Модуль 6. Автоматизация тестирования.

# В этом модуле:

Проверка качества работы разработанного программного обеспечения является важным этапом разработки, позволяющим сократить расходы на устранение замечаний, снизить объем технического долга. В машинном обучении это особенно важно, так как расходы ресурсов на поиск и устранение замечаний в этом случае обычно гораздо выше, чем в обычных проектах.

Различные средства автоматизации позволяют тестировать как отдельные функции и компоненты системы, так и всю систему целиком, в том числе с учетом окружения системы. Прежде всего проверке подвергаются параметры, связанные с бизнес-метриками, оговоренными в техническом задании на разработку системы. Это может быть: объем оперативной памяти, производительность процессора, скорость выполнения операций, точность работы модели, поддержка необходимых протоколов взаимодействия и многое другое. Любое изменение в программном обеспечении или данных может изменить поведение модели машинного обучения, сделать его непредсказуемым.

В данном модуле слушатели познакомятся с основными понятиями из области тестирования программного обеспечения, а также с практическими инструментами для автоматизации тестирования.

Темы, изучаемые в модуле:

1. Роль процесса тестирования в проекте разработки программного обеспечения и какие бывают виды тестирования. Обзор средств автоматизации тестирования. Особенности тестирования для проектов машинного обучения.
2. Тестирование данны.
3. Автоматизация тестирования с unittest и pytest.

# Модуль 6. Юнит 1. Виды тестирования.

*Введение:* В этом юните вы познакомитесь с процессом тестирования, являющимся важным этапом любого проекта разработки программного обеспечения. Вы узнаете какие бывают тесты, на что необходимо обращать внимание при планировании и выполнении тестов. В проектах машинного обучения тестирование тоже необходимо. При этом, из-за особенностей проектов машинного обучения, тестирование в таких проектах имеет свои специфические нюансы. Информация, полученная при изучении этого юнита, позволит лучше понять идеологию тестирования и подготовиться к следующим юнитам, в которых будут рассматриваться инструменты для тестирования их применение для решения практических задач.

*Содержание юнита:*

Назначение тестирования в любом проекте разработки программного обеспечения одинаковое: с помощью тестов проверяют, что после внесенных изменений в программном коде и настройках разрабатываемая система продолжает работать в соответствии с функциональными и техническими требованиями, по которым эта система разрабатывается. Требования бывают самые разнообразные, затрагивающие как небольшие части программного обеспечения, так и всю систему целиком, в том числе во взаимодействии с окружающими системами. Кроме того, проектные команды разработки как правило создают собственные тесты, не относящиеся к бизнес-метрикам, чтобы контролировать качество разработки на внутренних этапах и уменьшить вероятность попадания ошибки в следующий этап разработки. Это позволяет упреждающе обнаруживать проблему и сокращает расходы на поиск и устранение неисправности в сложных проектах, состоящих из множества этапов и различных компонентов.

Тестирование программного обеспечения можно классифицировать разными способами. Например, можно сделать разделение по способу выполнения тестов:

* **Ручное тестирование** выполняется специалистом, взаимодействующим с системой через интерфейсы, например, GUI, API, устройства ввода-вывода. Этот вид тестирования является очень затратным, так как объем выполняемой работы ограничен человеческими возможностями. Скорость такого тестирования невелика. Высока вероятность ошибки из-за «человеческого фактора». На больших объемах этот вид тестирования менее предпочтителен. Лучше всего это тестирование подходит для задач, где при тестировании необходимо проявить смекалку, придумать нестандартную ситуацию, которую нельзя описать шаблонным сценарием и автоматизировать.
* **Автоматическое тестирование** применяется для проверки с использованием стандартных повторяемых операций, которые можно описать в виде шаблонов тестирования. С использованием алгоритмов можно выполнить большой объем тестирования в короткое время. Можно запустить тестирование в течении продолжительного времени, чтобы оценить стабильности системы под длительной нагрузкой. Автоматизация позволяет выполнять тесты в неурочное время, например, ночью. В большинстве проектов тесты как раз проводятся по ночам, когда разработка не ведется. Автоматическое тестирование является важной частью непрерывной интеграции (CI) и непрерывного развертывания (CD), которые мы обсуждали в предыдущих модулях.

Также виды тестирования отличаются по выполняемым задачам и месту в общей архитектуре программного обеспечения:

* модульное тестирование расположено ближе всего к исходному программному коду, проверяет работу отдельных функций и модулей,
* функциональное тестирование проверяет соответствие бизнес требованиям, изложенным в техническом задании, то есть проверяет правильность функционирования разработанной системы,
* интеграционное тестирование проверяет совместную работу нескольких компонентов системы и их взаимодействие между собой,
* системное тестирование проверяет работу системы в целом,
* приемочные тесты предполагают проверку выполнения требований пользователя, описанных в приемо-сдаточной документации, часто пишутся на основе технического задания, в их выполнении как правило принимают участие специалисты заказчика системы,
* регрессионное тестирование проверяет, что внесенные в код или настройки изменения не приводят к найденным ранее и уже известным ошибкам, позволяют убедиться, что изменения не вызывают ошибки повторно,
* тестирование стабильности и производительности проверяет работу системы под нагрузкой, в течении длительного времени, с использованием больших объемов данных, при значительном увеличении параметров, позволяет проверить защитные механизмы системы при перегрузках,
* тестирование безопасности проверяет возможные уязвимости в коде и архитектуре, часто требует привлечения профильных специалистов, пентестеров, однако распространено и применение методов машинного обучения или правил, по которым обнаруживаются уязвимости, в частности, github может подсказывать разработчикам о тех или иных угрозах, которые содержатся в коде программ, опубликованных в репозитории,
* тестирование пользовательского интерфейса (UX/UI, User eXperience User Interface) проверяет правильность работы пользовательских интерфейсов, непротиворечивость, доступность, работоспособность.

Кроме перечисленных существует еще множество других видов тестирования для отдельных специфических задач: тестирование правильности установки программного обеспечения, тестирование удобства использования, тестирование на отказ и восстановление и пр.

Ручное тестирование обладает бесспорным преимуществом в ситуациях, когда важно проверять работу не по сценарию, искать неизвестные уязвимости в работе системы, создавать непредвиденные сложности. Однако разрабатываемые в настоящее время информационные системы как правило имеют сложную архитектуру из множества компонентов, в их разработке участвуют большие команды разработчиков, такие системы работают в сложных окружениях во взаимодействии с другими системами, на различных платформах. Поэтому тестирование сложных систем невозможно без автоматизации. Для разработки скриптов автоматизации тестирования используют разные языки программирования, например, Java, Python, или bash скрипты. Также существуют специальные библиотеки и фреймворки для создания и выполнения тестов.

В таблице ниже приведены некоторые инструменты автоматического тестирования.

| unittest | <https://docs.python.org/3/>  library/unittest.html | Python библиотека для написания unit тестов. |
| --- | --- | --- |
| pytest | https://docs.pytest.org | Python фреймворк для написания небольших удобно читаемых тестов, а также для их масштабирования на большие проекты для функционального тестирования приложений или библиотек. |
| Robot Frameworks | <https://robotframework.org/> | Фреймворк для разработки приемочных автотестов, позволяющий автоматизировать действия пользователей. |
| Mocha | https://mochajs.org/ | Специализированная среда для тестов для javascript |
| JUnit | https://junit.org/junit5/ | Библиотека для модульного тестирования программного обеспечения, написанного на языке Java. |
| TestProject | https://testproject.io/ | Бесплатный фреймворк для автоматизации тестирования. Используется для тестирования мобильных, web и других видов приложений с помощью открытого SDK. Поддерживаются html отчеты. |

Конечно же перечень библиотек и фреймворков для автоматизации гораздо шире и постоянно пополняется.

Давайте теперь познакомимся с важными терминами тестирования.

* испытательный стенд (test fixture), это аппаратная и программная инфраструктура для выполнения тестов, как правило создание и удаление этой инфраструктуры унифицируется и автоматизируется, испытательный стенд может иметь сложную топологию и включать множество компонентов, требующих специальной настройки,
* тестовый случай, тест кейс (test case) это минимальная единица или блок процедуры тестирования, проверяет конкретную рабочую ситуацию или сценарий,
* набор тестов (test suite) это наборов тестов и/или тест кейсов, используется для объединения тестов, которые должны быть выполнены вместе, обычно по принципу группировки тестов для модуля, функциональности, интерфейса,
* исполнитель тестов (test runner) это компонент, который управляет выполнением тестов и предоставляет пользователю результат, может содержать графический интерфейс,
* отчет о тестировании (test report) это документ, который появляется в результате выполнения тестов, содержит результаты.

Простые действия, выполняемые при тестировании, соответствуют так называемому паттерну Arrange-Act-Assert, или “3A”, который организует и упорядочивает входные данные, действия и результаты при выполнении тестовых действий:

* arrange – настроить различные входы для тестирования,
* act – применить входные данные к тестируемому компоненту,
* assert – подтвердить получение ожидаемого результата.

Вышеописанные фреймворки, в том числе unittest и pytest, которые мы будем рассматривать подробнее в дальнейшем, позволяют выполнять эти действия.

В разных командах тестирование организуется по-разному. Иногда за написание тестов несут ответственность сами разработчики. В других случаях существуют специальные отделы тестирования с делением на junior, middle и senior специалистов и руководителем тестирования. Поэтому и степень погружения специалиста по тестированию в детали проекта бывает разная: от самого поверхностного уровня до глубокого погружения в архитектуру продукта, наравне с ведущими разработчиками и архитекторами. Конечно же второй вариант означает более высокую и востребованную квалификацию, которая позволяет тестировщику быть эффективным и уважаемым участником команды, демонстрируя свою полезность в достижении общего успеха.

В проектах машинного обучения есть свои особенности тестирования, связанные с тем, что в машинном обучении используются специальные объекты: данные, модели, среды, конвейеры. Необходимо проверять не только то, что процесс доработал до конца, но и то, что соблюдаются необходимые характеристики в данных и моделях, например, отсутствие аномалий. Поскольку проекты машинного обучения связаны с большими объемами информации, то невозможно обойтись без автоматизации тестов. **Специфику создания, тестирования, вывод в производственное окружение и эксплуатацию учитывает концепция MLOps, используя лучшие методы и инструменты из инженерии программного обеспечения DevOps, а также собственные, уникальные для машинного обучения.**

Для решения практических задач тестирования в проектах машинного обучения достаточно средств, используемых в обычных проектах разработки программного обеспечения, например unittest или pytest. Автоматизированные тесты с использованием этих библиотек делаются частью пайплайна проекта разработки, обучения и вывода в производственную среду модели машинного обучения, позволяя осуществлять проверки качества на отдельных этапах и локализовать проблему как можно раньше, что позволяет существенно снизить расходы на поиск и устранение ошибок.

Однако в проектах машинного обучения есть то, что отличает эти проекты от других, а именно: наборы данных или датасеты. Наличие ошибок в датасетах, как тренировочных, так и реальных, появляющихся в практической эксплуатации модели, может приводить к ошибкам, которые очень сложно диагностировать. Поэтому в проектах машинного обучения большое внимание придают проверке данных.

В основных этапа жизненного цикла модели машинного обучения актуальны обычные для проектов разработки виды тестирования:

* на этапе создания кода для проектов актуальны модульные тесты, проверяющие корректность работы отдельных,
* при включении модели машинного обучения в общую систему или при интеграции модели с другими системами актуально выполнение интеграционных тестов,
* в процессе эксплуатации модели в производственной среде важна организация мониторинга, тестирующего параметры работы модели под реальной нагрузкой и на реальных производственных данных.

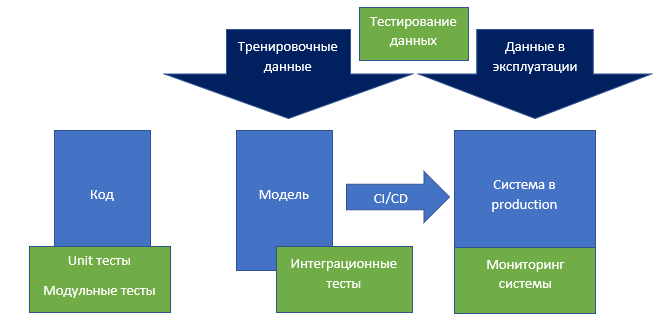


Рисунок «Задачи и объекты тестирования в проекте машинного обучения».

Кроме тестирования кода и интеграционных тестов системы в машинном обучении можно выделить три важных категории тестов, специфических для машинного обучения:

* тестирование признаков (features, фичей) применяется, чтобы проверить важность признака с точки зрения бизнес-цели, например, улучшает ли данный признак точность модели машинного обучения, может быть частью процесса конструирования признаков (feature-engineering),
* тестирование данных на различных этапах, в том числе при сборе данных, конструировании признаков, обучении модели, эксплуатации модели,
* тестирование моделей, что является по сути главной частью машинного обучения, проверяется качество работы модели, ее устойчивость к шумам в данных, способность давать надежные результаты, соблюдение расходования аппаратных и программных ресурсов в рамках технических требований,
* тестирование инфраструктуры ML, в ходе которого проверяется правильность конфигурации элементов инфраструктуры, наличие необходимых ресурсов, правильность зависимостей библиотек. Может выполняться как перед развертыванием системы, так и регулярно в процессе эксплуатации, для контроля параметров системы.

При практическом тестировании модели машинного обучения используют тестовые данные - наборы параметров и датасетов, подготовленные и упорядоченные для последующего использования в тестах, с заранее известным результатом, получение которого проверяется. Тестовые данные бывают валидные и невалидные. На валидных данных проверяется то насколько правильно программа их обрабатывает. На невалидных данных проверяется устойчивость алгоритма к разным шумам, аномалиям, нестандартным ситуациям. Создание тестовых наборов данных может выполняться:

* вручную, с использованием специальных скриптов,
* с использованием программ автоматизации, в том числе специальных генераторов данных,
* с использованием основных средств и функций разработки проекта, например работа напрямую с базой данных на уровне back-end для вноса тестовых данных. Требует внимательность, полезно иметь UI или специальные утилиты, осуществляющие контроль и проверку таких действий и дающие возможность быстро вернуть назад сделанные изменения.

*Тест*

1. Укажите виды тестирования (0.25)
   1. **модульное**
   2. корпусное
   3. **интеграционное**
   4. дифференциальное
2. Что проверяет регрессионное тестирование? (0.25)
   1. Возможность решения задачи с помощью линейной регрессии
   2. Корреляцию между величинами
   3. **Отсутствие ранее известных ошибок после внесения изменений**
   4. Отсутствие новых ошибок после внесения изменений
3. Отметьте фреймворки для автоматизации тестов (0.25)
   1. **pytest**
   2. windows
   3. **unittest**
   4. assembler
4. Что необходимо тестировать в проекте машинного обучения? (0.25)
   1. **модели**
   2. **данные**
   3. **признаки**
   4. **инфраструктуру**

# Итоги/выводы

В этом юните вы узнали о важном этапе любого проекта разработки программного обеспечения и, в том числе, в проектах машинного обучения: о тестировании. Теперь вы знаете каких видов бывает тестирование и чем отличаются друг от друга разные виды: модульное, функциональное, системное, интеграционное, нагрузочное и другие виды тестирования. Также вы знаете плюсы и минусы ручного и автоматического видов тестирования, после чего не должно остаться сомнений в необходимости использования автоматизированного тестирования в проектах машинного обучения, поскольку в таких проектах много данных и параметров для обучения.

В проектах машинного обучения важную роль играют данные, поэтому в следующем юните мы подробнее рассмотрим вопрос тестирования данных.

В юните мы перечислили и кратко описали некоторые фреймворки и библиотеки для автоматизации тестирования. В третьем юните мы подробнее рассмотрим unittets и pytest и используем их для решения практической задачи.

# Модуль 6. Юнит 2. Тестирование данных

*Введение:* Данные составляют важную часть проекта машинного обучения. Без данных невозможно обучить модель и не имеет смысла эксплуатация модели. От ошибок в данных проблемы могут быть как в обучении, так и в эксплуатации. После того как такая ошибка случилась ее сложно обнаружить и понять причину ее появления, так как такие ошибки нельзя обнаружить стандартными средствами разработки, например, пошаговой отладкой программы. Зачастую, для понимания причины ошибки и ее влияния на процесс необходимо заново повторить эксперимент, что затратно или невозможно. Поэтому целесообразно предпринять все меры для того, чтобы ошибки в данных не появлялись вообще, в том числе с использованием тестирования данных. В этом юните вы познакомитесь с практическими аспектами тестирования данных в проектах машинного обучения.

*Содержание юнита:*

Использование данных в реальных практических проектам машинного обучения связано с серьезными сложностями:

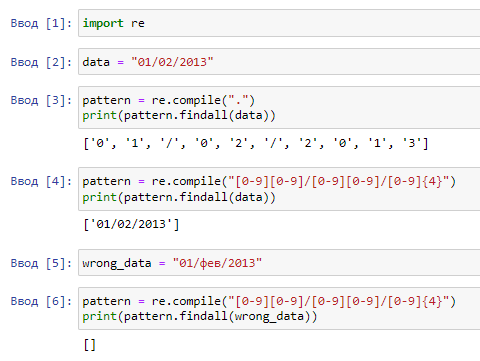
* данных не хватает,
* формат и способ получения данных могут меняться,
* источники данных работают ненадежно,
* нужные данные принадлежат другой компании,
* данные низкого качества,
* данные занимают слишком много места,
* …

Специалист в области машинного обучения как правило сталкивается именно с этими сложностями при переходе от теории и решении соревновательных задач к практическим производственным проектам машинного обучения. Поэтому, в проектах машинного обучения кроме Data Science и Machine Learning специалистов очень значимой является роль специалистов Data Quality, работающих с данными и отвечающих за их качество. Data Quality инженер следит за источниками данных, конвейерами получения данных, отслеживает ошибки в получении и обработке данных, проверяет качество данных. На этом этапе делаются выводы о качестве данных на входе и выходе, характеристиках данных, достаточности данных, соблюдении «контракта данных». Data Quality является частью более важной парадигмы Data Management, которая выходит за рамки этого учебного курса и с которой вы можете при желании ознакомиться по ссылкам, приведенным в конце этого модуля.

При проверке качества данных может проверяться множество параметров, их перечень и требования к ним зависят от специфики конкретного проекта, однако можно выделить наиболее распространенные и часто проверяемые параметры данных:

* уникальность: если в наборе данных много повторяющихся элементов, то такие данные имеют более низкий приоритет для работы с моделью, поскольку не дают полезной информации для обучения модели,
* консистентность, непротиворечивость: в данных не должно быть противоречий, влияющих на обучение модели, например поле «Дата отправки» должна предшествовать полю «Дата получения», если нам важно запомнить такой порядок как важное свойство процесса,
* интегральность: наличие признака, по которому можно связать данные из разных источников, если в модели используются данные из разных источников, отсутствие такого признака делает невозможным их использование,
* целостность: отсутствие пропусков в данных,
* своевременность: данные должны быть за актуальный временной период, имеющий смысл для обучения модели, содержащий характерные для процесса свойства, которые мы хотим выявить с помощью модели машинного обучения,
* соответствие синтаксису: наиболее часто возникают проблемы с датами, суммами, адресами. Пример такой проверки синтаксиса: правило, что идентификатор должен состоять из 10 цифр,
* соответствие статистическим характеристикам,
* корректность данных: при этой проверке в данных надо проверять:
  + соответствие типам,
  + отсутствие поврежденных, искаженных данных,
  + количественные характеристики, например, совпадение общего количества записей до и после обработки,
  + правильности названий признаков, столбцов в датасетах, при изменении формата исходных данных, например, csv файла или json в API, можно столкнуться с тем, что изменились названия признаков.

Полезным инструментом, который позволяет проверять данные по заранее определенным шаблонам, являются регулярные выражения. Это очень мощный инструмент, изучение которого выходит за рамки данного курсы, если вы с ним незнакомы, то рекомендуем обратиться к статьям и книгам, описывающим этот инструмент. Здесь же мы ознакомим только с некоторыми опциями, которые можно применить при тестировании данных. Регулярные выражения это инструмент, который можно применить для извлечения данных по определенному шаблону, который можно задавать очень гибко. В python есть соответствующий модель re для работы с регулярными выражениями, который позволяет удобно создавать и применять шаблоны регулярных выражений. Вот пример такого шаблона, который проверяет формат даты:



Видно, что во втором случае шаблон обнаружил ошибку, не смог определить данные по заданному шаблону.

Давайте рассмотрим наиболее распространенные варианты синтаксиса шаблоны регулярных выражений, которые можно проверять при проверке данных:

1. Основной синтаксис

| . | один символ кроме новой строки |
| --- | --- |
| \. | точка, обратный слеш \ убирает специальные символы |
| \d | одна цифра |
| \D | один символ, кроме цифры |
| \w | один буквенный символ, включая цифры |
| \W | один символ, кроме буквы и цифры |
| \s | один пробельный символ, включая табуляцию и перенос строки |
| \S | один непробельный символ |
| \b | границы слова |
| \n | новая строка |
| \t | табуляция |

1. Модификаторы

| $ | конец строки |
| --- | --- |
| ^ | начало строки |
| ab|cd | соответствует ab или de. |
| [ab-d] | один символ: a, b, c, d |
| [^ab-d] | любой символ, кроме: a, b, c, d |
| () | извлечение элементов в скобках |
| (a(bc)) | извлечение элементов в скобках второго уровня |

1. Повторы

| [ab]{2} | 2 непрерывных появления a или b |
| --- | --- |
| [ab]{2,5} | от 2 до 5 непрерывных появления a или b |
| [ab]{2,} | 2 и больше непрерывных появления a или b |
| + | одно или больше |
| \* | 0 или больше |
| ? | 0 или 1 |

С использованием таких конструкций удобно составлять правила проверки соответствия исследуемых данных определенному шаблону и автоматизировать такую проверку с помощью скриптов.

Важен контроль качества данных на всей цепочке – «data chain». Если данные получаются в результате цепочки операций, например, в ETL-процессе, то необходимо проверять правильность всех изменений, произведенных ETL-процессом. Если данные хранятся в SQL базе, то проверка может быть реализована с помощью тестирующих SQL запросов, проверяющих вышеуказанные характеристики на небольших объемах считываемых данных.

Практическая реализация тестирования такой цепочки тестирования может быть выполнена с использованием уже рассмотренного в Модуле 5 инструмента Apache Aiflow, где операции тестирования можно добавлять в общий конвейер проекта, описываемый с помощью графа операций DAG. Airflow поддерживает операторы для выполнения таких задач, например, оператор **airflow.contrib.operators.bigquery\_check\_operator.BigQueryCheckOperator**, который проверяет данные с использованием SQL запроса к базе данных или другому источнику информации (например, JSON).

Например, мы хотим проверить JSON на отсутствие в event\_id пропущенных или null значений. JSON имеет следующую структуру:

**{"timestamp":1500233640,"user\_id":1234,"event\_id":"view",...}**

**{"timestamp":1500233641,"user\_id":4321,"event\_id":"post",...}**

Вы можете добавить в файл с описанием графа операций Airflow DAG оператор, осуществляющий такую проверку следующим образом

**# Проверяем, что результат равен 0.**

**expected = 0**

**sql = """**

**SELECT COUNT(\*) AS event\_id\_null\_count**

**FROM event\_log.event\_log\_{{ id }}**

**WHERE JSON\_EXTRACT(event\_id) IS NULL**

**"""**

**checker = BigQueryValueCheckOperator(**

**dag=dag,**

**task\_id='bq\_checker',**

**bigquery\_conn\_id='bq\_connection\_id',**

**sql=sql,**

**pass\_value=expected)**

Далее этот оператор следует встроить в общий граф с использованием стандартных средств Airflow, которые вы уже изучили.

Полезной опцией системы тестирования данных, особенно применяемой в реальной эксплуатации, является возможность направлять уведомления в мессенджеры сотруднику, осуществляющему оперативную поддержку или отвечающему за качество данных. Такая опция поддерживается, например, в Apache Airflow. Для подключения этой возможности необходимо создать и добавить в Airflow DAG следующий оператор:

**slack = SlackAPIPostOperator(**

**dag=dag,**

**task\_id='post\_error\_message\_to\_slack',**

**token=YOUR\_SLACK\_TOKEN,**

**channel='#data-quality',**

**username='airflow',**

**text='event\_log on {{ yesterday\_ds\_nodash }} has record(s) whose event\_id is null.',**

**trigger\_rule=TriggerRule.ALL\_FAILED)**

*Тест*

1. Что необходимо тестировать в данных в проекте машинного обучения? (0.25)
   1. **Соответствие новых данных заданным типам**
   2. **Изменение структуры, например, названий признаков**
   3. **Сохранение статистических характеристик**
   4. Надежность источника
2. Что проверяет Data Quality инженер (0.25)
   1. **Уникальность данных**
   2. **Целостность данных**
   3. **Соответствие синтаксису**
   4. **Своевременность данных**
3. Что такое невалидные тестовые данные? (0.25)
   1. Данные, непригодные для проверки модели
   2. **Данные с ошибками для проверки работы во внештатных ситуациях**
   3. Данные из непроверенных источников
   4. Данные, ссылка на которые потеряла актуальность
4. Что такое регулярные выражения (0.25)
   1. Корректные математические выражения
   2. **Формальный язык поиска данных и манипуляций с ними**
   3. Повторяющиеся регулярно операторы, например, в цикле
   4. Функции, запускаемые по расписанию

# Итоги/выводы

В этом юните отдельно рассмотрен процесс тестирования данных в проектах машинного обучения. Теперь вы знаете с какими проблемами можно столкнуться при использовании некачественных данных в проекте машинного обучения и о важной роли Data Quality инженера. Кроме того, теперь вы знаете о регулярных выражениях и о том как их можно применять для проверки правильности данных, а также о более сложном примере проверки с использованием типовых операторов графа операций DAG Airflow.

# Модуль 6. Юнит 3. Модульное и функциональное тестирование

*Введение:* В этом юните вы узнаете о стандартных python библиотеках для создания модульных и функциональных тестов, а также выполните практическое задание с применением этих библиотек.

*Содержание юнита:*

Наиболее популярными для создания функциональных тестов являются python библиотеки unittest и pytest. Большинство задач тестирования в проектах разработки программного обеспечения вполне успешно решается с использованием этих библиотек. В этом юните мы их рассмотрим подробнее.

Создание unittest было вдохновлено другим популярным инструментом, JUnit, библиотекой для тестирования программного обеспечения на языке Java. Ранее эта библиотека называлась pyunit. Версия для python2 называется unittest2. Unittest поддерживает важные функции:

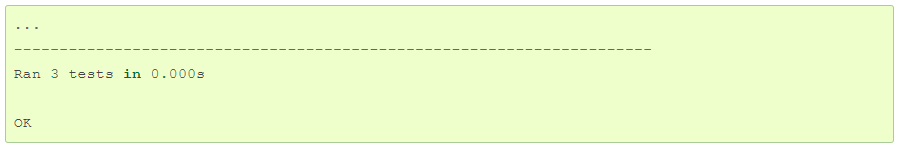
* автоматизация тестирования,
* переиспользование кода для выполнения одинаковых операций в разных тестах, например, подготовка к тесту и завершение теста,
* агрегация тестов в коллекции,
* независимая работа тестов от контролирующей системы.



Пример использования unittest, с сайта https://docs.python.org/3/library/unittest.html#

Создаваемый в этом примере тестовой кейс наследуется от стандартного класса unittest.TestCase и содержит методы, выполняющие тестовые действия с помощью стандартных функций unittest. Каждый тест содержит метод **assertEqual()** для проверки выполнения равенства, **assertTrue()** или **assertFalse()** для проверки выполнения условия и **assertRaises()** для проверки появления исключительной ситуации. Метод **setUp()** позволяет описать действия, выполняемые в начале каждого теста, а метод **tearDown()** содержит действия, выполняемые при завершении теста.

После запуска данного скрипта вы увидите в консоли результат выполнения тестов



Результат выполнения unittest, с сайта https://docs.python.org/3/library/unittest.html#

unittest можно запускать из консоли, например, с помощью команды

**python -m unittest test\_module1 test\_module2**

При этом test\_module1 и test\_module2 должны быть python модулями, то есть директориями, содержащими файл \_\_init\_\_.py.

Опции unittest:

* -b (--buffer) - вывод программы на консоль при неудачном тесте будет показан,
* -c (--catch) - после нажатия комбинации Ctrl+C во время выполнения теста ожидается завершение текущего теста и затем сообщаются результаты на данный момент. Второе нажатие комбинации Ctrl+C вызывает обычное исключение KeyboardInterrupt,
* -f (--failfast) - выход сразу после первого неудачного теста,
* --locals (начиная с Python 3.5) - показывает локальные переменные для неуспешных тестов,

Для обнаружения тестов используется опция discover при этом тесты должны быть модулями

**python3 -m unittest discover**

Эту команду можно выполнять со следующими параметрами:

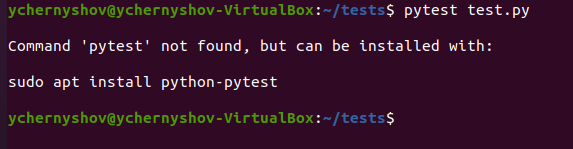
* -v (--verbose) - подробный вывод,
* -s (--start-directory) directory\_name - директория начала обнаружения тестов (текущая по умолчанию),
* -p (--pattern) pattern - шаблон названия файлов с тестами (по умолчанию test\*.py),
* -t (--top-level-directory) directory\_name - директория верхнего уровня проекта (по умолчанию равна start-directory).

Из других полезных вещей – unittest содержит функции-декораторы, например, позволяющие пропустить отдельный тест (@skip, @skipIf).

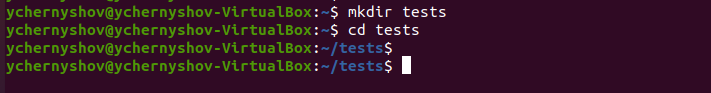
В итоге, unittest хорошо подходит для модульного тестирования, позволяет писать и выполнять тесты, поддерживает генерацию отчетов. К недостаткам, которые не соответствуют парадигме python и поэтому могут оказаться неудобными, можно отнести:

* недостаточную понятность кода в виду использования абстракций при наследовании классов,
* большое количество шаблонного кода,
* использование принятой в JUnit camel-нотации (например, CamelNotation) вместо принятой в python snake-нотации (snake\_notation).

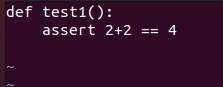
Более гибкая и функциональная альтернатива для unittest это библиотека pytest. pytest часто используется для написания тест кейсов для API. В отличие от unittest библиотека pytest не входит в стандартный набор программ, устанавливаемый вместе с python, поэтому ее необходимо устанавливать отдельно.



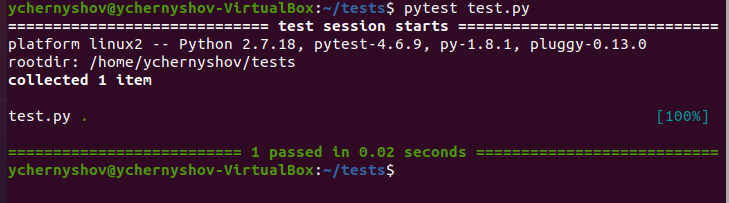
Знакомство с pytest проще всего начать с примера. Создадим папку tests



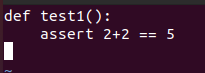
и в ней создадим файл test.py, содержащий тест. Вот так выглядит тест



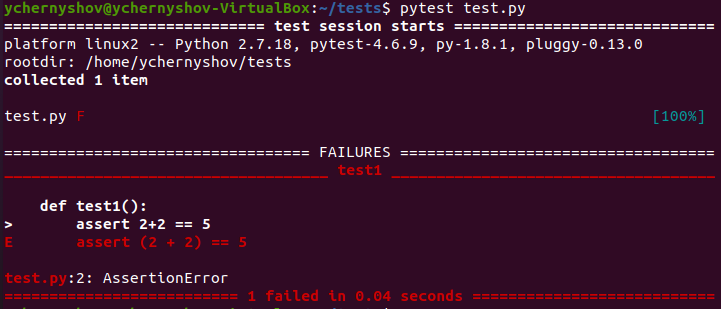
Этот тест легко запустить с помощью pytest и увидеть результаты выполнения теста



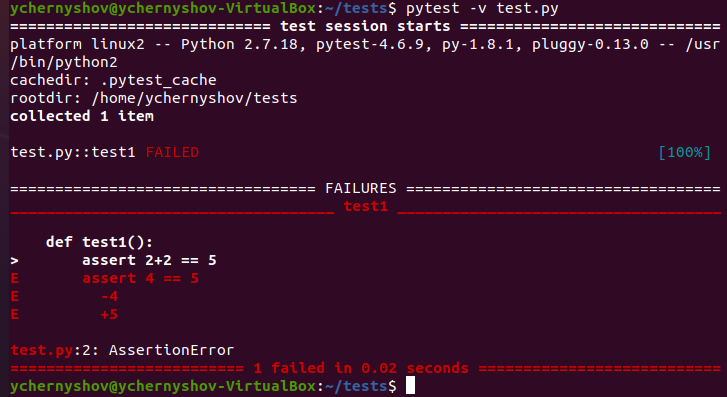
Давайте внесем в файл намеренную ошибку, чтобы увидеть как в интерфейсе pytest будет отображен неправильно выполненный тест, для этого поправим файл test.py



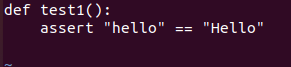
и после запуска pytest увидим результат



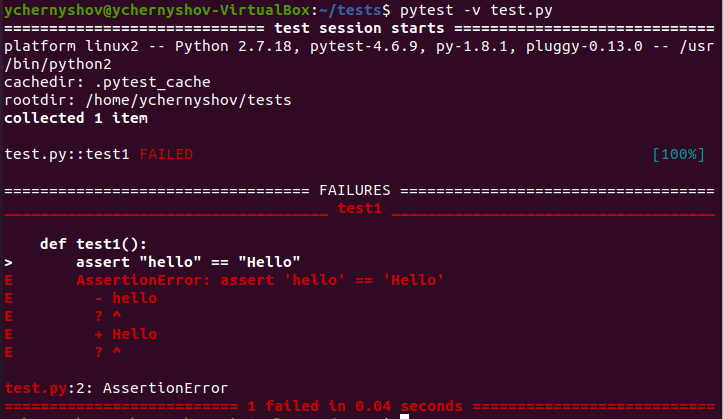
Более того, если выполнить pytest с флагом -v, то можно увидеть еще более подробное описание, указывающее на причину ошибки, например, для нашего случая будет в явном виде указано правильное значение 4 вместо неправильного 5:



А если для указания ошибки необходимо выделить позицию ошибки в тексте, то pytest использует для этого символ “^”. Например, для неправильного символа в строке



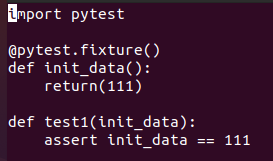
будет указана его позиция:



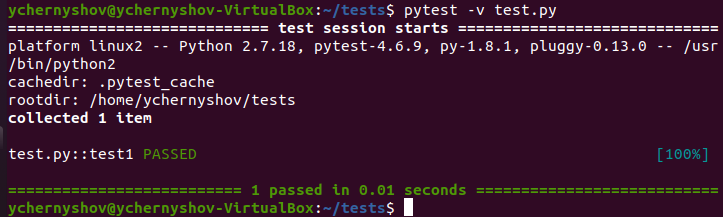
При планировании тестирования вы можете заставить pytest пропустить тест, используя функции- декораторы @pytest.mark.skip() или pytest.mark.skipif().

И, в заключение знакомства с pytest давайте познакомимся с понятием фикстур (fixture). Фикстуры необходимы для структурирования тестирующего кода, давая возможность добавлять стандартные части кода, необходимые для конкретной программной среды. Это функции, выполняемые pytest до или после тестовых функций для инициализации или завершения теста. Код в фикстуре может выполнять любые операции, например. вы можете использовать фикстуры, чтобы загрузить датасет. Это позволяет правильно инициализировать тестовое окружение, а также корректно освободить ресурсы после завершения работы.

Вот простой пример фикстуры:



И результат функции с ее выполнением



Больше опций можно получить в документации pytest на официальной странице модуля <https://docs.pytest.org/>, также удобно можно получить пользовательскую справку командой pytest -h.

*Тест*

1. От какого фреймворка унаследовал идеологию и часть функций unittest? (0.25)
   1. Java
   2. pytest
   3. **JUnit**
   4. Jenkins
2. Какие методы в unittest проверяют выполнение условия? (0.25)
   1. checkCondition
   2. **assertTrue**
   3. **assertFalse**
   4. assert
3. Что такое фикстуры в pytest? (0.25)
   1. Специальные константы
   2. **Специальные функции-декораторы**
   3. Специальные функции агрегаторы
   4. **Функции для выполнения типовых операций в начале или конце теста**
4. Какой стандартный метод в python и pytest используется для проверки утверждений? (0.25)
   1. check
   2. **assert**
   3. case
   4. statement

# Итоги/выводы

В этом юните вы познакомились с python библиотеками для автоматизации модульного тестирования unittest и pytest.

# Итоги/выводы по модулю

В этом модуле были рассмотрены задачи, связанные с тестированием в проектах машинного обучения. Тестирование играет важную роль в любом проекте разработки программного обеспечения, а машинное обучение вносит свои особенности в этот процесс.

Теперь вы знаете основные виды тестирования, например,

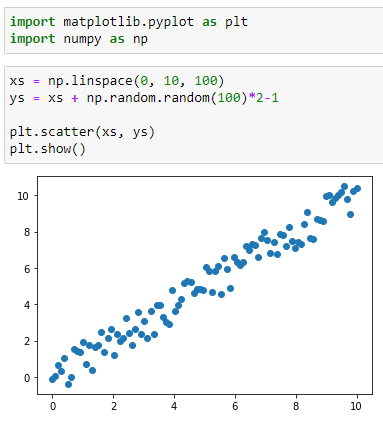
* модульное - применяется для тестирования отдельных модулей,
* интеграционное - для тестирования взаимодействия компонентов системы между собой,
* системное - для тестирования системы в целом,
* нагрузочное - для проверки работы системы в течении долгого времени под нагрузкой.

Вы узнали, что ручное тестирование является более затратным и подходит больше для нестандартных ситуаций, в которых необходимо проявить творческое мышление, и которые невозможно описать типовыми кейсами. Для тестирования больших проектов, содержащего большое количество стандартных повторяемых операций, лучше подходит автоматическое тестирование, по этой причине в проектах машинного обучения предпочтительнее применять автоматизацию тестирования.

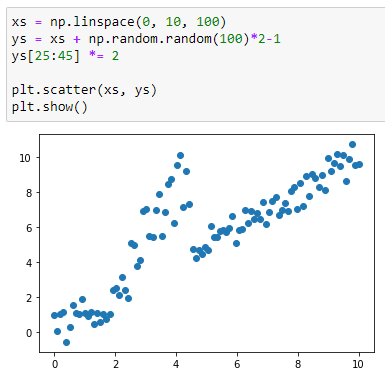
Вы знаете основные библиотеки и фреймворки для автоматизации тестирования и умеете применять две типовых python библиотеки для автоматизации тестирования: unittest и pytest.

# Практическое задание

* Цель задания: применить средства автоматизации тестирования python для автоматического тестирования качества работы модели машинного обучения на различных датасетах.
* Содержание задания:
  + Создать три датасета с «качественными» данными, на которых можно обучить простую модель линейной регрессии, например



* + На одном из этих датасетов обучить модель линейной регрессии
  + Создать датасет с шумом в данных, например



* + Провести тестирование работы модели на разных датасетах с использованием pytest, анализируя качество предсказания, обнаружить проблему на датасете с шумами.
* Критерии: данное задание необходимо полностью выполнить в виде jupyter ноутбука и предоставить его на проверку.
* Подсказка: вы можете записать содержимое ячейки jupyter ноутбука в отдельный файл с помощью команды

**%%writefile”имя файла”**

А также можете выполнить любую linux команду прямо из ячейки jupyter ноутбука, с помощью синтаксиса

**! “имя команды”**

# Список источников

Стандарты Data Management

https://www.iso.org/committee/45342/x/catalogue/

DAMA-DMBOK: Data Management Body of Knowledge: 2nd Edition

<https://www.amazon.com/DAMA-DMBOK-Data-Management-Body-Knowledge/dp/1634622340>

Описание задач и типов тестирования при непрерывном развертывании от Jira

<https://www.atlassian.com/ru/continuous-delivery/software-testing/types-of-software-testing>

Описание основ юнит тестирования

https://docs.microsoft.com/ru-ru/visualstudio/test/unit-test-basics?view=vs-2022

Обзорная статья о различных фреймворках тестирования

<https://habr.com/ru/company/otus/blog/576760/>

Официальная страница pytest

<https://docs.pytest.org/>

Официальная страница unittest

https://docs.python.org/3/library/unittest.html