state` для воспроизводимости и посмотрим на шапку данных:

```
data = data.sample(frac=1,random_state=42).reset_index(drop=True)
data.head(10)
```

	brand
0	Volvo
1	Skoda
2	Infinity
3	BMW
4	Mercedes
5	Mercedes
6	Volvo
7	Volvo
8	Mercedes
9	Nissan

результат выполнения кода

- Применим Label Encoder:

```
from sklearn.preprocessing import LabelEncoder
labelencoder = LabelEncoder()
data_new = labelencoder.fit_transform(data.values)
data_new[:10]
```

```
array([6, 5, 2, 1, 3, 3, 6, 6, 3, 4])
```

результат выполнения кода

- Посмотрим на "правило преобразования" с помощью метода `classes_`

```
labelencoder.classes_
```

```
array(['Audi', 'BMW', 'Infinity', 'Mercedes', 'Nissan', 'Skoda', 'Volvo'],  
      dtype=object)
```

результат выполнения код

Реализация Label Encoder в sklearn прежде всего сортирует по алфавиту уникальные значения, потом присваивает им порядковый номер!

Давайте поговорим про **главный недостаток Label Encoder'a - создание избыточных зависимостей в данных.**

После преобразования получилось, что по данному признаку значение Volvo имеет численное значение 6, а BMW 1, что дает **нам право говорить, что Volvo в 6 раз больше (круче и тд.) чем BMW по признаку brand.**

Однако, в исходных данных таких зависимостей не было, что и является существенным недостатком данного варианта кодирования.

One-Hot Encoder

Данный тип кодирования, основывается на создании бинарных признаков, которые показывают принадлежность к уникальному значению. Проще говоря, на примере нашего признака brand, мы создаем бинарные признаки для всех уникальных значений: brand_Volvo, brand_BMW, ..., где признак принадлежности к бренду Volvo brand_Volvo имеет значение 1, если объект в признаке brand имеет значение Volvo и нуль при всех других. Давайте посмотрим на примере признака brand:

```
from sklearn.preprocessing import OneHotEncoder  
onehotencoder = OneHotEncoder()  
data_new = onehotencoder.fit_transform(data.values)  
pd.DataFrame(data_new.toarray(),  
              columns=onehotencoder.categories_.head(10))
```

	Audi	BMW	Infinity	Mercedes	Nissan	Skoda	Volvo
0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
1	0.0	0.0	0.0	0.0	0.0	1.0	0.0
2	0.0	0.0	1.0	0.0	0.0	0.0	0.0
3	0.0	1.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	1.0	0.0	0.0	0.0
5	0.0	0.0	0.0	1.0	0.0	0.0	0.0
6	0.0	0.0	0.0	0.0	0.0	0.0	1.0
7	0.0	0.0	0.0	0.0	0.0	0.0	1.0
8	0.0	0.0	0.0	1.0	0.0	0.0	0.0
9	0.0	0.0	0.0	0.0	1.0	0.0	0.0

результат выполнения кода

Главный недостаток One-Hot Encoder'a заключается в существенном увеличении объема данных, так как большие по количеству уникальных значений признаки кодируются большим количеством бинарных признаков.

Label Encoder и One-Hot Encoder представлены в библиотеке *sklearn*. Далее рассмотренные кодировщики представлены в библиотеке ***category_encoders***!

Binary Encoder

Для решения проблемы One-Hot Encoding'a с размером получаемого после кодирования пространства, была предложена идея, использующая в себе принцип перевода десятичных чисел в двоичное представление.

Принцип перевода заключается в том, что десятичное число N можно представить $\log(N)$, где \log - логарифм по основанию 2, бинарными значениями, принимающими значения $\{0, 1\}$. Например число 22 можно представить как 10110, т.е 5 битами.

Давайте тогда сформулируем идею кодирования. Предположим у нас есть N уникальных значений категориального признака, тогда мы можем закодировать его, используя не N бинарных признаков, а **всего лишь $\log(N)$** , представляя порядковый номер уникального значения в виде строки двоичного представления.

Вернемся к примеру с brand. Так, у нас всего 7 уникальных значений, то для данного типа кодирования достаточно 3 признаков, вместо 7. Тогда Volvo (первое уникальное значение в признаке, порядковый номер 1, в двоичном 001) будет закодирован как [0,0,1].

Обратите внимание, что в *Binary Encoder*'е уникальные значения никак не сортируются, т.е. которое попало первым в данных будет закодировано меньшим порядковым числом!
Кодирование зависит от расположения строк в данных!

- Посмотрим на преобразование:

```
!pip install category_encoders
from category_encoders.binary import BinaryEncoder
```

- Вызов, как в sklearn:

```
bn = BinaryEncoder()
bn.fit_transform(data.values)[:10]
```

Моя Все
лента потоки

Разработка Администрирование Дизайн Менеджмент Маркетинг Научпоп



	0_0	0_1	0_2
0	0	0	1
1	0	1	0
2	0	1	1
3	1	0	0
4	1	0	1
5	1	0	1
6	0	0	1
7	0	0	1
8	1	0	1
9	1	1	0

результат выполнения кода

Основная проблема данного подхода, что в погоне за оптимизацией количества признаков после преобразования, теряется интерпретируемость бинарных признаков, которая характерна признакам One-Hot Encoder.

Далее, чтобы не растягивать повествование, опишем оставшиеся две группы категориальных энкодеров, о которых вы вряд ли слышали ранее: контрастные и таргет кодировщики.

Contrast Encoding

По своей сути, контрастные энкодеры можно назвать гибридом, между Binary и One-Hot Encoder'ами.

*"Гибридом" они являются потому, что они так же как и One-Hot Encoder кодируют N уникальных признаков несколькими признаками, если быть точнее N-1 признаком, однако они не являются бинарными! Данные признаки принято называть **dummy переменными**.*

Разберем на примере Helmert Encoder и Backward-Difference Encoder:

- **Helmert Encoder** кодирует по следующему принципу: N признаков кодируются матрицей (N, N-1), где на главной диагонали матрицы и выше нее находятся единицы со знаком минус, а сразу под главной диагональю идет порядковый номер значения и ниже него нули.

Важный момент, что при кодировании Helmert Encoder идет сортировка при кодировании уникальных значений!

Для наглядности, закодируем отсортированный список уникальных значений, а затем посмотрим как это будет выглядеть в данных.

```
from category_encoders.helmert import HelmertEncoder
he = HelmertEncoder(drop_invariant=True)
he.fit_transform(sorted(['BMW', 'Mercedes', 'Nissan', 'Infinity', 'Audi', 'Volvo', 'Skoda']))
['Audi', 'BMW', 'Infinity', 'Mercedes', 'Nissan', 'Skoda', 'Volvo']
```

	0_0	0_1	0_2	0_3	0_4	0_5
0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0
1	1.0	-1.0	-1.0	-1.0	-1.0	-1.0
2	0.0	2.0	-1.0	-1.0	-1.0	-1.0
3	0.0	0.0	3.0	-1.0	-1.0	-1.0
4	0.0	0.0	0.0	4.0	-1.0	-1.0
5	0.0	0.0	0.0	0.0	5.0	-1.0
6	0.0	0.0	0.0	0.0	0.0	6.0

результат кодирования, индекс соответствует индексу в списке комментария

Тогда на всех данных кодирование будет иметь следующий вид:


```
he = HelmertEncoder(drop_invariant=True)
he.fit_transform(data.values)[:10]
```

	0_0	0_1	0_2	0_3	0_4	0_5
0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0
1	1.0	-1.0	-1.0	-1.0	-1.0	-1.0
2	0.0	2.0	-1.0	-1.0	-1.0	-1.0
3	0.0	0.0	3.0	-1.0	-1.0	-1.0
4	0.0	0.0	0.0	4.0	-1.0	-1.0
5	0.0	0.0	0.0	4.0	-1.0	-1.0
6	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0
7	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0
8	0.0	0.0	0.0	4.0	-1.0	-1.0
9	0.0	0.0	0.0	0.0	5.0	-1.0

результат выполнения кода

- **Backward-Difference Encoder** кодирует категориальные данные по схожей схеме, что и Helmert Encoder, т.е с разницей до и после главной диагонали. Здесь проще посмотреть иллюстрацию кодирования категориального признака с N=4 уникальными значениями k = N-1 = 3 dummy признаками.

	Dummy_1	Dummy_2	Dummy_3
a	$-\frac{k-1}{k}$	$-\frac{k-2}{k}$	$-\frac{k-3}{k}$
b	$\frac{1}{k}$	$-\frac{k-2}{k}$	$-\frac{k-3}{k}$
c	$\frac{1}{k}$	$\frac{2}{k}$	$-\frac{k-3}{k}$
d	$\frac{1}{k}$	$\frac{2}{k}$	$\frac{3}{k}$

кодирование признака с 4 значениями

Посмотрим на кодирование нашего признака brand, сначала на отсортированных уникальных значениях, потом на всех данных:

```
from category_encoders.backward_difference import BackwardDifferenceEncoder
bd = BackwardDifferenceEncoder(drop_invariant=True)
bd.fit_transform(sorted(['BMW', 'Mercedes', 'Nissan', 'Infinity', 'Audi', 'Volvo', 'Skoda']))
```

	0_0	0_1	0_2	0_3	0_4	0_5
0	-0.857143	-0.714286	-0.571429	-0.428571	-0.285714	-0.142857
1	0.142857	-0.714286	-0.571429	-0.428571	-0.285714	-0.142857
2	0.142857	0.285714	-0.571429	-0.428571	-0.285714	-0.142857
3	0.142857	0.285714	0.428571	-0.428571	-0.285714	-0.142857
4	0.142857	0.285714	0.428571	0.571429	-0.285714	-0.142857
5	0.142857	0.285714	0.428571	0.571429	0.714286	-0.142857
6	0.142857	0.285714	0.428571	0.571429	0.714286	0.857143

результат выполнения кода

```
bd = HelmertEncoder(drop_invariant=True)
bd.fit_transform(data.values)[:10]
```

	0_0	0_1	0_2	0_3	0_4	0_5
0	-0.857143	-0.714286	-0.571429	-0.428571	-0.285714	-0.142857
1	0.142857	-0.714286	-0.571429	-0.428571	-0.285714	-0.142857
2	0.142857	0.285714	-0.571429	-0.428571	-0.285714	-0.142857
3	0.142857	0.285714	0.428571	-0.428571	-0.285714	-0.142857
4	0.142857	0.285714	0.428571	0.571429	-0.285714	-0.142857
5	0.142857	0.285714	0.428571	0.571429	-0.285714	-0.142857
6	-0.857143	-0.714286	-0.571429	-0.428571	-0.285714	-0.142857
7	-0.857143	-0.714286	-0.571429	-0.428571	-0.285714	-0.142857
8	0.142857	0.285714	0.428571	0.571429	-0.285714	-0.142857
9	0.142857	0.285714	0.428571	0.571429	0.714286	-0.142857

результат выполнения кода

Target Encoding

Основная цель, объединяющая данный тип кодировщиков, заключается в **использовании целевой метки, для кодирования** категориальных признаков.

К Target Encoder'ам относятся Target Encoder, Leave-One-Out Encoder, James-Stein Encoder.

*Target Encoder реализован "под капотом" в модели градиентного бустинга над решающими деревьями **CatBoost!***

- **Target Encoder для задачи регрессии** использует **среднее значение целевой метки** по данному значению категориального признака.

*Другими словами, если у нас задача предсказания цены авто, целевая метка - цена авто, то каждое значение марки авто в нашем признаке *brand* кодируется средней ценой автомобиля данного бренда.*

- **Target Encoder для задачи бинарной классификации** использует **вероятность единичного класса** для данного значения категориального признака.

*Другими словами, если у нас задача предсказания цены продажи авто (продано/не продано), единичный класс - успешная продажа авто, то каждое значение марки авто в нашем признаке *brand* кодируется вероятностью продажи авто.*

Важно понимать, что для тестовой выборки кодирования производится значениями, полученными на обучающей выборке. Поэтому, важно смотреть за статистической схожестью выборок, в противном случае теряется главный плюс данного кодирования - **сохранение исходной зависимости между признаком и целевой меткой во время кодирования.**

- **Leave-One-Out Encoder** является расширением Target Encoder'a, в котором, кодирование **конкретного объекта обучающей выборки** производится с использованием для подсчета среднего/вероятности единичного класса **без учета значения данного объекта** (т.е мы как раз удаляем его значение, отсюда и происходит название кодировщика). Для **тестовой выборки** ничем не отличается от Target Encoder'a.
- **James-Stein Encoder** является некоторым **средневзвешенным** между значениями для **данного значения категориального признака и значением для всей выборки**. Важным моментом является тот факт, что из формул для веса B , можно сказать, что данный энкодер и его **оптимальный параметр B определен корректно и однозначно в случае, когда целевая метка распределена нормально!**

James-Stein Encoder определен только для задачи регрессии!

$$JS_i = (1 - B) * mean(y_i) + B * mean(y)$$

$$B = \frac{var(y_i)}{var(y_i) + var(y)}$$

Итоги

В данной статье, мы познакомились с вами с различными методами кодирования категориальных признаков, узнали плюс и минусы их использования. Надеюсь эта информация будет полезна как и начинающим DS, так и уже матерым специалистам. В данной теме мы осветили важный способ кодирования хеширование (Hashing) и, основанный на нем, трюк хеширования (Hashing Trick), который я постараюсь осветить в будущих статьях.

Материал и код подготовил: Андрей Дзись, канал в ТГ @dzis_science.

Материал защищен авторскими правами.

Копирование и использование материалов возможно только с согласия автора, с упоминанием источника.

Теги: категориальные признаки, preprocessing, encoder, ml

Хабы: Python, Data Mining, Data Engineering



5

Карма

0

Рейтинг

Андрей Дзись @dzis_science

DS, веду блог в ТГ @dzis_science.

Подписаться



Telegram

Реклама



пришло время убратся ~~на орбиту~~ на орбите



Комментарии



Здесь пока нет ни одного комментария, вы можете стать первым!

Вы можете оставлять комментарии только к свежим публикациям

Публикации

ЛУЧШИЕ ЗА СУТКИ

ПОХОЖИЕ



SergeyNovak 8 часов назад

Смарт-избушка на курьих ножках без электросетей

Средний 10 мин 2.6K

Тutorial

+37 17 19 +19



PatientZero 9 часов назад

Почему В-деревья быстрые?

Простой 7 мин 4.7K

Обзор

Перевод

+33 84 0



aio350 9 часов назад

Знакомство с WebTransport API

Средний 12 мин 1.6K

Обзор

♦ +20 20 0



vasiliyovchinnikov 11 часов назад

Эпоха Flash: как разработчики в одиночку делали мировые шедевры

🟢 Простой ⌚ 4 мин 👁 2.3K

Ретроспектива

♦ +19 15 20 +20



e2e4e2e4 8 часов назад

Как не заскучать в 1С-разработке

🟢 Простой ⌚ 6 мин 👁 2.5K

Мнение

♦ +18 5 25 +25



Fedor_Trifonov 5 часов назад

Безопасность — это процесс, а не результат

⌚ 8 мин 👁 641

♦ +17 7 5 +5



dairok 10 часов назад

Как систематизировать работу с входящими документами в компании с помощью OCR-инструментов. Часть 1

💧 Средний ⌚ 10 мин 👁 856

♦ +17 10 1 +1



myoffice_ru 4 часа назад

Итоги 2023 года: как изменился МойОфис и рынок офисного ПО

⌚ 6 мин 👁 1K

♦ +16 4 2 +2



AlexHanguery 21 час назад

NVM+RVC = вокал профи?

Средний 17 мин 1.7K

Мнение

+16 23 1 +1



ru_vds 4 часа назад

Почему люди не пользуются вашим продуктом (даже если он может спасти тысячи жизней)

Простой 8 мин 1.1K

Мнение

Перевод

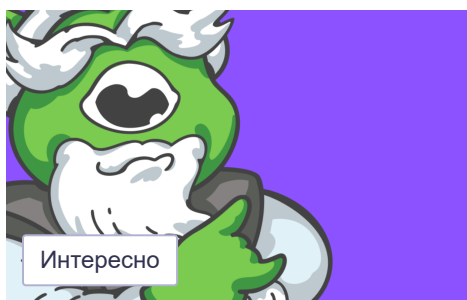
+15 15 1 +1

LLVM-обфускатор, протектор и другие идеи для защиты кода

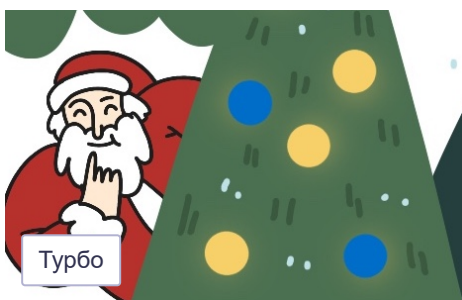
Турбо

Показать еще

МИНУТОЧКУ ВНИМАНИЯ



Глупым вопросам и ошибкам — быть! IT-менторство на ХК



Почаствуй в новогодней охоте за секретами



Кого и как обучать инфобезу: поделитесь своим мнением

КУРСЫ



Мидл Python-разработчик

11 января 2024 · 6 месяцев · 110 000 Р



Аналитик данных буткемп


18 января 2024 · 4 месяца · 128 000 Р



Управление IT и digital-проектами

 Инфографика

9 января 2024 · 7 недель · 47 715 ₽

 Node.js. Профессиональная разработка REST API

9 января 2024 · 2 месяца · 32 900 ₽

Больше курсов на Хабр Карьере

Реклама

РЕКЛАМА · RUVDS.HABR.IO



пришло
время
убраться
~~на орбиту~~
на орбите



Ваш аккаунт

Профиль
Трекер
Диалоги
Настройки
ППА

Разделы

Статьи
Новости
Хабы
Компании
Авторы
Песочница

Информация

Устройство сайта
Для авторов
Для компаний
Документы
Соглашение
Конфиденциальность

Услуги

Корпоративный блог
Медийная реклама
Нативные проекты
Образовательные
программы
Стартапам



[Настройка языка](#)

[Техническая поддержка](#)

© 2006–2023, Habr

ЧИТАЮТ СЕЙЧАС

Реальная Грузия: грустные факты, которые вас разочаруют

 92K  477 **+477**

Как я носил датчик сахара в крови и теперь ем сладкое (часть 1)

 19K  40 **+40**

Эксперт уговорил чат-бота автодилера на базе ChatGPT продать ему новый внедорожник GM за \$1

 23K  26 **+26**

Система водяных «тёплых полов» в квартире и частном доме. Что нужно знать, чтобы не пожалеть о содеянном?

 29K  168 **+168**

JetBrains опубликовала итоги ежегодного опроса среди программистов

 2K  2 **+2**

LLVM-обфускатор, протектор и другие идеи для защиты кода

Турбо

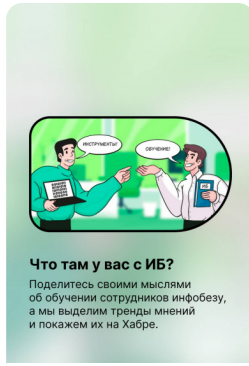
ИСТОРИИ



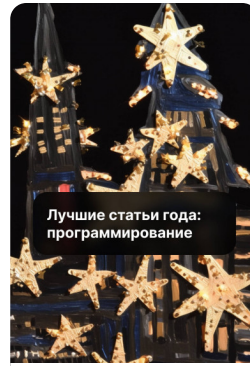
В Новый Год с
Яндекс 360



Годнота от
компаний



Что у вас с ИБ?
Поделись своими мыслями
об обучении сотрудников инфобезу,
а мы выделим тренды мнений
и покажем их на Хабре.



Лучшие статьи года:
программирование



По ту сторону
собеседований
Полезная подборка для тех,
кто проводит собеседования



Поч
пиш
Мы з
140 а
их мо

РАБОТА

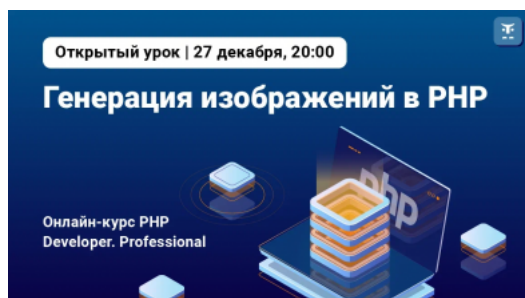
Data Scientist
66 вакансий

Python разработчик
111 вакансий

Django разработчик
43 вакансии

Все вакансии

БЛИЖАЙШИЕ СОБЫТИЯ



Открытый урок
«Генерация изображений
в PHP»

27 декабря 20:00

Онлайн

[Подробнее в календаре](#)

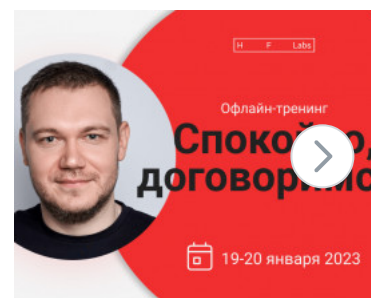


Открытый урок
«Особенности и отличия
получения оффера в ML
в Азии, Европе и США»

29 декабря 19:00

Онлайн

[Подробнее в календаре](#)



«Спокойно,
договоримся!» —
по переговорам и
отношениям с кли
в B2B

19 – 20 января

10:00 – 18:00

Москва

РЕКЛАМА



Узнайте планы компаний
в РФ по работе с K8s
до 2025 года

[→](#)