# Модуль 1. Создание инфраструктуры для простого проекта машинного обучения.

## В этом модуле

Мы продолжим работу на созданном тестовом стенде из трех виртуальных машин virtualbox, созданных в предыдущих юнитах 1-4 Модуля 1. Мы собрали три виртуальных машины, имитирующие работу серверов:

* сервер data-srv для хранения данных, исходных датасетов и их измененных версий, а также для хранения моделей в бинарном формате pickle,
* сервер ml-srv для проведения экспериментов, обучения моделей и создания программного кода системы,
* сервер prod-srv для эксплуатации модели.

В следующих юнитах 5-8 мы реализуем этапы проекта машинного обучения:

* создадим конвейер для обработки данных,
* обучим модель,
* создадим проект, включающий модель машинного обучения,
* запустим проект в эксплуатацию.

Что нужно знать:

* Для управления наборами данных мы используем dvc.
* Также dvc будет использоваться для трекинга обучения моделей и контроля метрик.
* Контроль версий программного кода мы будем делать с git.
* Исследователи проверяют гипотезы с использованием Jupyter Notebook.
* Наиболее удачные гипотезы оформляются в виде python кода для обучения модели, и обучение выполняется на больших данных, а результат сохраняется в виде обученных моделей в формате pickle.
* dvc используется для сохранения комбинаций «данные»-«модель»-«метрика».
* Разработчики системы используют VSCode для работы над кодом проекта, создают бэкенд и фронтенд системы и добавляют модель машинного обучения, полученную исследователем.
* Для эксплуатации система, включая модель машинного обучения, разворачивается в виде docker контейнеров под управлением docker-compose.

## Содержание юнитов

Какие темы будем изучатьм и:

|  |
| --- |
| 1. Создание и редактирование конвейера для обработки данных. |
| 1. Обучение модели. |
| 1. Создание проекта. |
| 1. Запуск модели и проекта в эксплуатацию. |

# Модуль 1. Юнит 5. Создание конвейера обработки данных.

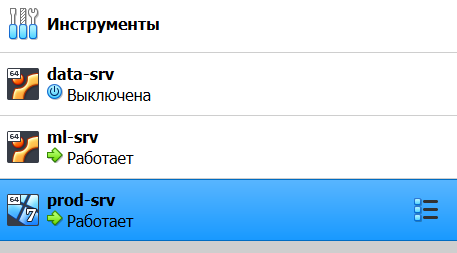
## *Введение*

В этом юните вы научитесь создавать конвейер для обработки данных с помощью dvc, а также сохранять, изменять и сравнивать такие конвейеры между собой.

## *Содержание*

В предыдущих юнитах 1-4 этого Модуля 1 мы создали с помощью virtualbox три виртуальные машины для нашего стендового макета:

* data-srv для хранения данных;
* ml-srv для экспериментов и обучения модели;
* prod-srv для вывода модели в эксплуатацию.



Изображение 1

На сервере data-srv у нас хранятся сырые данные в формате csv, тренировочный и тестовый наборы данных из задачи «Titanic disaster» с сайта https://www.kaggle.com/c/titanic/data.

Датасет, который мы используем, содержит обычные для реальных проектов проблемы с данными:

* пропущенные значения;
* текстовые значения признаков, которые не умеют обрабатывать большинство моделей машинного обучения;
* выбросы, искажающие общую статистику.

Поэтому перед применением машинного обучения данные обрабатывают, в том числе

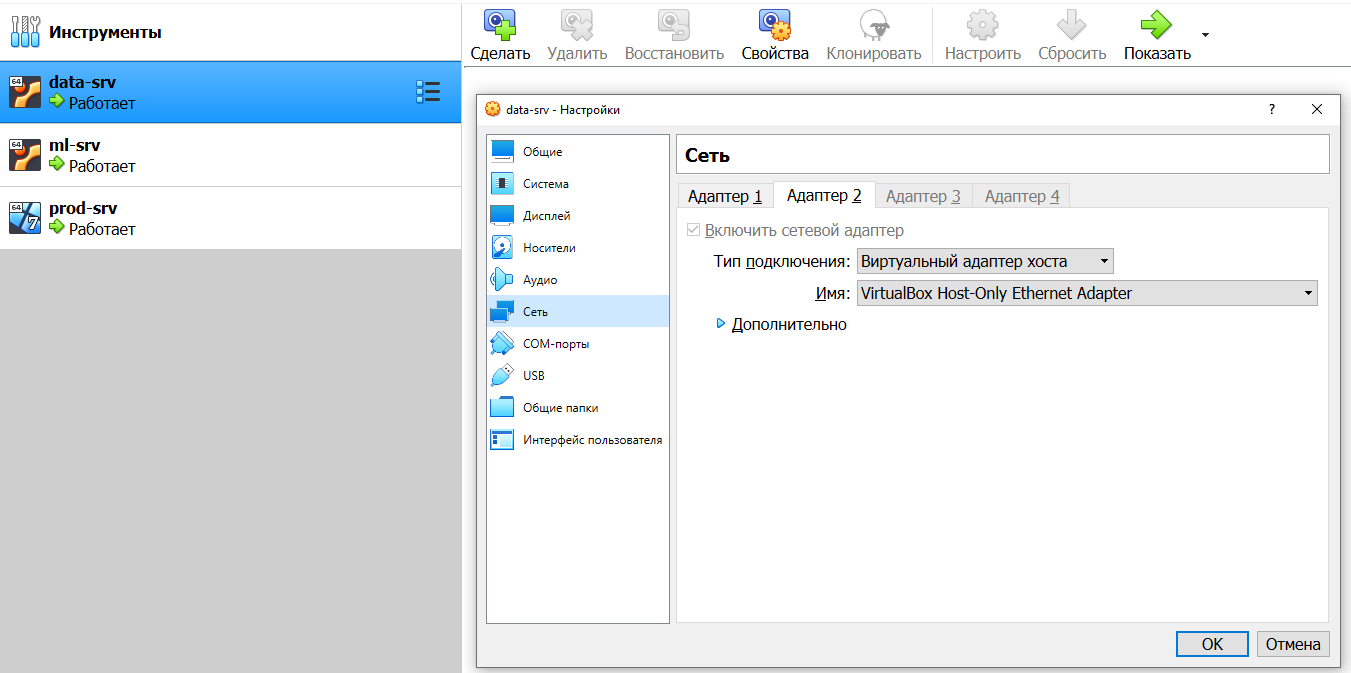
* проводят заполнение (импутацию) пропущенных значений различными способами:
  + средним значением;
  + минимальным значением;
  + максимальным значением;
  + отдельным новым признаком, указывающим на то, что значение у определенного признака отсутствует;
  + либо удаляют экземпляр данных, что не очень хорошо в целом, так как при большом количестве пропущенных значений какого-то признака можно потерять много данных об остальных признаках;
* создают новые признаки, позволяющие ускорить обучение модели или улучшить качество обучения;
* приводят данные конкретных признаков к единому формату, стараясь оптимизировать память для хранения и обработки этих данных, например, если признак принимает всего несколько численных значений, то неэффективно использовать для его хранения и обработки тип float, достаточно одного байта.

Каждый этап обработки данных — это часть общего эксперимента, цель которого состоит в прояснении влияния конкретного действия, применяемого к данным, на общее качество модели машинного обучения. Поскольку в машинном обучении нужно проводить как можно больше разнообразных экспериментов, необходимо иметь возможность запоминать сочетание и содержание этапов конкретного конвейера обработки данных, чтобы иметь возможность проанализировать различия в конвейерах и в любой момент вернуться к нужному.

Открытый программный продукт dvc — это удобный инструмент для создания, редактирования, сохранения и сравнения таких конвейеров обработки данных. В следующих юнитах мы подробно рассмотрим применение dvc для проекта машинного обучения.

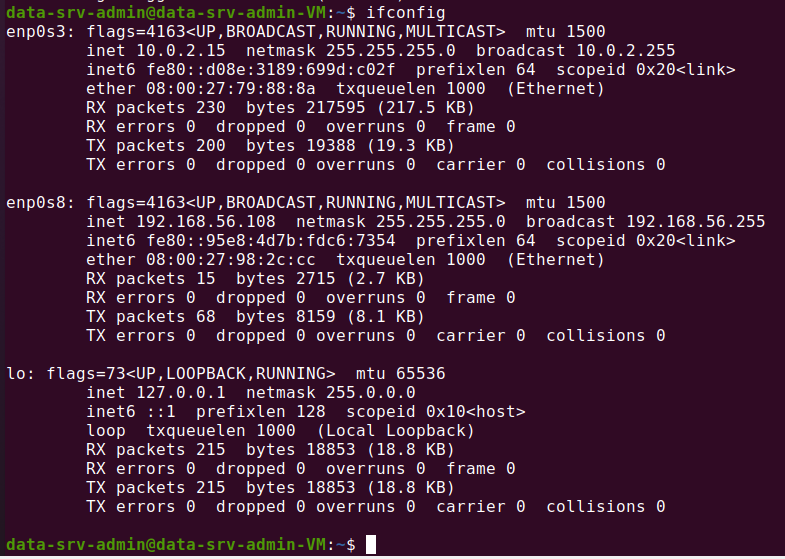
В нашем проекте хранилище данных организовано на виртуальном сервере data-srv, а инженер машинного обучения и аналитик данных работают на сервере ml-srv. Нам необходимо организовать доступ к данным. Приступим.

Для начала давайте загрузим данные через dvc из хранилища данных на сервере data-srv в рабочее пространство на сервере ml-srv. Обратите внимание, что для этого необходимо обеспечить видимость виртуальной машины data-srv из ml-srv. При настройке виртуального сетевого порта в режиме NAT такой связности нет, поэтому на виртуальной машине data-srv необходимо добавить виртуальный сетевой порт в режиме «Виртуальный адаптер хоста»



Изображение 2

В виртуальных машинах virtualbox вы можете добавлять до четырех адаптеров, в данном случае на нашей виртуальной машине data-srv работает два сетевых порта. Убедитесь в этом с помощью команды ifconfig.

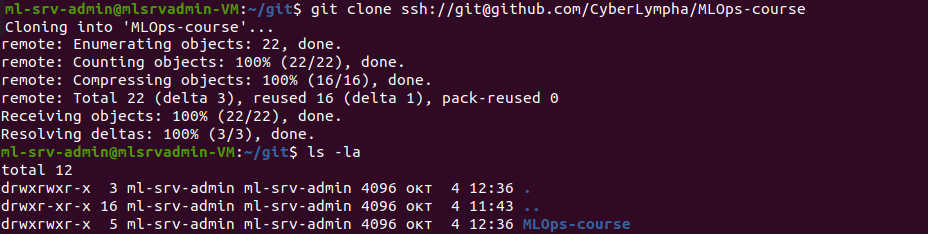


Изображение 3

На сервере data-srv интерфейс enp0s3 с IP адресом 10.0.2.15 является внутренним и не виден из сервера ml-srv, в то время как интерфейс enp0s8 доступен с другой виртуальной машины ml-srv, в том числе для подключения по ssh.

Мы продолжим работать с учебным репозиторием CyberLympha/MLOps-course на github.com, хотя вы также можете создать собственный репозиторий на git и работать с ним. Клонирование рабочего репозитория выполняется с помощью команды

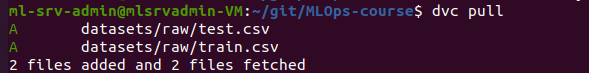
**git clone ssh://git@gitlab.com/CyberLympha/MLOps-course**



Изображение 4

после чего можно выполнить загрузку «сырых» датасетов (raw datasets) из хранилища данных с помощью команды

**dvc pull**



Изображение 5

После этого в папке ./datasets/raw появляются нужные нам файлы train.csv и test.csv.



Изображение 6

Известно, что сырые данные нельзя давать на вход модели машинного обучения, так как они содержат множество ошибок, имеют неэффективный формат представления, содержат выбросы. Кроме того, важнейший подготовительный этап к процессу обучения модели — это конструирование новых признаков на основе сырых данных (feature engineering). Мы продемонстрируем эту часть работы аналитика данных на нашем тестовом примере.

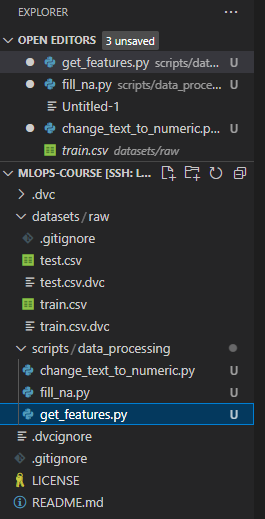
Давайте организуем конвейер обработки данных, который выполняет следующие операции над данными:

* извлекаем три признака: «Age», «Pclass», «Sex»,
* заполняем найденные пропущенные значения средним,
* преобразовываем текстовые значения в численные.

Результат преобразования должен быть записан в файл ./dataset/prepared/prepared\_train.csv. Для этого мы напишем три скрипта обработки:

* get\_features.py,
* fill\_na.py,
* change\_text\_to\_numeric.py,

эти скрипты разместим в папке ./scripts/data-processing:



Изображение 7

Первый скрипт (get\_features.py):

* Получает на вход имя csv файла, содержащего сырые данные.
* Читает построчно содержание этого файла и разделяет каждую строку на отдельные токены, используя разделитель «.csv», при этом учитывается, что в поле «Name» может находиться запятая, например, в значении «Braund, Mr. Owen Harris», эту запятую необходимо обрабатывать отдельно, так как в данном случае она не разделяет значения;
* Признаки Survived, Pclass, Sex, Age записывает в новый csv файл в папку stage1.



Изображение 8

При написании кода мы использовали примеры, приведенные в репозитории example-get-started от авторов dvc, эти и другие примеры можно найти здесь: <https://github.com/iterative/example-get-started/tree/main/src>

В получившемся итоговом файле stage1/train,csv находятся одна целевая переменная (Survived) и три признака (Pclass, Sex, Age). В признаках по-прежнему сохраняются проблемы (отсутствующие значения, строковые переменные), которые мы исправим на следующих шагах обработки данных с использованием скриптов fill\_na.py (заполняет пропущенные значения) и change\_numeric\_to\_text.py, преобразующий строковые значения из переменной категориального типа Sex в числовые, с которыми может работать модель машинного обучения.

Второй скрипт fill\_na.py:

* получает в качестве входного файла csv файл stage1/train.csv,
* для признака Age вычисляет среднее значение и заполняет отсутствующие значения признака Age этими средними значениями,
* записывает все признаки, в том числе преобразованный признак Age, в новый файл stage2/train.csv.

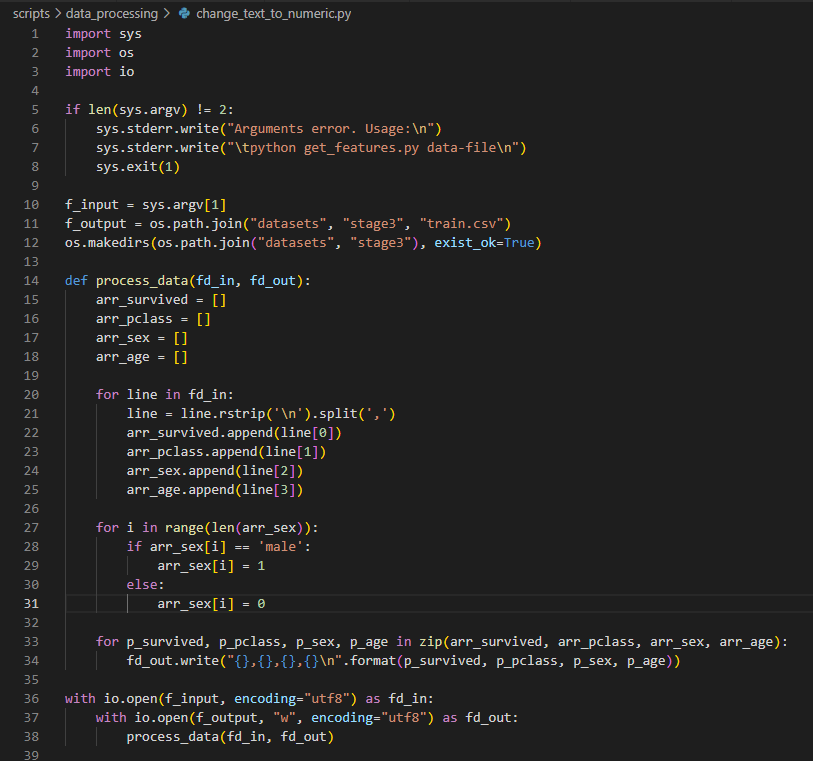


Изображение 9

В итоге после этого этапа предобработки мы можем пользоваться всеми экземплярами данных из исходного датасета, в том числе теми, где значение признака Age было пропущено. Конечно, заполнение этого признака средним значением является сильным допущением. Мы могли бы заполнять это отсутствующее значение в поле возраста Age, ориентируясь на стоимость билета, так как более взрослые пассажиры, как правило, более состоятельные и путешествуют с большим комфортом. Проверка подобных гипотез и проведение экспериментов и составляет суть конструирования признаков и машинного обучения.

Третий скрипт change\_text\_to\_numeric.py:

* считывает входной файл stage2/train.csv,
* преобразует признак Sex из строкового в числовой, меняя значение «male» на «1», а «female» на «0»,
* записывает результат в файл stage3/train.csv.



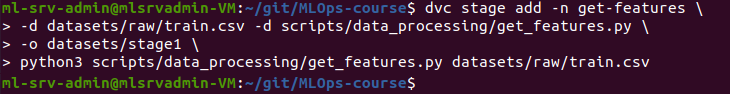
Изображение 10

Давайте теперь для каждого из описанных выше действий над данными создадим свой этап (stage) в контексте dvc, после чего мы сможем эти отдельные шаги объединить в единый конвейер. Это делается с помощью команды

**dvc stage add**

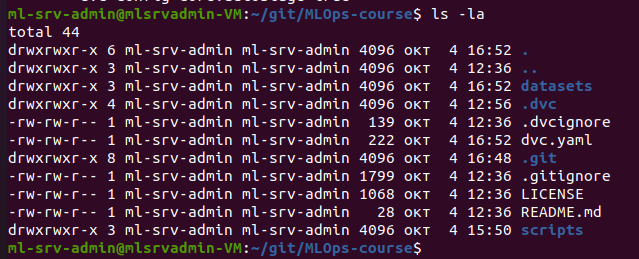
с соответствующим набором параметров, например, флаг –d задает зависимости (dependencies), то есть объекты, от которых зависит выполнение этапа, а флаг –o определяет выходной результат (outputs). Также есть и другие флаги, –p для задания параметров, или –M для работы с метриками модели машинного обучения. Эти флаги мы рассмотрим в следующих юнитах.

Сначала создадим этап на котором из «сырого» (raw) исходного датасета raw/train.csv извлекаются признаки и записываются в папку /stage1.



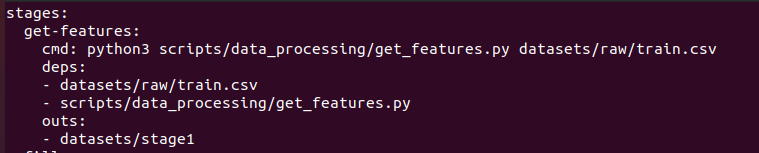
Изображение 11

При выполнении этой команды был создан файл dvc.yaml, который описывает заданную нами часть пайплайна обработки данных.



Изображение 12

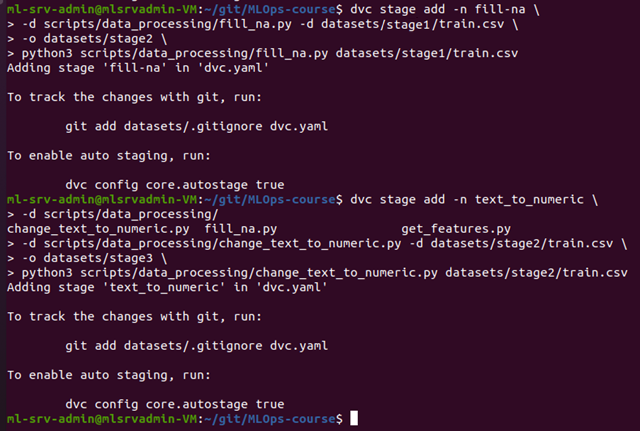
Это текстовый файл, который можно открыть в текстовым редакторе и проанализировать содержание. По сути, мы видим в этом файле информацию, заданную нами в команде dvc stage add.



Изображение 13

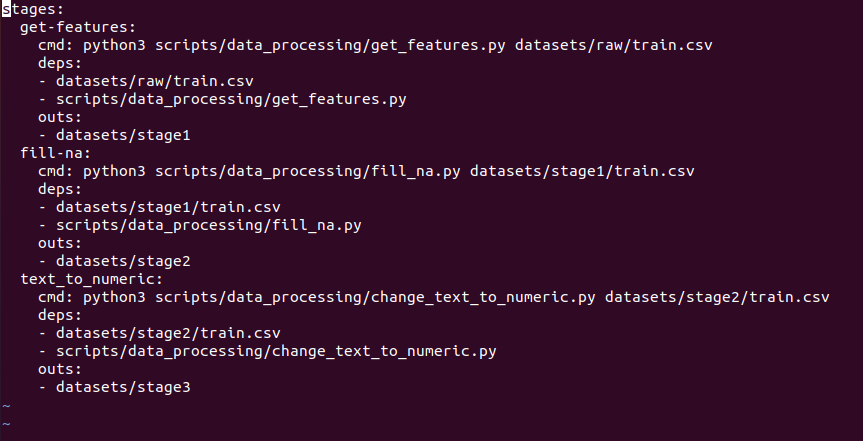
Теперь мы можем описать оставшиеся два этапа обработки исходного датасета, после чего посмотреть итоговый граф операций по обработке данных.

Каждый следующий этап берет артефакты предыдущего, обрабатывает их в соответствии с заданной в скрипте логикой и сохраняет результат своей работы. Последовательное применение правил обработки данных и представляет собой конвейер обработки данных.



Изображение 14

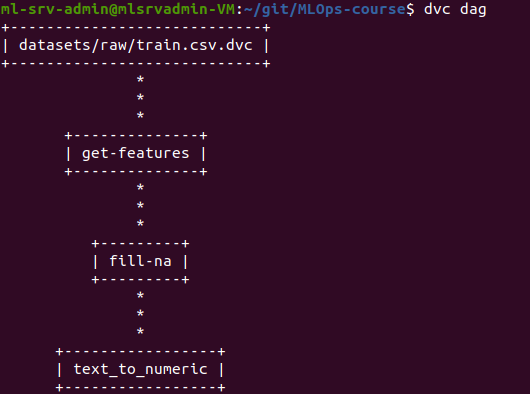
После создания этих трех этапов файл dvc.yaml будет иметь вид



Изображение 15

Зная структуру этого файла, можно вносить изменения непосредственно в сам файл, и они будут применяться при выполнении конвейера операций. Файл dvc.yaml сохраняется в git репозитории, тем самым фиксируется набор операций над данными. Далее любой участник команды может взять этот файл из репозитория и воссоздать конвейер по обработке данных. В этом и состоит большое преимущество в использовании dvc.

Последовательность операций представляет собой ацикличный ориентированный граф (DAG, Directed Acyclic Graph). Для нашего конвейера граф операций над данными имеет следующий вид

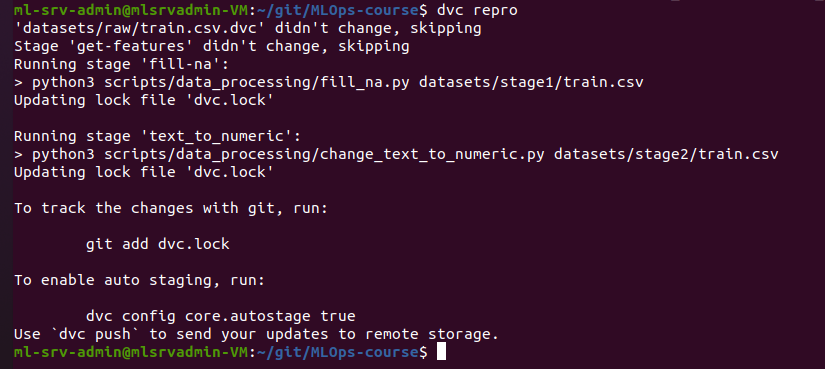


Изображение 16

Теперь командой

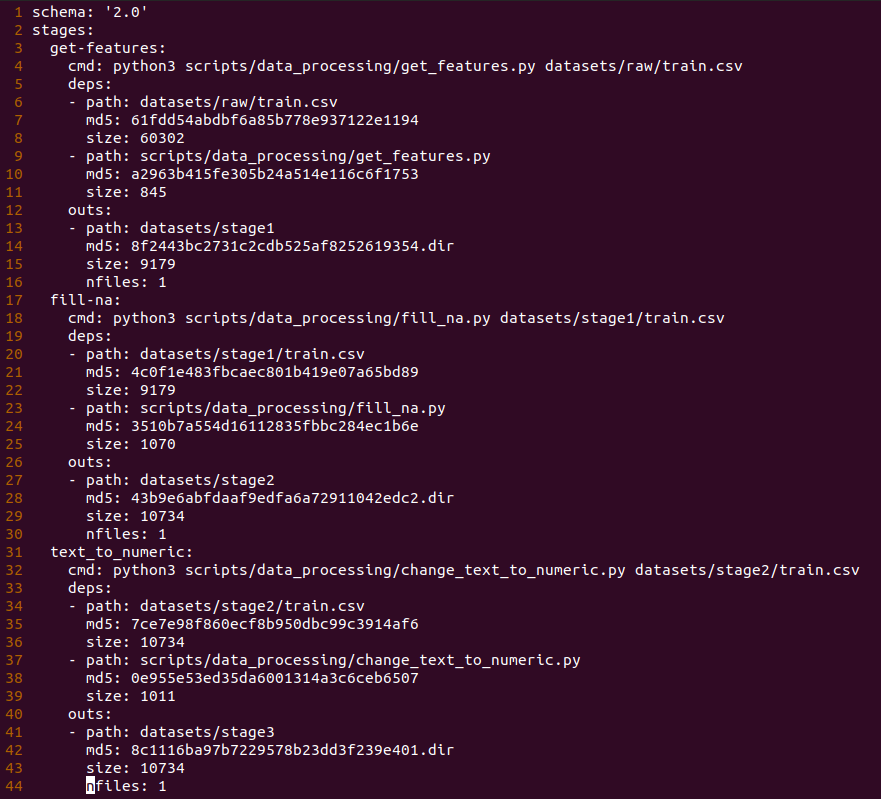
**dvc repro**

вы можете запустить конвейер обработки данных на выполнение



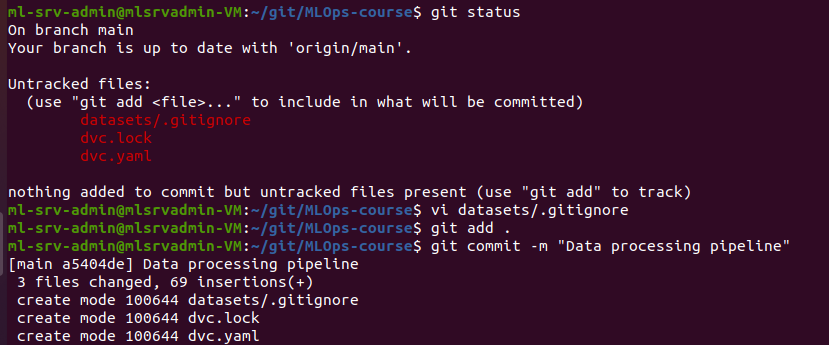
Изображение 17

Автоматически созданный файл dvc.lock содержит информацию о конвейере



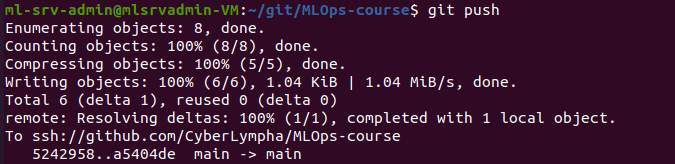
Изображение 18

Созданные файлы автоматически попадают под контроль git, при этом файлы датасетов описываются в .gitignore и отсутствуют в списке для публикации в git.



Изображение 19

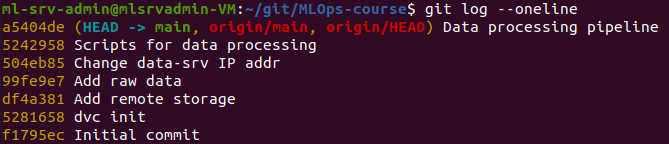
Теперь можно опубликовать сделанные изменения в репозитории git



Изображение 20

Текущее состояние изменений в git репозитории можно посмотреть с помощью команды

**git log - -oneline**



Изображение 21

И, наконец, давайте сохраним в dvc хранилище информацию, относящуюся к датасетам, чтобы любой участник команды мог использовать наш конвейер в своих экспериментах



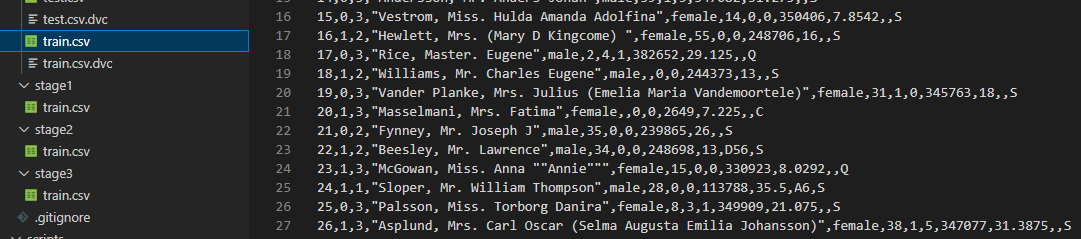
Изображение 22

После этого данные опубликованы в хранилище данных на data-srv



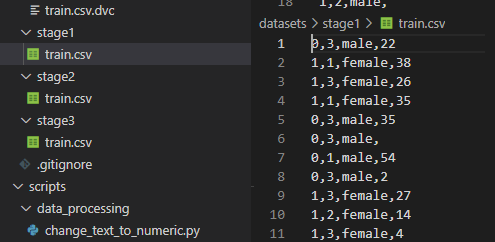
Изображение 23

Итак, мы создали конвейер данных, который последовательно преобразует сырые данные. Сначала исходный файл выглядел так:



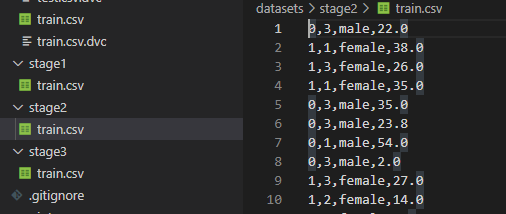
Изображение 24

Затем мы оставили в нем только отдельные нужные нам признаки



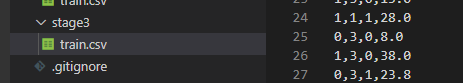
Изображение 25

После этого мы обработали признак Age, заполнили отсутствующие в нем значения средним значением этого признака:



Изображение 26

И, наконец, строковые значения признака Sex мы заменили числовыми значениями:



Изображение 27

Вы можете самостоятельно добавлять или менять этапы, создавая новые конвейеры и сохраняя информацию о них в git и dvc.

## Тест

1. Какая команда используется для создания этапа обработки данных? (0.25)
   1. dvc data
   2. dvc step
   3. dvc add
   4. **dvc stage**
2. В каком файле сохраняется описание конвейера обработки данных dvc? (0.25)
   1. **dvc.yaml**
   2. dvc.data
   3. dvc.descr
   4. dvc.freeze
3. Какая команда выводит граф операций dvc? (0.25)
   1. dvc graph
   2. **dvc dag**
   3. dvc operations
   4. dvc info
4. Какие флаги используются в dvc stage add? (0.25)
   1. **-n**
   2. **-o**
   3. -a
   4. **-p**

## Итоги/выводы

В этом юните мы создали пайплайн по обработке данных, состоящий из нескольких операций. Получившиеся данные используются для обучения модели, о которой идет речь в следующем юните.

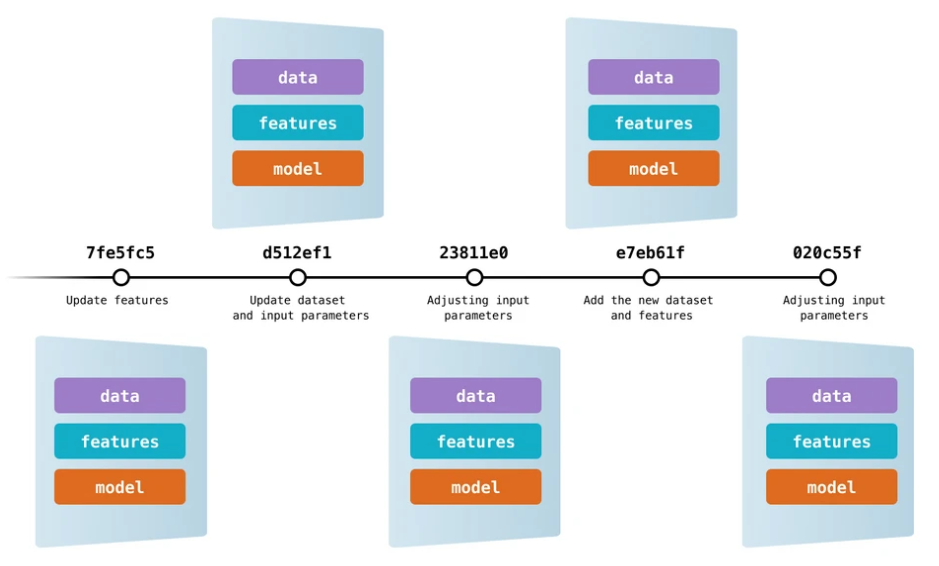
# Модуль 1. Юнит 6. Обучение модели.

## *Введение*

В предыдущем юните вы научились создавать конвейеры операций над данными. Цель этих операций — подготовка качественных данных для обучения модели В этом юните вы научитесь использовать автоматизацию dvc для того, чтобы проводить эксперименты с различными моделями машинного обучения с использованием данных, а также управлять этими экспериментами: редактировать, сохранять и сравнивать между собой. Итогом экспериментов будет обученная модель, которая показывает наилучший результат.

## *Содержание*

В этом юните вы научитесь использовать dvc для проведения экспериментов в проекте машинного обучения. Цель экспериментов — обучение наиболее эффективной модели. Как вы уже знаете, результат работы модели машинного обучения зависит от правил обработки данных, конфигурации окружения, набора гиперпараметров и вычисленных в процессе обучения внутренних параметров модели. Каждый из перечисленных компонентов важен для повторяемости эффективной работы модели. Качество работы модели контролируется с помощью метрик. Утилита dvc позволяет эффективно работать с сочетаниями «данные»-«модель»-«метрика».

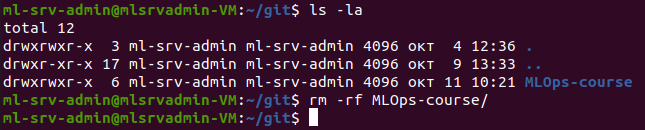


Изображение 28. С официального сайта dvc.org

С использованием dvc вы можете версионировать наборы «данные»-«модель»-«метрика», сохраняя нужные файлы в git репозитории и dvc хранилище.

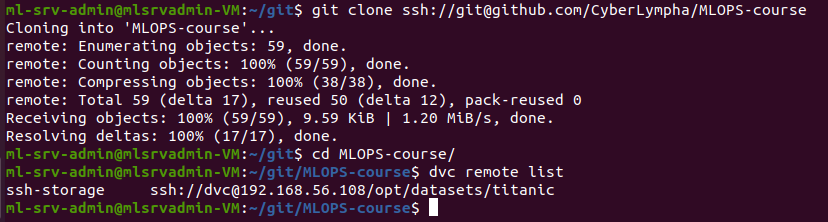
В этом юните вы последовательно выполните загрузку и преобразование данных через dvc, обучите модель и сохраните артефакты обучения — метрики. Сравнение метрик позволяет определить наилучшую модель, а использование инструментов версионирования dvc и git позволяет быстро перейти к ветке, в которой сохранена успешная конфигурация, и повторить эксперименты с минимальными затратами времени.

Давайте удалим на сервере ml-srv все файлы проекта



Изображение 29

и загрузим из git-репозитория и dvc хранилища последнюю актуальную ветку проекта, которая содержит в себе конвейер для обработки данных:



Изображение 30

Напомним, что у нас не задан пароль для удаленного доступа по ssh к хранилищу данных, поскольку пароль небезопасно хранить в git репозитории, поэтому если сразу выполнить операцию dvc pull, то вы получите сообщение об ошибке доступа:



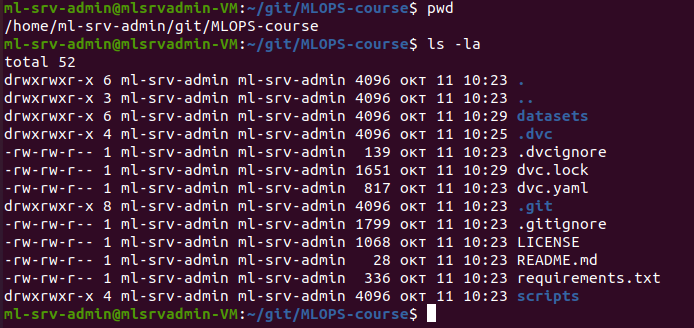
Изображение 31

Необходимо указать пароль с помощью команды:



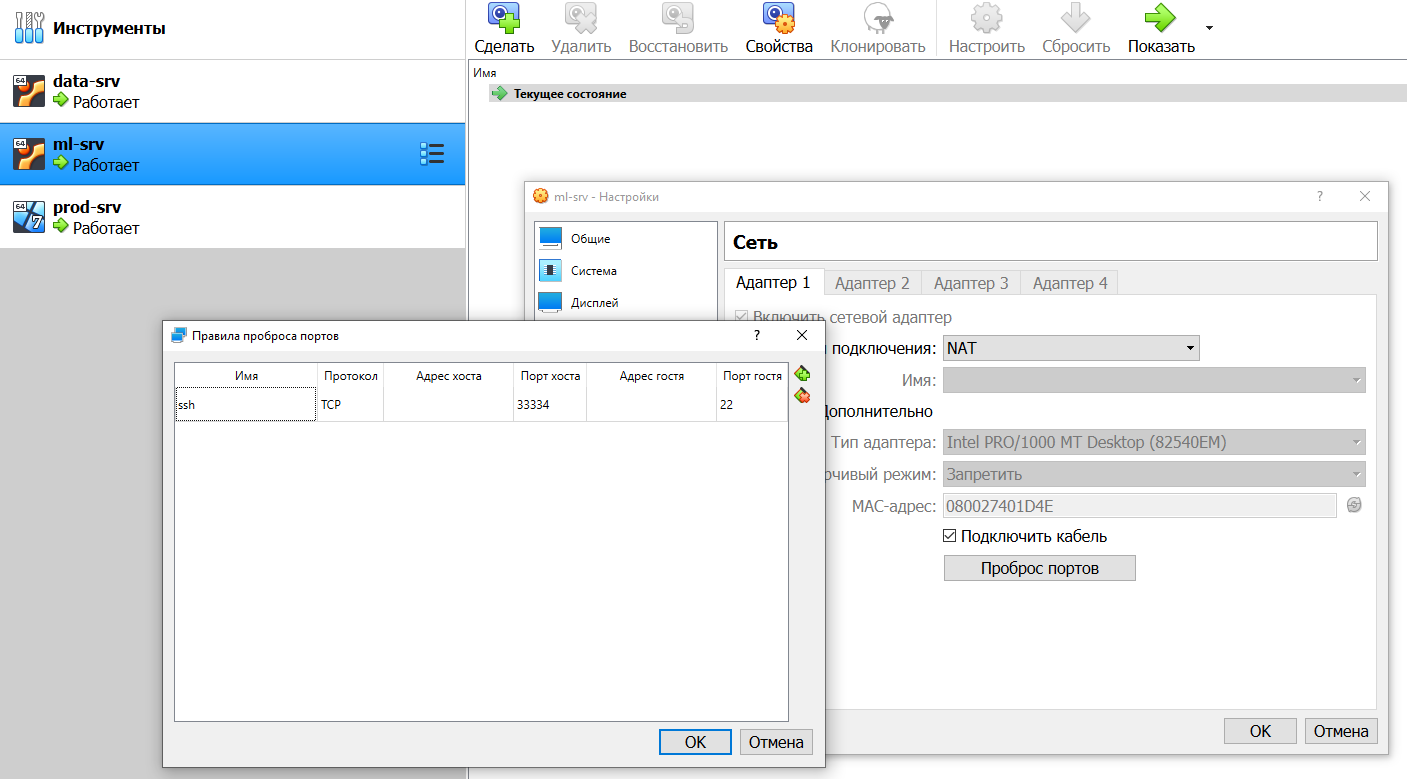
Изображение 32

Теперь нам доступны все файлы проекта:



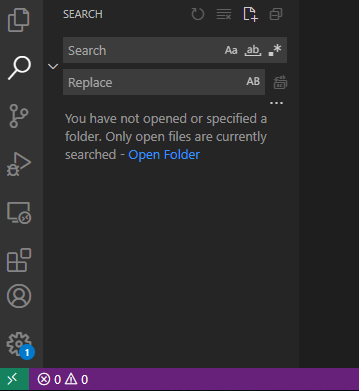
Изображение 33

Для создания скрипта обучения модели откроем рабочий директорий проекта в VSCode. Для виртуального сервера ml-srv у нас проброшен порт 33334 для хостовой системы.



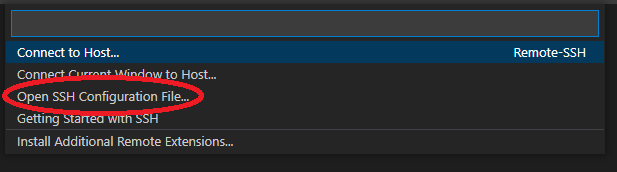
Изображение 34

Этот порт мы указываем при подключении к удаленному серверу в VSCode. Для этого надо нажать на символ в левом нижнем углу рабочего пространства:



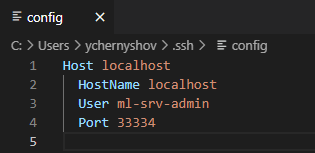
Изображение 36

Затем выбрать в меню файл конфигурации:



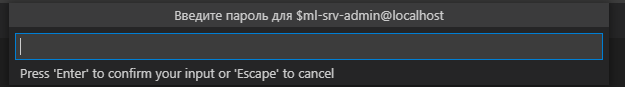
Изображение 37

и установить в нем правильные параметры для подключения к виртуальной машине ml-srv через проброшенный порт 33334 на хосте:



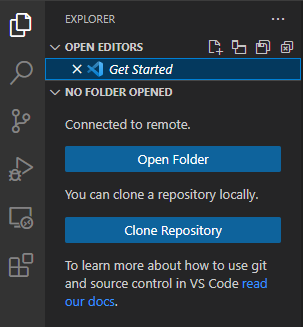
Изображение 38

понадобится ввести пароль:

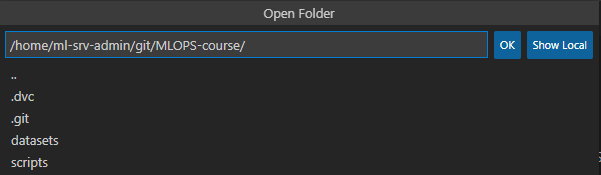


Изображение 39

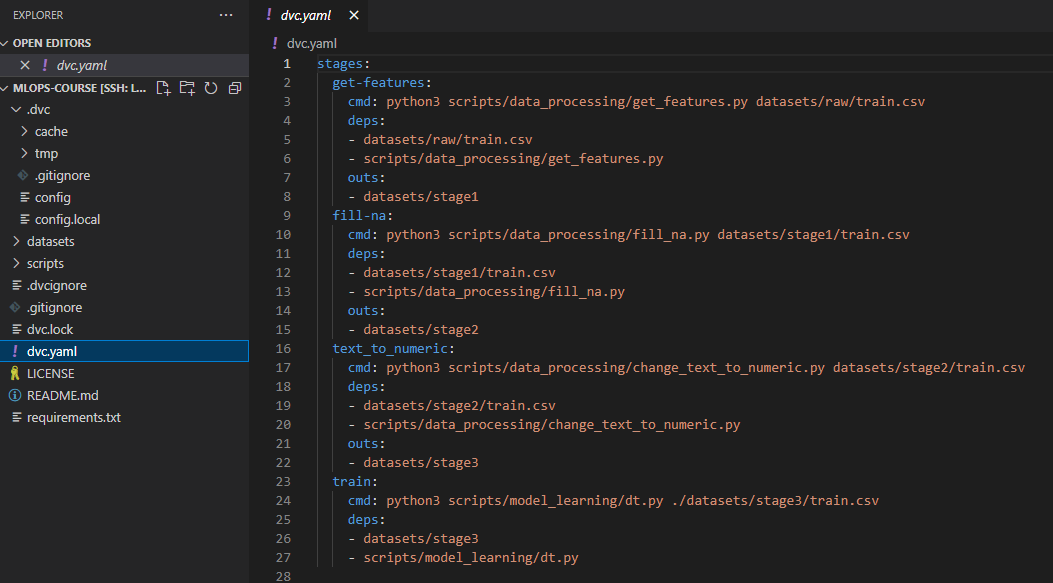
После чего вы можете выбрать рабочую папку и редактировать в ней файлы проекта:



Изображение 40

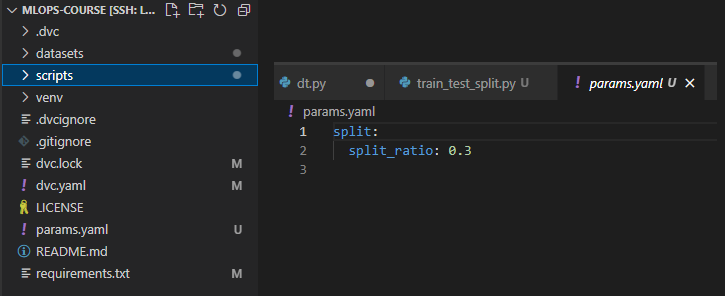


Изображение 41



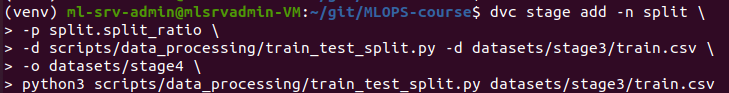
Изображение 42

Созданный нами в предыдущем юните конвейер по обработке данных не разделяет их на тренировочную и тестовую выборку. Давайте добавим этот этап в конвейер по обработке данных, который в результате создаст два файла stage5/train.csv и stage5/test.csv. При этом мы будем задавать отношение между объемом тренировочной и тестовой выборки с помощью параметра. Сначала создадим конфигурационный файл params.yaml, в котором будем сохранять значения параметра:



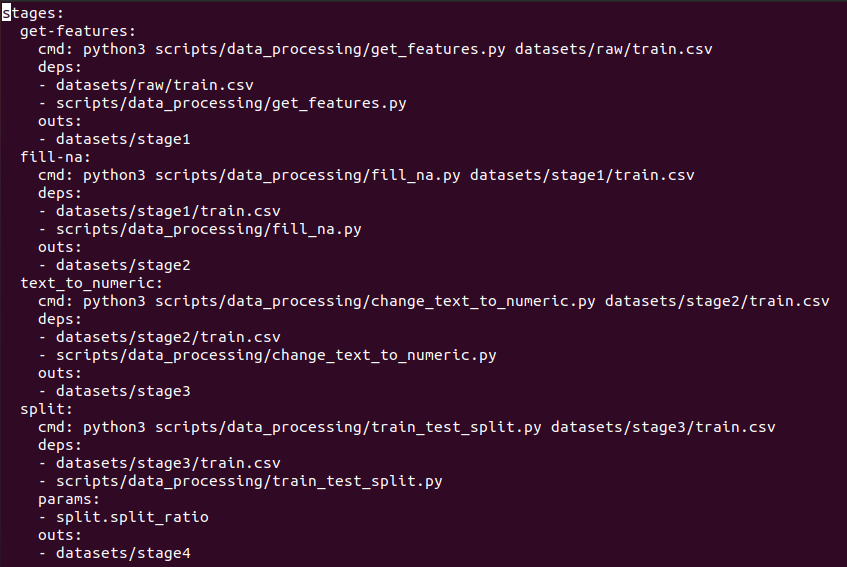
Изображение 43

Теперь давайте добавим новый этап (stage) в наш конвейер, при этом укажем используемый параметр split\_ratio с помощью флага -p:



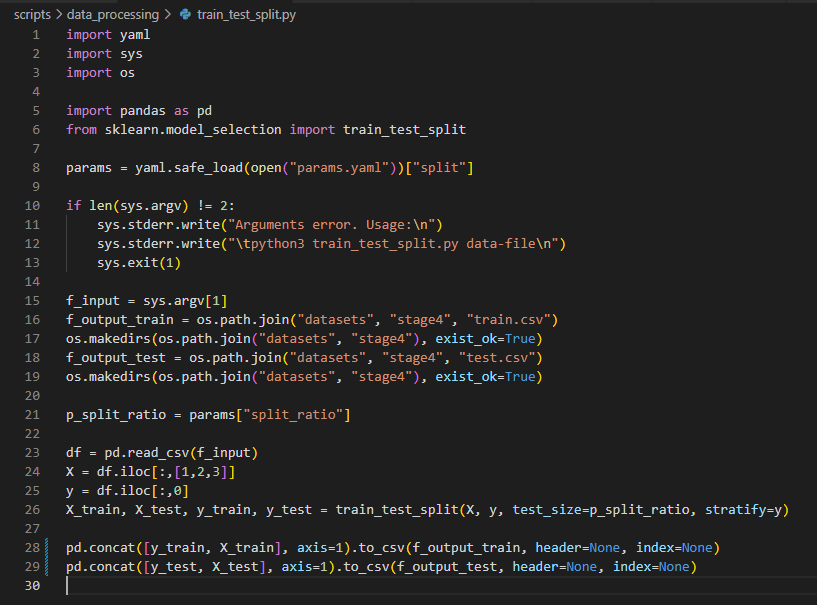
Изображение 44

Информация об этом этапе, в том числе и используемом параметре, добавляется в файл dvc.yaml:



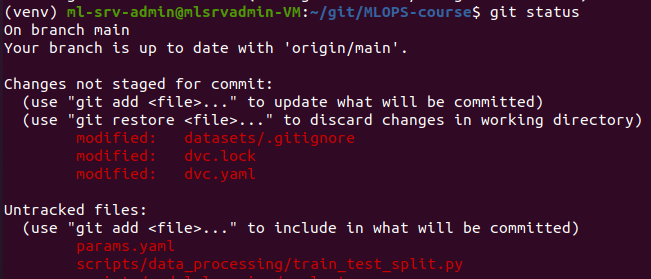
Изображение 45

Скрипт, разделяющий датасет на тренировочные и тестовые данные, имеет вид:



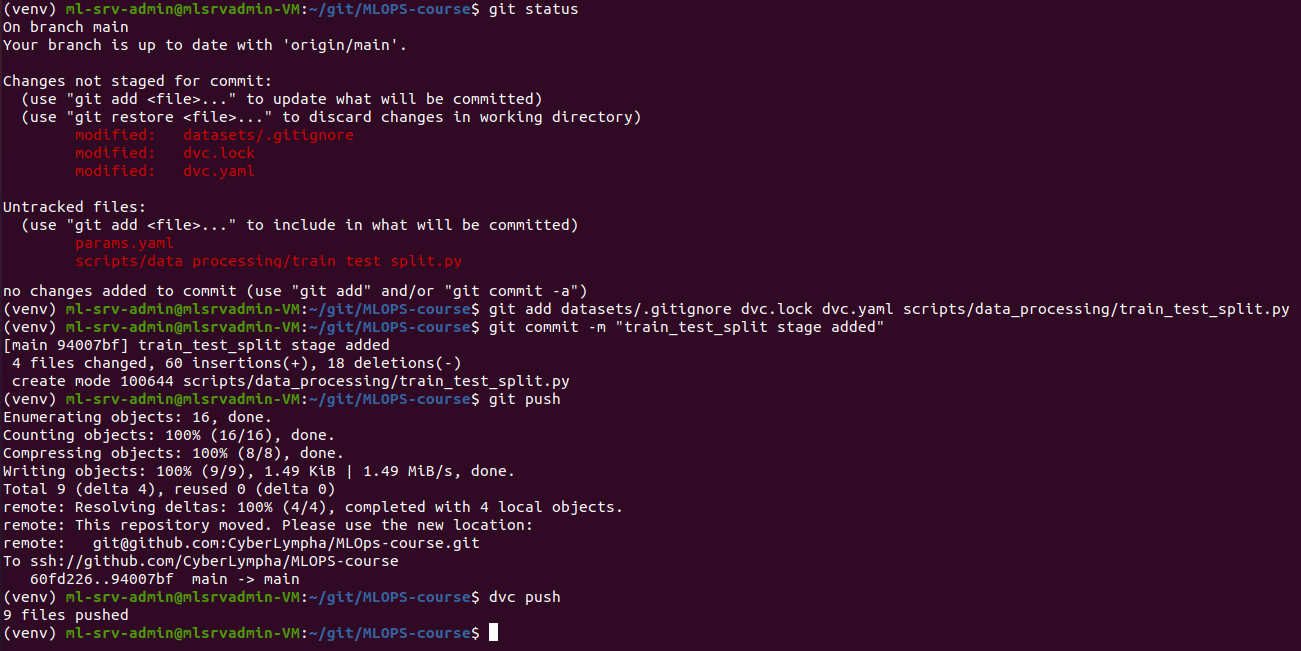
Изображение 46

В рабочем репозитории проекта созданы файлы, относящиеся к добавленному этапу.



Изображение 47

Давайте их опубликуем для последующего использования в git и dvc:



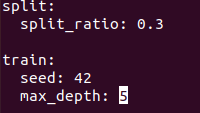
Изображение 48

Теперь у нас есть два набора данных: stage5/train.csv для обучения модели и stage5/test.csv для тестирования модели. Давайте создадим два скрипта, которые выполняют обучение и оценку качества работы модели.

Скрипт scripts/model\_learning/dt.py принимает два параметра:

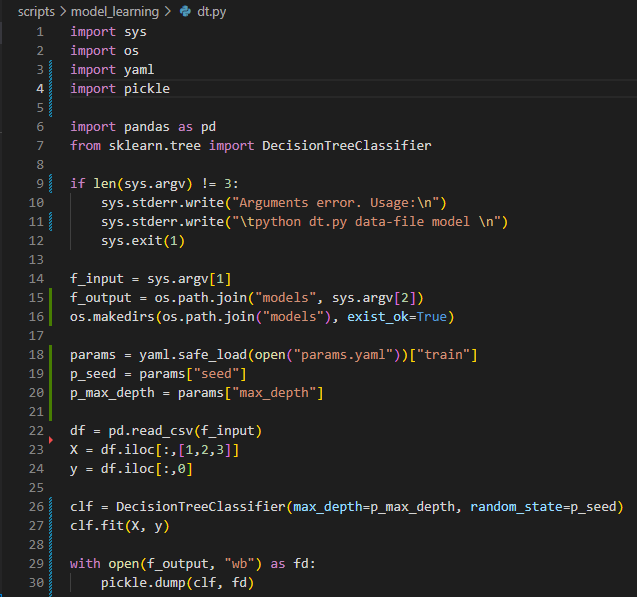
* название файла pickle для сохранения модели,
* датасет для обучения модели.

Этот скрипт создает модель машинного обучения с использованием библиотеки sklearn. В данном примере это классификатор, основанный на модели дерева принятия решений (DecisionTreeClassifier). В качестве параметров для работы модели мы передаем seed и max\_depth, которые сохранены в файле params.yaml и используются для инициализации гиперпараметров random\_state и max\_depth в модели:



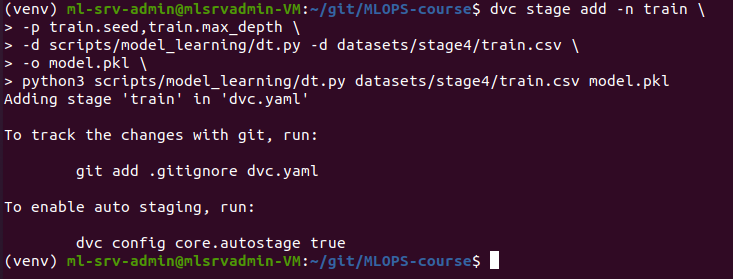
Изображение 49

Скрипт для обучения модели scripts/model\_learn/dt.py выглядит следующим образом:



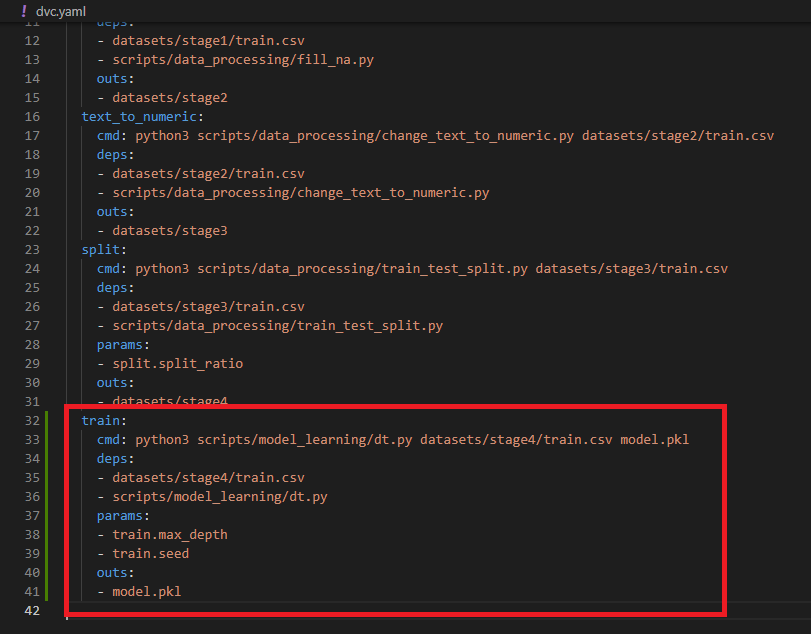
Изображение 50

Чтобы добавить этап (stage) в наш конвейер проекта необходимо выполнить команду:



Изображение 51

В результате выполнения которой обновится файл dvc.yaml, в который добавится информация об этапе train



Изображение 52

В итоге мы получили конвейер, который выбирает данные, производит их обработку и разделение на тренировочную и тестовую выборки и проводит обучение модели ML. В результате работы этого конвейера у нас появились датасеты train.csv для обучения модели, test.csv — для тестирования, и обученная модель в формате pickle. Давайте сохраним этот конвейер в git и dvc.

Теперь оценим качество работы нашего эксперимента с использованием скрипта scripts/model\_learn/evaluate.py. Для оценки качества работы алгоритма машинного обучения используют метрики. В нашей задаче классификации логично применять стандартные метрики accyracy, precision, recall и f1-меру.

После оценки качества модели на тестовой выборке datasets/stage5/test.csv полученные значения сохраняются в файл metrics/evaluation.json в формате json. Для включения метрик в трекинг dvc используется флаг –M с указанием имени файла метрик. В итоге для оценки качества работы модели с контролем выходных артефактов (флаг -o) и данных на входе (флаг -d) необходимо запустить скрипт

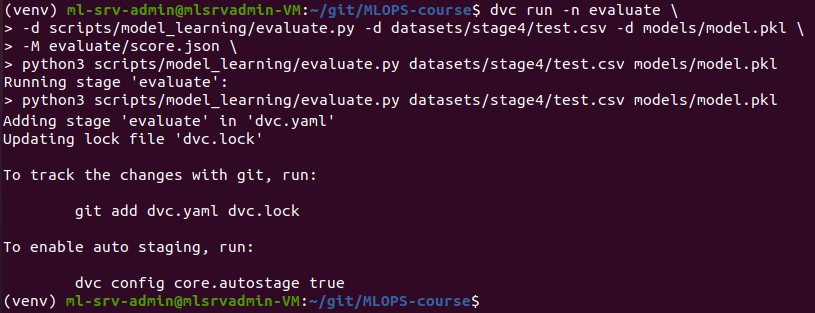
**dvc run -n evaluate \**

**-d scripts/model\_learn/evaluate.py -d datasets/stage4/test.csv \**

**-d models/model.pkl \**

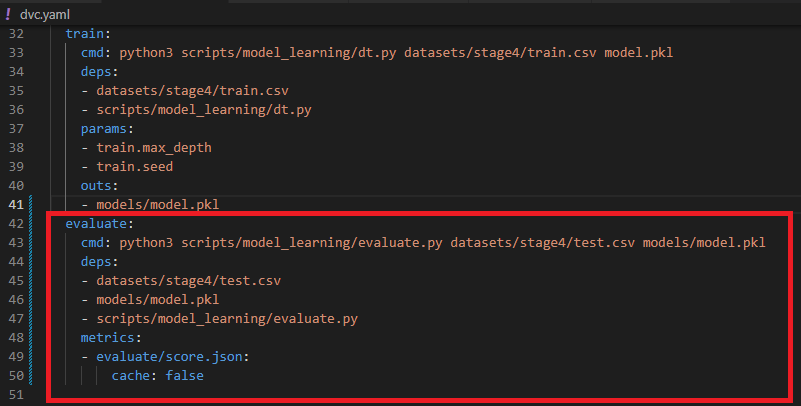
**-M metrics.csv \**

**python3 scripts/model\_learn/evaluate.py datasets/stage4/test.csv models/model.pkl**



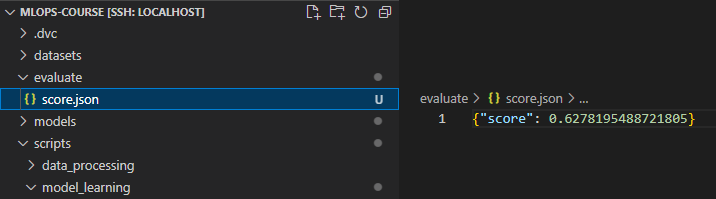
Изображение 53

Применение этой команды добавило дополнительный этап в файле dvc.yaml



Изображение 54

Также появилась папка evaluate c файлом метрики score.json, содержащим значение метрики модели



Изображение 55

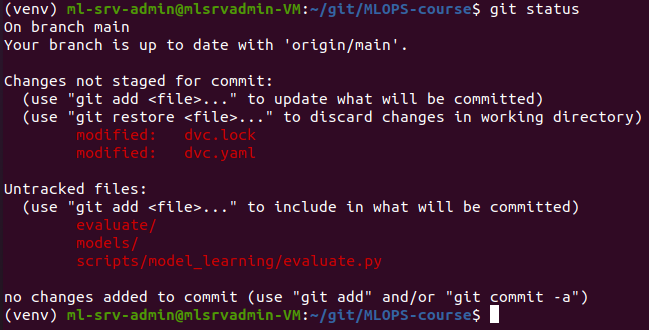
Теперь можно посмотреть информацию по достигнутым метрикам dvc metrics show



Изображение 56

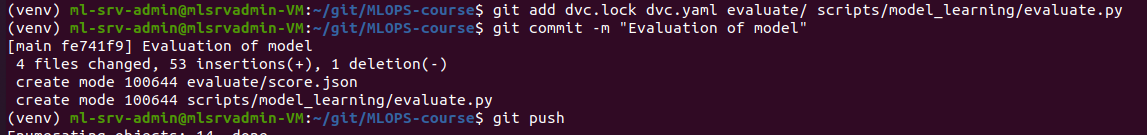
И, наконец, пришло время сохранить окончательный пайплан, включающий выбор и обработку данных, обучение модели с заданными гиперпараметрами и оценку качества работы модели.

Посмотрим статус репозитория, добавленные и измененные файлы. Видим, что изменились файлы с описанием конвейера dvc.lock и dvc.yaml, а также добавился скрипт evaluate.py, папка evaluate с факлом метрик score.json и обновилась папка models, содержащая обученную модель в формате pickle:

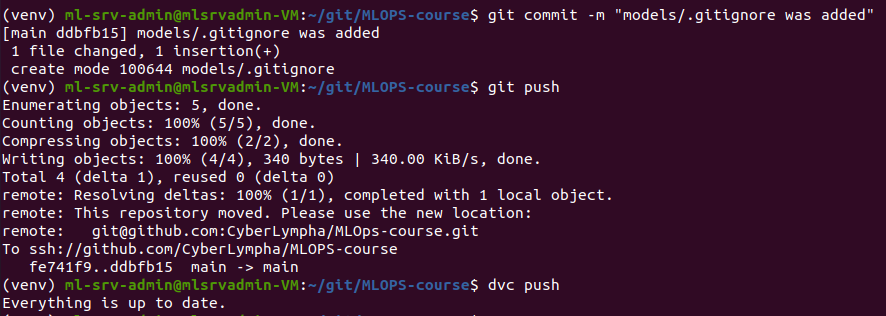


Изображение 57

Давайте внесем эти изменения в git и dvc:

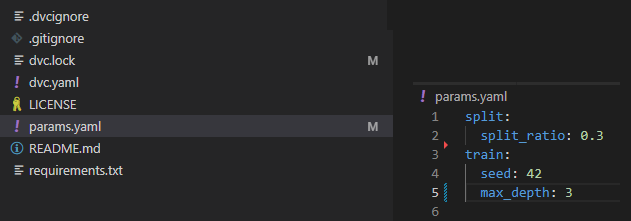


Изображение 58



Изображение 59

Теперь внесем изменения в наш конвейер, поменяем значение гиперпараметра max\_depth, определяющего максимальную глубину дерева принятия решения. Изначально у нас было значение 5, мы ограничим глубину дерева значением 3.



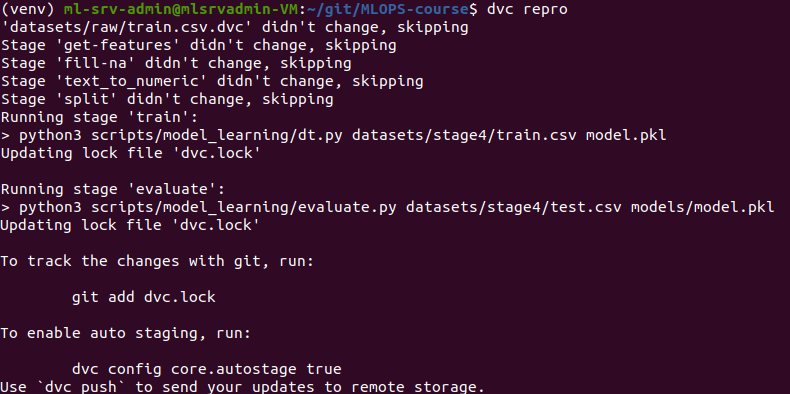
Изображение 60

Это изменение мы сразу можем увидеть как разницу между последней актуальной веткой в git (HEAD) и текущим состоянием рабочего проекта (workspace):



Изображение 61

Теперь запустите эксперимент командой dvc repro, при этом неизмененная часть конвейера будет взята из кэша, что значительно оптимизирует временные затраты на эксперимент



Изображение 62

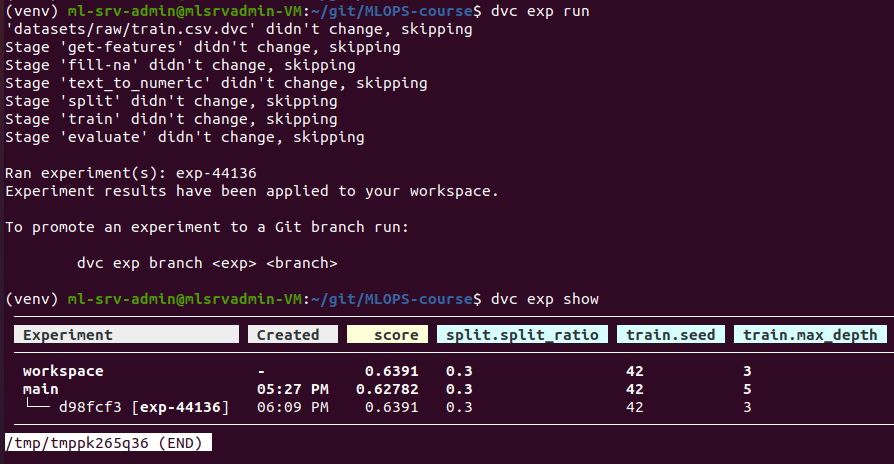
Теперь вы можете сравнить качество работы двух конвейеров:



Изображение 63

Видим, что при изменении значения шиперпараметра max\_depth с 5 до 3 произошел незначительный рост метрики. Улучшенную модель вы можете теперь сохранить в git и dvc.

Для управления экспериментами в dvc необходимо использовать команды dvc exp run и dvc exp show:



Изображение 64

В таблице результатов удобно сопоставлять изменения в данных, гиперпараметрах и метриках.

## Тест

1. На основе чего git понимает, что не требуется публикация файла в git репозитории? (0.25)
   1. Анализируется размер файла, большие файлы не публикуются
   2. Анализируется тип файла (расширение)
   3. **Файлы, перечисленные в .gitignore исключаются из списка для публикации, остальные можно публиковать**
   4. В параметрах команды git push можно указать файлы, не требующие публикации в git
2. Какой флаг задает метрику в команде dvc run? (0.25)
   1. -a
   2. **-M**
   3. -quality
   4. -metrics
3. Какая команда позволяет сравнить метрики в разных экспериментах? (0.25)
   1. dvc compare metrics
   2. **dvc metrics diff**
   3. dvc check
   4. dvc run -M
4. Какая команда позволяет сравнить результаты экспериментов? (0.25)
   1. **dvc exp show**
   2. dvc exp compare
   3. dvc exp check
   4. dvc exp results

## Итоги/выводы

В этом юните вы научились создавать конвейеры проекта машинного обучения полностью, включая этапы проведения экспериментов, а также управление этими экспериментами: контроль метрик, управление ветками проекта через git и dvc. Теперь вы можете выбирать наилучшую модель для выведения ее в эксплуатацию. Вопросам создания сервиса с использованием модели машинного обучения и запуску этого сервиса в эксплуатацию посвящены следующие юниты.

# Модуль 1. Юнит 7. Создание проекта и вывод в эксплуатацию.

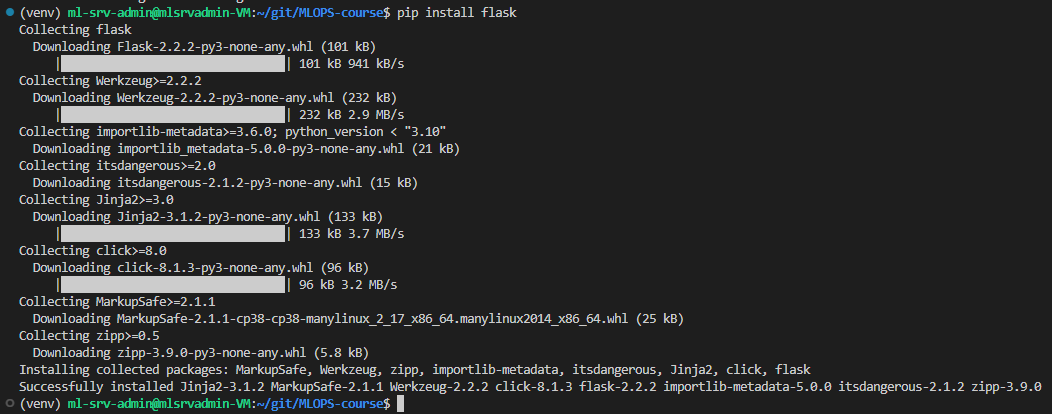
## Введение

В этом юните вы создадите простое web-приложение, которое использует разработанную модель машинного обучения.

## Содержание

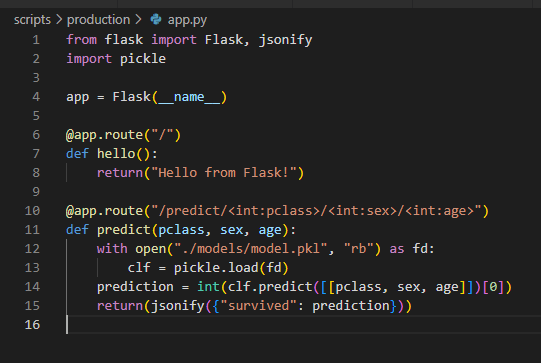
Давайте соберем проект.

Установим в рабочее виртуальное окружение flask



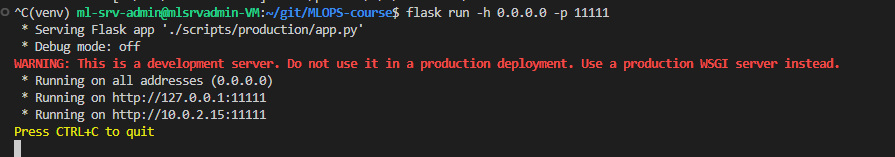
Изображение 65

Создадим папку scripts/production и в ней — исполняемый скрипт, описывающий логику работы web сервера:



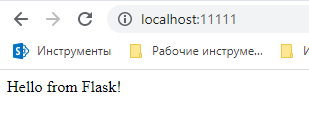
Изображение 66

Теперь, если мы запустим сервер



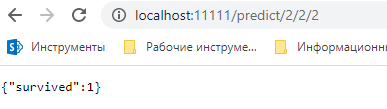
Изображение 67

И зайдем через браузер, то увидим сообщение

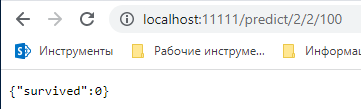


Изображение 68

А если передадим параметры, то получим обратно предсказание модели



Изображение 69



Изображение 70

## Тест

1. Как называется программное обеспечение, обеспечивающее работу приложения через web? (0.25)
   1. Html сервер
   2. Браузер
   3. **Web сервер**
   4. Java сервер
2. Какой фреймворк мы использовали для организации работы сервиса? (0.25)
   1. Django
   2. FastAPI
   3. **Flask**
   4. aiohttp
3. Какой формат использовали для передачи результата? (0.25)
   1. xml
   2. **json**
   3. doc
   4. html
4. Что отвечает за обработку правил маршрутизации url во flask? (0.25)
   1. flask.url
   2. **app.route**
   3. flask.route
   4. flask.web

## Итоги/выводы

В этом юните вы создали простой пользовательский web сервис, позволяющий посмотреть результаты работы модели.

# Практическое задание по модулю

## Постановка задачи

В практическом задании этого модуля вам необходимо создать собственный репозиторий проекта машинного обучения в git и инфраструктуру для хранения артефактов в dvc, повторив шаги, описанные в модуле. Еще нужно установить и настроить необходимое программное обеспечение для работы (virtualbox, putty, VSCode, ssh сервер, системные linux утилиты и библиотеки python). Установку необходимых библиотек python важно осуществлять в виртуальное окружение venv, сохранить версии библиотек в requirements.txt, который потом опубликовать в git.

Для выполнения задания рекомендуем выбрать свой произвольный датасет, например, на kaggle.com или в открытых источниках в интернете.

Построить модель, определить метрику, задать гиперпараметры, провести эксперимент, сохранить эксперимент.

Произвести изменения в конвейере, сравнить результаты метрик.

Этапы выполнения задачи:

1. Создать git репозиторий на github.com
2. Создать две виртуальные машины в virtualbox: для хранения данных (data\_srv), для обучения модели ML (ml\_srv)
   1. Установить ОС (linux, рекомендуемая Ubuntu).
   2. Настроить сетевые интерфейсы (NAT).
   3. Прокинуть порт, запустить sshd и сделать подключение по ssh.
3. В хранилище данных data\_srv
   1. Загрузить датасет (произвольный, свой).
   2. Клонировать репозиторий проекта.
   3. Инициализировать git и dvc, сохранить в git (**commit -m “Init project”**).
   4. Организовать удаленное ssh хранилище, сохранить настройки в git и dvc(**commit -m “Remote ssh server configuration”**).
4. В виртуальной машине ml\_srv
   1. Установить dvc, осуществить клонирование репозитория из git и загрузку датасетов из dvc.
   2. Установить venv, запустить виртуальное окружение.
   3. Установить VSCode и сделать удаленное подключение к ml\_srv через прокинутый порт по ssh, открыть рабочий репозиторий.
   4. Создать в dvc этап (stage) отбора признаков, сохранить в features.
   5. Создать в dvc этап (stage) преобразования признаков, сохранить в prepared.
   6. Проверить качество работы пайплайна данных командой dvc repro.
   7. Осуществить сохранение полученных артефактов в git иdvc (**commit -m “Data pipeline”**).
   8. Создать в dvc этап (stage) обучения модели, гиперпараметры использовать из файла params.yaml.
   9. Выполнить обучение модели dvc командой dvc repro, проверить правильность, сохранить результаты в git и dvc (**commit -m “ML model with parameters”**).
   10. Добавить скрипт для оценки метрики, сохранение в файл evaluate.json, метрику добавить под контроль dvc (флаг -M), проверить правильность, результаты сохранить в git и dvc(**commit -m “Baseline”**).
   11. Осуществить запуск пайплайна модели с изменением параметров, сохранением метрик, контролем экспериментов, добиться лучшей метрики, сохранить в git и dvc (commit-m “Experiment 1”).

## Критерии для проверки

Есть git-репозиторий, содержащий необходимую для повторения эксперимента информацию (git и dvc). В git должны быть сделаны как минимум все указанные в описании этапов решения задачи коммиты (commits).

# Итоги/    выводы по модулю

В юнитах 5-8 вы реализовали на практике этапы проекта машинного обучения с использованием dvc:

* создали конвейер для обработки данных,
* обучили модель, включив этот этап в общий конвейер,
* провели эксперименты с различным набором параметров и сравнили метрики между собой,
* создали проект, включающий модель машинного обучения, в виде простого web-сервиса с использованием flask.

Для решения задач понадобилось установить и настроить вспомогательные средства

* сервер ssh,
* клиент ssh, в нашем случае использовался putty,
* система для разработки, IDE, использовали VSCode,
* утилиты linux, например, ifconfig, netstat, venv
* библиотеки python.

# Термины

|  |  |
| --- | --- |
| CI/CD/CT | Continuous Integration/Continuous Delivery/Continuous Testing – непрерывная интеграция (добавление новых функций в разрабатываемое программное обеспечение), непрерывная доставка (вывод разработанного программного обеспечения в производственную среду) и непрерывное тестирование, методы автоматизации в процессе разработки программного обеспечения. |
| MLOps | Machine Learning Operations: архитектура и набор правил и инструментов, позволяющих автоматизировать задачи проекта машинного обучения. |
| DevOps | Development Operations: архитектура и набор правил и инструментов, позволяющих автоматизировать задачи проекта разработки программного обеспечения. |
| git | Инструмент для версионирования данных. |
| dvc | Data version control. |
| ssh | Security Shell: протокол безопасного доступа с одного компьютера или сервера на другой. Представляет собой техническую архитектуру, включающую специальные данные, называемые ключами безопасности, протокол и специальное программное обеспечение. |
| CPU, GPU | Central Processor Unit, Graphics Processor Unit |
| API | Application Programming Interface |
| DAG | Directed Acyclic Graph: направленный граф без циклов, используется в задаче для определения последовательности выполнения операций. |
| IDE | Integrated Development Environment: интегрированная среда разработки, программное обеспечение, позволяющее повысить эффективность написания программного кода, включающее в себя такие полезные функции как проверка синтаксиса, подсказки, автоподстановки, пошаговая отладка. |

# Список источников

Образовательные и обзорные статьи, справочники, FAQ

1. Статья компании Google об уровнях MLOps в компании, занимающейся проектом ML <https://cloud.google.com/solutions/machine-learning/mlops-continuous-delivery-and-automation-pipelines-in-machine-learning>.
2. Большое количество учебных статей по linux [https://losst.ru](https://losst.ru/)
3. Инструкция по установке python3 и виртуального окружения <https://www.digitalocean.com/community/tutorials/how-to-install-python-3-and-set-up-a-programming-environment-on-an-ubuntu-20-04-server-ru>
4. «Шпаргалка по tmux» <https://habr.com/ru/post/327630>
5. «Установка ssh в ubuntu» <https://losst.ru/ustanovka-ssh-ubuntu-16-04>
6. «Введение в bash» <https://habr.com/ru/post/471242>
7. <https://git-scm.com>
8. https://www.digitalocean.com/community/tutorials/how-to-use-visual-studio-code-for-remote-development-via-the-remote-ssh-plugin-ru

Официальные страницы программных продуктов, инструментов, утилит

1. [https://www.virtualbox.org](https://www.virtualbox.org/wiki/Downloads)
2. [https://jupyter.org](https://jupyter.org/)
3. <https://code.visualstudio.com>
4. https://www.jetbrains.com/ru-ru/pycharm
5. [https://dvc.org](https://dvc.org/)
6. <https://python.org>
7. <https://ubuntu.com>
8. <https://www.docker.com>
9. putty.org

Репозитории и источники, использованные в модуле для практической работы

1. https://catboost.ai/en/docs/concepts/python-reference\_datasets\_titanic
2. пример репозитория проекта <https://github.com/CyberLympha>
3. источник датасета MNIST https://www.cs.toronto.edu/~kriz/cifar.html