Тестирование

Миша Трофимов, весна 2021

Зачем вообще нужно тестирование?

Тестирование - инструмент, который позволяет проверять выполнение наших предположений / ожиданий о работе системы или ее части.

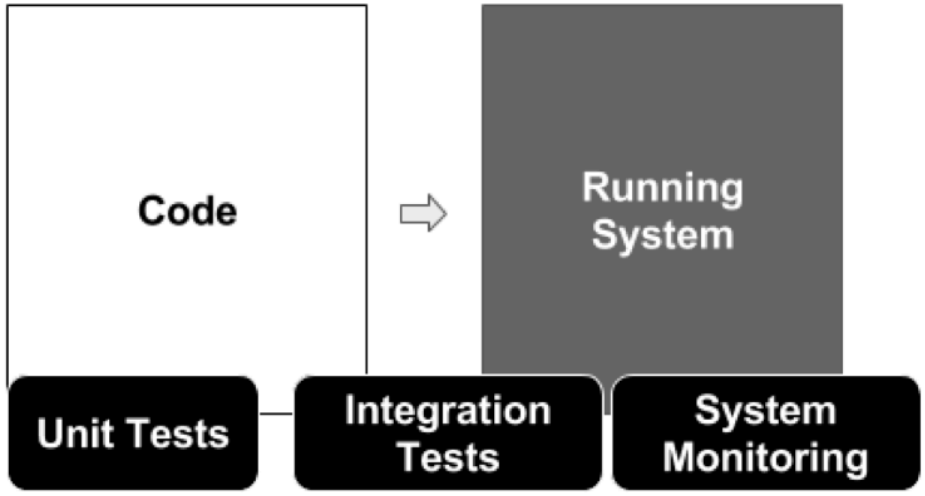
В самом простом случае - мы задаем вход и ожидаемый выход для какого-то участка системы.

Что это дает?

- Позволяет менять имплементацию существующего функционала
- Проверять, что новые изменения не ломают старую логику

Практики тестирования хорошо развиты в классической разработке, мы рассмотрим сначала их, а потом перейдем к специфике ML.

Тестирование в классической разработке



Traditional System Testing and Monitoring

Картинка из статьи *“The ML Test Score: A Rubric for ML Production Readiness and Technical Debt Reduction”*

Тесты можно грубо разделить на несколько категорий в зависимости от того, что они проверяют и на каком уровне системы применяются.

Выделяют несколько типов тестов:

- блочные / юнит / unit
- интеграционные / integration
- системные / system, end-2-end, e2e
- приемочные / acceptance

Юнит-тесты

- Юнит - тесты одного участка кода в изоляции от остальной системы.
- Есть вход, есть выход, проверяем совпадение.
- Как правило, покрывают небольшой участок кода / функцию.

Примеры: сортировка массива, на вход - [1 ,4 ,0, 2], выход - [0, 1, 2, 4]

Интеграционные тесты

- Интеграционный тест - тест того, как разные компоненты системы взаимодействуют друг с другом.
- Пример: интеграция приложения с базой данных.
    - Инициализация - очищаем тестовую базу, кладем в нее значение.
    - Запускаем часть системы
    - Проверяем, что ответ соотвествует нашим ожиданиям.

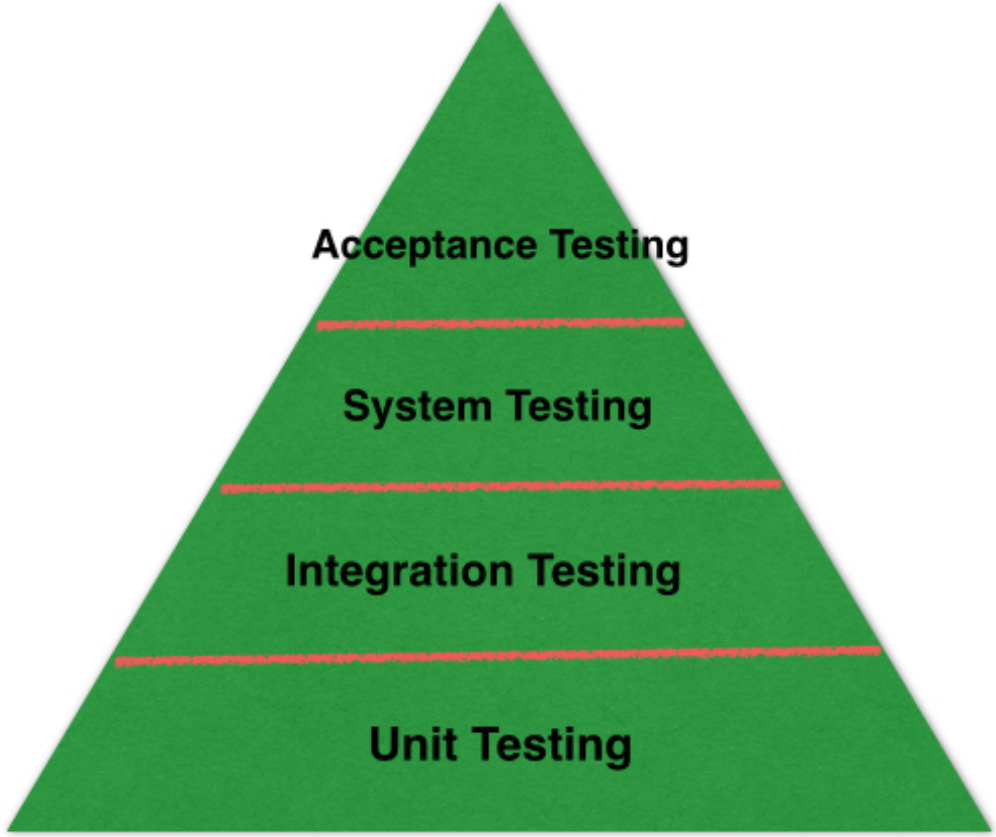
Системное тестирование

- Системное тестирование - проверяет, что система целиком работает.
- Пример
    - Развернули приложение со всеми тестовыми базами,
    - Отправили запрос на выдачу рекомендаций,
    - Проверили что ответ пришел
    - И что он *разумный*

Приемочное тестирование

- Приемочное / продакшн тестирование - проверяет, что система работает “как ожидается” в бою.
- Пример:
    - Выдерживание нагрузки и времени ответа
    - Переключение реплик
    - Плавная деградация
    - И все то, что без реальной нагрузки проверить сложно

Пирамида тестирования



Инструменты

- Для упрощения написания тестов есть разные подходы/библиотеки для разных языков программирования.
- Самыми известными инструментами для Python являются pytest и tox.

Mock-объекты

- При написании интеграционных тестов часто нужно создавать объект-заглушку
  - например, (псевдо-) базу, которая всегда возвращает один и тот же результат.
  - или (псевдо-) внешний сервис, который всегда возвращает одинаковый ответ в рамках теста.
- Такие объекты называют mock-объектами.
  - Они позволяют изолировать часть системы при интеграционном тестировании
  - и явно управлять поведением вовлеченных компонент

Ремарки

- Важны тесты всех типов
- Чем ниже тип теста в пирамиде, тем их проще писать и тем больше их требуется
- Покрыть тестами все - очень сложно
    - Метрика покрытия кода тестами - code coverage
- Хорошая практика - на любой баг заводить регрессионный тест.
    - Регрессия бага - когда починили, а он снова вернулся

Тестирование в ML

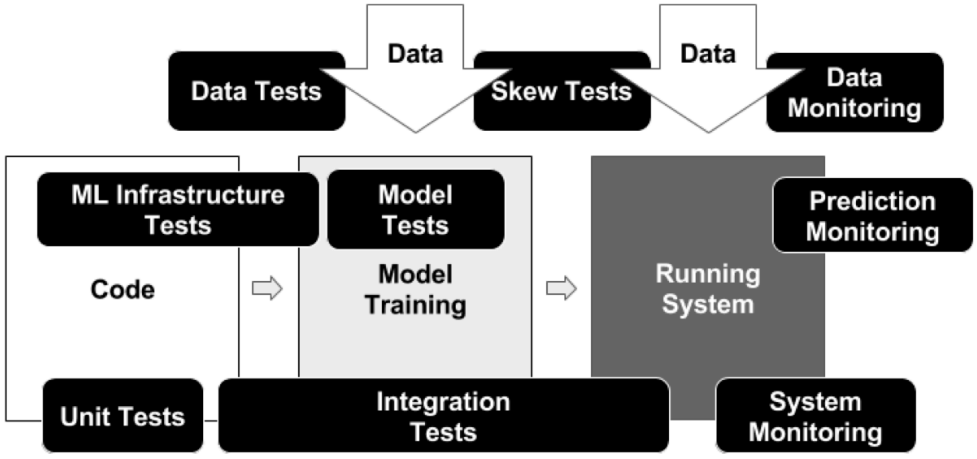
Тестирование разработок с использованием ML включает все те же приемы, что и в классической разработке, но добавляет дополнительную степень сложности.

Причина - зависимость кода от модели, а модели - от данных. Это увеличивает пространство потенциальных ошибок.

Это привело к появлению новых типов тестов:

- Тесты данных (data tests)
- Обнаружение смещения распределений (skew tests)
- Вероятностные тесты (fuzzy tests)

При этом, все уже известные подходы - особенно юнит и интеграционные тесты - остаются актуальными.



ML-Based System Testing and Monitoring

Картинка из статьи *“The ML Test Score: A Rubric for ML Production Readiness and Technical Debt Reduction”*

Тесты данных

- Цель тестов данных - убедиться, что наши ожидания от данных выполняются.
- Чаще всего это требуется на входе в систему и при обучении.

Такие тесты часто включают в себя проверку на:

- наличие необходимых полей в данных
- формат (тестовый/числовой)
- пропуски
- количество записей
- ожидаемые возможные значения (если речь о enum-like полях).

Иными словами, это валидация *схемы данных*.

Обнаружение смещения распределений

- Теория машинного обучения работает при условии, что распределение данных на обучении и на тесте совпадает.
- Проверка этого условия - и есть обнаружение смещения распределений.
- Оно позволяет отловить баги при обучении (признаки при обучении и в проде вычисляются по-разному), баги при изменении кода, “отвалившиеся” признаки и др.

Вероятностные тесты

- Тесты, которые проверяют вероятностные свойства части системы.
- Пример: хитрая функция сэмплирования коэффициента. Запускаем ее 1000 раз и проверяем среднее/мин/макс сэмпла.

Инструменты

- Инструментов много и разных, большинство можно реализовать самостоятельно с помощью базовой статистики.
- TensorFlow имеет инструментарий для тестов данных и обнаружения смещения распределений: TensorFlow Data Validation (TFDV)
- Для больших данных и Spark есть библиотека с похожим функционалом - dequ

Заключение

В реальных системах покрывать тестами стоит все хрупкие части или те, где возникают ошибки.

Набор тестов стоит собирать под конкретный продукт и требования.
“Классический набор” можно описать так:

- юнит-тесты на код расчета признаков и все внутренние преобразования данных
- интеграционные тесты на сервис с моделью
- тесты данных перед обучением
- детекция смещения между трейном и тестом
- тесты качества модели (определение устаревания)