**Лекция 15**

**Разработка веб-приложения, использующего методы машинного обучения**

Независимо от того, сколько моделей вы создали, оффлайн их смогут увидеть лишь несколько человек. Поэтому необходимо их развертывать, чтобы любой смог поработать с ними посредством UI.

В этой статье мы с нуля развернем модель линейной регрессии с помощью Flask, фреймворка Python для разработки веб-приложений. По результатам этого гайда вы сможете поэкспериментировать с простой моделью машинного обучения в браузере, как показано ниже.



Изображение автора

Руководство разделено на две части. В первой мы создадим модель, а во второй — веб-приложение с нуля. Основная задача — разработка веб-приложения на базе машинного обучения, поэтому вы можете [скачать](https://github.com/ifrankandrade/ml-web-app) уже готовую модель из первой части (model.pkl) и перейти во вторую.

Не хочется читать? Тогда посмотрите [видеоверсию](https://youtu.be/O6BB08Zo2uk)!

**1. Создание простой модели машинного обучения**

Линейная регрессия — один из простейших алгоритмов машинного обучения. В этой части мы построим множественную линейную регрессию, которая прогнозирует стоимость домов. Затем сохраним модель используя pickle и уже в следующем разделе развернем ее с помощью Flask.

**Библиотеки и набор данных**

Для этого раздела нужно установить scikit-learn и pandas:

pip install scikit-learn
pip install pandas

Кроме того, мы будем использовать набор данных, состоящий из трех колонок: стоимость (в тысячах долларов), комнаты (количество комнат в помещении) и расстояние (среднее расстояние до центров занятости). Данные случайно сгенерированы с помощью Excel, их можно найти на [Github](https://github.com/ifrankandrade/ml-web-app.git) и [Google Drive](https://drive.google.com/file/d/1tWpwUG56-7mBLmyETzf7ZtrcuzNMNjul/view?usp=sharing).

Посмотрим на этот набор данных с помощью Pandas:

**import pandas as pd**
df = pd.read\_csv('prices.csv')



Изображение автора

Целью модели линейной регрессии будет прогнозирование стоимости дома, исходя из количества комнат в нем и расстояния до центров занятости.

**Зависимые и независимые переменные**

Для создания модели линейной регрессии необходимо определить зависимые (результаты) и независимые (предикторы) переменные.

* Зависимая переменная: “Стоимость”.
* Независимые переменные: “Комнаты” и “Расстояние”.

Код для установки обеих переменных:

y = df['Value']
X = df[['Rooms', 'Distance']]

**Подгонка и сохранение модели**

Обычно данные нужно обрабатывать и использовать для обучения, а затем проверять результаты. Однако для простоты мы сфокусируемся только на подгонке и развертывании модели.

Для построения регрессионной модели нужно импортировать linear\_model из sklearn.

**from** sklearn **import** linear\_modellm = linear\_model.LinearRegression()
lm.fit(X, y)

После создания модели lm можно прогнозировать value с учетом rooms и distance.

lm.predict([[15, 61]]

Если вы запустите вышеуказанный код, то на экран будет выведена прогнозируемая цена.

Теперь произведем развертывание всего этого на сайте, чтобы каждый мог поработать с этой моделью. В следующей части этого руководства мы перенесем все данные из бэкенда во фронтенд с помощью Flask, но сперва сохраним модель lm, используя pickle:

**import** pickle
pickle.dump(lm, open('model.pkl','wb'))

*Примечание:* как показано в предыдущем примере, lm.predict() получает вводные данные в формате датафрейма [[]]. Помните об этом при изучении следующей части.

**2. Создание веб-приложения с помощью Flask**

Python предлагает различные варианты для создания веб-приложений, и Flask — самый простой их них.

Для этого раздела лучше создать новую среду. Работа с разным окружением дает возможность лучше управлять различными ресурсами, необходимыми для построения модели и развертывания веб-приложения.

**Создание новой виртуальной среды**

Итак, мы создали виртуальную среду с именем deploy-mlmodel-env, выполнив в терминале команду:

python3 -m venv deploy-mlmodel-env

Создадим несколько других папок, чтобы привести в порядок каталог:



Изображение автора

Организовывайте каталог как угодно, но убедитесь, что model.pkl, которую мы сохранили, находится в одной папке с веб-приложением (мы скоро его создадим).

**Установка Flask и быстрая настройка**

Для установки Flask введите следующую команду:

pip install flask

Чтобы проверить, что все работает нормально, выполните несколько действий.

Создайте новый файл и назовите его app.py. Вставьте нижеприведенный код (это пример минимального приложения, взятый из документации по Flask):

**from** flask **import** Flaskapp = Flask**(**\_\_name\_\_**)**@app.route**(**"/"**)**
**def** hello\_world**():**
 **return** "<p>Hello, World!</p>"if \_\_name\_\_ == "\_\_main\_\_":
 app.run()

Откройте терминал, используйте cd для перехода в рабочий каталог, в котором находиться app.py, и введите команду:

python app.py

Должно появиться такое сообщение:



Последняя строка содержит ссылку формата 127.0.0.1:port, которая показывает, где запущено приложение. Копируем ее и вставляем в браузер. Если после нажатия клавиши enter вы увидели сообщение “Hello, World!”, то все сделано правильно.



Источник: документация

Рассмотрим подробно, что делает этот маленький скрипт:

* app = Flask**(**\_\_name\_\_**):** создает экземпляр класса Flask. \_\_name\_\_ — это переменная, которая отображает имя модуля приложения (чтобы мог Flask понять, где искать ресурс типа “шаблон”, который мы будем использовать позже).
* @app.route**(**"/"**): @** представляет декораторы (они изменяют поведение функции или класса). Декоратор route() сообщает Flask, какой URL должен запустить функцию. В данном случае, домашняя страница (/) должна запускать функцию hello\_world().
* def hello\_world**():** возвращает сообщение, которое будет выведено в браузере.

**Загрузка модели, построение функции home и фронтенда**

Пришло время для создания веб-приложения в ранее созданном файле app.py. Сперва импортируем Flask, request и render\_template.

Затем загружаем модель линейной регрессии model.pkl, сохраненную с помощью pickle.

Также мы переименуем функцию hello\_world на home. Она будет отображать HTML-файл index.html при посещении домашней страницы (‘/’).

from flask import Flask, request, render\_template
import pickle

app = Flask(\_\_name\_\_)
model = pickle.load(open('model.pkl', 'rb')) # loading the model

@app.route('/')
def home():
 return render\_template('index.html')

if \_\_name\_\_ == "\_\_main\_\_":
 app.run()

Этот файл будет содержать фронтенд сайта (навигационную панель, кнопки, изображения и т. д.). Создание фронтенда обычно требует некоторых знаний JavaScript и CSS. Тем не менее создавать веб-приложения можно, используя только Bootstrap и базовый HTML.

Для создания фронтенда создайте папку “шаблоны” (используйте только это имя) внутри папки, где находится приложение. Создайте HTML-файл в папке “шаблоны” и дайте ему любое имя. Здесь он назван index.html.

Каталог должен выглядеть примерно так:



Откройте HTML-файл, напишите doc или html и нажмите клавишу tab. Если вам повезет, то IDE сгенерирует базовый HTML-шаблон. Если этого не произошло, то скопируйте и вставьте следующее:

<!DOCTYPE html>
<html>
<head>
 <meta charset="utf-8">
 <title> My Machine Learning Model </title>
</head>
<body>

</body>
</html>

Для добавления начальной загрузки перейдите на этот [сайт](https://getbootstrap.com/docs/5.1/getting-started/introduction/) и скопируйте/вставьте код из секции CSS в секцию <head> файла index.html. Это загрузит их CSS.

Если не смогли найти, то скопируйте это:

<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-1BmE4kWBq78iYhFldvKuhfTAU6auU8tT94WrHftjDbrCEXSU1oBoqyl2QvZ6jIW3" crossorigin="anonymous">

Смеха ради, добавим панель навигации (она не обязательно, но с ней будет легче привыкнуть к bootstrap). Для этого перейдите в секцию [navbar](https://getbootstrap.com/docs/5.1/components/navbar/), скопируйте код и вставьте в секцию <head> (сразу под CSS).

Файл index.html должен выглядеть так (выглядит пугающе, но это просто скопированный код):

<!DOCTYPE html>
<html >
<head>
 <meta charset="UTF-8">
 <title>My Machine Learning Model</title>

 <!-- Quick Start: CSS -->
 <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-1BmE4kWBq78iYhFldvKuhfTAU6auU8tT94WrHftjDbrCEXSU1oBoqyl2QvZ6jIW3" crossorigin="anonymous">

 <!-- Nav Bar -->
 <nav class="navbar navbar-expand-lg navbar-light bg-light">
 <div class="container-fluid">
 <a class="navbar-brand" href="#">Navbar</a>
 <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarSupportedContent" aria-controls="navbarSupportedContent" aria-expanded="false" aria-label="Toggle navigation">
 <span class="navbar-toggler-icon"></span>
 </button>
 <div class="collapse navbar-collapse" id="navbarSupportedContent">
 <ul class="navbar-nav me-auto mb-2 mb-lg-0">
 <li class="nav-item">
 <a class="nav-link active" aria-current="page" href="#">Home</a>
 </li>
 <li class="nav-item">
 <a class="nav-link" href="#">Link</a>
 </li>
 <li class="nav-item dropdown">
 <a class="nav-link dropdown-toggle" href="#" id="navbarDropdown" role="button" data-bs-toggle="dropdown" aria-expanded="false">
 Dropdown
 </a>
 <ul class="dropdown-menu" aria-labelledby="navbarDropdown">
 <li><a class="dropdown-item" href="#">Action</a></li>
 <li><a class="dropdown-item" href="#">Another action</a></li>
 <li><hr class="dropdown-divider"></li>
 <li><a class="dropdown-item" href="#">Something else here</a></li>
 </ul>
 </li>
 <li class="nav-item">
 <a class="nav-link disabled">Disabled</a>
 </li>
 </ul>
 <form class="d-flex">
 <input class="form-control me-2" type="search" placeholder="Search" aria-label="Search">
 <button class="btn btn-outline-success" type="submit">Search</button>
 </form>
 </div>
 </div>
 </nav>

</head>

<body>

</body>
</html>

Перед запуском скрипта меняем цвет навигационной панели на темный путем переключения элементов light на dark в первой строке кода (если оставите как есть, то цвет будет светлым).

<nav class=”navbar navbar-expand-lg navbar-**dark** bg-**dark**”>

Также в секцию body добавим “Hello World!”:

<body>

 <h1>"Hello World!"</h1></body>

Чтобы увидеть все изменения, нажмите CTRL+C для выхода из текущего процесса, запущенного на терминале, а затем выполните команду python app.py.

Теперь при переходе на [http://127.0.0.1:5000](http://127.0.0.1:5000/) вы должны увидеть следующее:



Изображение автора

У нас есть заголовок, навигационная панель и сообщение. Теперь создадим реальное приложение, которое будет отвечать за прогнозирование.

**Создание формы и функции прогнозирования**

Функция прогнозирования будет использовать модель машинного обучения для расчетов. Для этого конечному пользователю необходимо предоставить данные для ввода (комнаты и расстояние) в форме.

Мы построим эту форму внутри секции <body> файла index.html.

Для начала создадим тег <form>. Он будет содержать входные данные, которые конечный пользователь должен предоставить.

<form action="{{url\_for('predict')}}" method=”post”></form>

URL внутри параметра action показывает, куда отправлять данные формы после ее запуска.

В этом случае отправляем их в “/predict” (новая секция сайта). Мы можем либо написать “/predict”, либо использовать функцию url\_for() для построения URL адреса к определенной функции.

Затем внутри раздела формы мы создаем два тега <input> и один <button>. За ними следует вполне понятный код — не будем сильно в него углубляться.

Важно помнить, что теги <input> содержат переменные комнат и расстояний. Чтобы произвести расчет, пользователю необходимо нажать на кнопку “Predict Value!”.

Вот код, который имеется на данный момент:

<body>
 <div class="login">
 <h2>Price Prediction</h2>
 <p>Introduce the number of rooms and distance:</p>
 <!-- Inputs for our ML model -->
 <form action="{{ url\_for('predict')}}"method="post">
 <input type="text" name="rooms" placeholder="Rooms" required="required" />
 <input type="text" name="distance" placeholder="Distance" required="required"/>
 <button type="submit" class="btn btn-primary btn-block btn-large">Predict Value!</button>
 </form>

 <br>
 <br>
 </div>
</body>

Обратите внимание, что здесь включены теги <br>, добавляющие пустые места, и тег <p> с инструкциями. Также эти данные перенесены в <div> с помощью класса login.

Теперь построим функцию прогнозирования!

Внутри файла app.py создаем декоратор с URL “/predict”, который запускает функцию и отображает файл index.html.

@app.route('/predict',methods=['POST'])
**def** predict(): **return** render\_template('index.html')

Теперь нужно импортировать request из flask, чтобы получить доступ к значениям, которые предоставил пользователь в форме.

Эти значения возвращаются в виде словаря, поэтому для получения каждого из них необходимо использовать квадратные скобки. Полученные значения берем в двойные квадратные скобки, чтобы сделать прогноз (это формат, который поддерживает model.pkl).

А вот и функция прогнозирования, над которой мы работам:

@app.route('/predict',methods=['POST'])
def predict():
 """Grabs the input values and uses them to make prediction"""
 rooms = int(request.form["rooms"])
 distance = int(request.form["distance"])
 prediction = model.predict([[rooms, distance]]) # this returns a list e.g. [127.20488798], so pick first element [0]
 output = round(prediction[0], 2)

 return render\_template('index.html', prediction\_text=f'A house with {rooms} rooms and located {distance} meters from the city center has a value of ${output}')

Обратите внимание, что метод render\_template имеет новый параметр с именем prediction\_text. Он содержит сообщение, которое всплывет после нажатия пользователем кнопки прогнозирования.

Это сообщение сейчас в бэкенде. Чтобы отправить его во фронтенд, добавьте переменную prediction\_text в <body> файла index.html:

<body>
 <div class="login">
 ...
 <b> {{ prediction\_text }} </b>
 </div>
</body>

Окончательный вариант index.html выглядит так:

<!DOCTYPE html>
<html >
<head>
 <meta charset="UTF-8">
 <title>My Machine Learning Model</title>

 <!-- Quick Start: CSS -->
 <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-1BmE4kWBq78iYhFldvKuhfTAU6auU8tT94WrHftjDbrCEXSU1oBoqyl2QvZ6jIW3" crossorigin="anonymous">

 <!-- Nav Bar -->
 <nav class="navbar navbar-expand-lg navbar-dark bg-dark">
 <div class="container-fluid">
 <a class="navbar-brand" href="#">Navbar</a>
 <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarSupportedContent" aria-controls="navbarSupportedContent" aria-expanded="false" aria-label="Toggle navigation">
 <span class="navbar-toggler-icon"></span>
 </button>
 <div class="collapse navbar-collapse" id="navbarSupportedContent">
 <ul class="navbar-nav me-auto mb-2 mb-lg-0">
 <li class="nav-item">
 <a class="nav-link active" aria-current="page" href="#">Home</a>
 </li>
 <li class="nav-item">
 <a class="nav-link" href="#">Link</a>
 </li>
 <li class="nav-item dropdown">
 <a class="nav-link dropdown-toggle" href="#" id="navbarDropdown" role="button" data-bs-toggle="dropdown" aria-expanded="false">
 Dropdown
 </a>
 <ul class="dropdown-menu" aria-labelledby="navbarDropdown">
 <li><a class="dropdown-item" href="#">Action</a></li>
 <li><a class="dropdown-item" href="#">Another action</a></li>
 <li><hr class="dropdown-divider"></li>
 <li><a class="dropdown-item" href="#">Something else here</a></li>
 </ul>
 </li>
 <li class="nav-item">
 <a class="nav-link disabled">Disabled</a>
 </li>
 </ul>
 <form class="d-flex">
 <input class="form-control me-2" type="search" placeholder="Search" aria-label="Search">
 <button class="btn btn-outline-success" type="submit">Search</button>
 </form>
 </div>
 </div>
 </nav>

</head>

<body>
 <div class="login">
 <h2>Price Prediction</h2>
 <p>Introduce the number of rooms and distance:</p>
 <!-- Inputs for our ML model -->
 <form action="{{ url\_for('predict')}}" method="post">
 <input type="text" name="rooms" placeholder="Rooms" required="required" />
 <input type="text" name="distance" placeholder="Distance" required="required"/>
 <button type="submit" class="btn btn-primary btn-block btn-large">Predict Value!</button>
 </form>

 <br>
 <br>
 <b> {{ prediction\_text }} </b>
 </div>
</body>

</html>

А app.py — так:

from flask import Flask, request, render\_template
import pickle

app = Flask(\_\_name\_\_)
model = pickle.load(open('model.pkl', 'rb'))

@app.route('/')
def home():
 return render\_template('index.html')

@app.route('/predict',methods=['POST'])
def predict():
 """Grabs the input values and uses them to make prediction"""
 rooms = int(request.form["rooms"])
 distance = int(request.form["distance"])
 prediction = model.predict([[rooms, distance]]) # this returns a list e.g. [127.20488798], so pick first element [0]
 output = round(prediction[0], 2)

 return render\_template('index.html', prediction\_text=f'A house with {rooms} rooms and located {distance} meters from the city center has a value of ${output}')

if \_\_name\_\_ == "\_\_main\_\_":
 app.run()

Вот и все! Теперь можно выйти из текущего процесса с помощью CTRL+C, выполнить в терминале команду python app.py и перейти на [http://127.0.0.1:5000](http://127.0.0.1:5000/).

Теперь модель доступна в браузере!

