$$S(U_1, U_2) = \frac{(|L_1 \cap L_2| + |D_1 \cap D_2| - |L_1 \cap D_2| - |L_2 \cap D_1|)}{|L_1 \cup L_2 \cup D_1 \cup D_2|}$$

В числителе вычитается количество конфликтующих симпатий и неприязнь двух пользователей из числа их общих симпатий и антипатий. Это приводит к тому, что формула индекса подобия имеет диапазон значений от -1,0 до 1,0. У двух пользователей, имеющих одинаковые вкусы, индекс сходства будет 1,0, тогда как у двух пользователей, имеющих полностью противоречивые вкусы в фильмах, будет показатель сходства -1,0.

После того, как будут получены индексы подобия пользователей, можно найти вероятность того, что пользователю U понравится элемент M:

$$P(U, M) = \frac{Z_L - Z_D}{|M_L| + |M_D|}$$

Z_L и Z_D представляют собой сумму индексов подобия пользователя U со всеми пользователями, которым понравился или не понравился элемент M. | M_L | + | M_D | - общее количество пользователей, которым понравился или не понравился элемент M. Результатвычисления P (U, M) дает число от -1,0 до 1,0.

Данная формула является механизмом рекомендаций. Алгоритм прост и понятен, а также его можно улучшать и расширять под собственную систему. На данной фундаментальной идее строятся многие рекомендательные библиотеки, позволяющие работать не только с маленькими выборками, но и большими базами данных, что делает данный механизм очень популярным в современных быстрорастущих системах.

Список использованных источников:

1. Розенберг Г. С. Поль Жаккар и сходство экологических объектов, М.: ИэВбРАН, 2012, 202 с.
2. Джанак Д., Занкер М., Фелферниг А., Фридрих Г. Введение в рекомендательные системы, М.: Издательство Кембриджского университета, 2010, 775 с.

АВТОМАТИЗАЦИЯТЕСТИРОВАНИЯ ПОЛЬЗОВАТЕЛЬСКОГО ИНТЕРФЕЙСА ДЕСКТОППРИЛОЖЕНИЙСИСПОЛЬЗОВАНИЕМ TITANIUM

Белорусский государственный университет информатики и радиоэлектроники
г. Минск, Республика Беларусь

Цегельник Н.Г.

Поттосина С. А. – канд. физ.-мат. наук, доцент

Сфера информационных технологий приобрела особое значение в современном мире. Быстрое развитие систем разработки программного обеспечения и сетевых технологий привело к увеличению производства на рынке программного обеспечения. Усиление конкуренции между производителями программного обеспечения требует повышенного внимания к качеству продукции. Большое количество компаний по всему миру начали инвестировать в повышение качества программного обеспечения. Таким образом, тестирование стало неотъемлемой частью процесса создания программного обеспечения. Тестирование позволяет определить, соответствует ли программный продукт предъявляемым к нему требованиям, выявить ситуации, в которых поведение программы является неправильным, нежелательным или не соответствующим спецификации. Своевременное выявление и исправление ошибок и недоработок имеет огромное значение в процессе разработки программного продукта, поскольку это уменьшает риски и при этом происходит снижение затрат на разработку программного обеспечения.

Анализируя рациональность введения автоматизированного тестирования в рамки проекта, необходимо оценить: степень сложности выполнения тестовых сценариев вручную, частоту запуска написанных тестов, необходимость в увеличении скорости прохождения тестов. Если все же решено автоматизировать тестовые сценарии, то необходимо ли это выполнять для всего набора тестов, или достаточно сделать частичную автоматизацию. Внедрение автоматизированного тестирования позволяет решить такие проблемы, как снижение времени, необходимого для выполнения тестирования, а также повышение качества процесса тестирования.

Автоматизация тестирования позволяет ускорить процесс тестирования, в особенности такие аспекты:

- часто используемую функциональность, проверку критической функциональности, где риски возникновения ошибок достаточно высоки;
- рутинные операции (проверка форм, в которых необходимо заполнять огромное количество полей);
- сложно доступные места в системе, такие как запись данных в базу, бекэнд процессы;
- проверка корректного поиска данных;

– проверки с использованием математических расчетов, длинные end-to-end сценарии.

В данной работе рассматривается тестирование десктоп приложений. Desktop приложения представляют собой программы, устанавливаемые на отдельных рабочих станциях или персональных компьютерах. Они не требуют для своей работы доступа к интернету. Взаимодействие с пользователем осуществляется посредством стандартного интерфейса. Особенностью данного типа приложений является зависимость от операционной системы, прямой доступ в локальной файловой системе, а также необходимость предварительной установки на каждый компьютер. В отличие от веб-приложений, к ним можно получить доступ только на том компьютере, на котором они установлены.

При тестировании необходимо обращать внимание на:

– тестирование установки;

– тестирование установки обновлений;

– тестирование деинсталляции;

– тестирование совместимости на разных ОС.

В рамках разрабатываемого проекта для построения полного и непрерывного процесса тестирования организованы следующие виды тестирования:

– Юзабилити тестирование – проверка работы с точки зрения пользователя.

– Регрессионное тестирование – непрерывная проверка существующих тестовых сценариев.

– Конфигурационное тестирование и совместимости – проверка работы программы при различных конфигурациях (различные версии продукта, разные конфигурации компьютера).

– Smoke тестирование – проверка базового функционала программы.

Стоит сказать, что разрабатываемый тестовый фреймворк нацелен на работу через визуальную форму (интерфейс пользователя).

В настоящее время большинство масштабных проектов по разработке программного обеспечения в процессе проверки качества использует совместно функциональное и автоматизированное тестирование. Кроме того, создается собственный инструментарий TAF (TestAutomationFramework), который позволяет одновременно тестировать несколько решений, построенных на различных технологиях.

При разработке тестового фреймворка была выбрана методология DDT (тестирование, управляемое данными).

Данная методология позволяется выстраивать процесс тестирования таким образом, что выполнение и верификация тестовых сценариев производится на основе данных, хранящихся в базе данных или иных источниках данных. Как правило сравнение происходит между эталонными данными и информацией, получаемой системой на выходе из метода (функции, программы). Тестирование, управляемое данными, предполагает наличие разделения тестов и проверяемых в них данных. Тестовые методы получают эталонные данные из некого источника и сравнивают их с результатами, полученными при тестировании объекта. Преимуществом DataDrivenTesting является простота добавления дополнительных входных значений в таблицу при обнаружении или добавлении новых разделов в тестируемый продукт или систему.

Для хранения тестовых сценариев, а также входных значений для них использован MTM (MicrosoftTestManager).

В MTM создается текущий проект, который содержит план тестирования. В рамках одного проекта может быть несколько планов тестирования. По этой причине разнообразные тестовые сценарии, тест-кейсы сохраняются в общее хранилище TFS (TeamFoundationServer), а планы тестирования позволяют осуществлять организованную работу с ними. Автоматические тест-кейсы связываются определенными идентификаторами с тестовыми методами.

Для написания тестовых сценариев использована технология под названием Titanium и язык программирования C#. Этот тестовый фреймворк является оберткой CodedUITest.

CodedUI– это решение для автоматизации тестирования интерфейса. Позволяет автоматизировать как web-приложения, так и desktop-приложения.Тестирование с использованием CodedUI подразумевает запись последовательности действий с помощью специального инструмента CodedUITestBuilder и дальнейшую проверку результатов. Однако этот инструмент имеет ряд недостатков, которые были решены в тестовом фреймворке Titanium. При использовании инструмента записи/воспроизведения генерируется слишком много кода, сложно поддающегося корректировке, высокая повторяемость кода, необходимость писать много кода даже для простейших команд.

Структура программы при использовании Titanium сформирована по типу паттерна PageObject, который является одним из наиболее полезных архитектурных решений в автоматизации. Данный шаблон проектирования помогает инкапсулировать работу с отдельными элементами страницы, что позволяет уменьшить количество кода и его поддержку. Допустим, если разметка и дизайн страницы был изменен, то изменению будет подлежать только соответствующий класс, описывающий эту страницу. Такой подход значительно уменьшает объем повторяющегося кода, потому что одни и те же объекты страниц можно использовать в различных тестах. Основное преимущество PageObject заключается в том, что в случае изменения пользовательского интерфейса, можно выполнить исправление только в одном месте, а не исправлять каждый тест, в котором этот интерфейс используется.

Таким образом, использование Titanium для автоматизации тестовых скриптов позволяет минимизировать количество кода, организовать четкую структуру тестового фреймворка, тем самым упростить задачу написания тестовых сценариев, повысить эффективность работы автоматизированных тестировщиков.

Список использованных источников:

1. Котляров, В.П. Основы тестирования программного обеспечения. – М.: Интернет-Университет Информационных

Технологий. Лаборатория знаний, 2006. – 285 с
2. Канер, С. Тестирование программного обеспечения. Фундаментальные концепции менеджмента бизнес- приложений// 2005. – №4. – с47-52
3. UIAutomationFundamentals [Электронный ресурс]. – Режим доступа: https://msdn.microsoft.com/en-us/library/ms753107(v=vs.110).aspx
4. Тестирование, управляемое данными [Электронный ресурс]. – Режим доступа: http://abap-blog.ru/osnovy-abap/testirovanie-upravlyaemoe-dannymi-data-driven-testing/

МОДЕЛЬ SKIP-GRAM ТЕХНОЛОГИИ WORD2VEC

Белорусский государственный университет информатики и радиоэлектроники
г. Минск, Республика Беларусь

Давыдовский С.В.

Сиротко С.И. – канд. физ.-мат. наук, доцент

Технология word2vec с момента своего появления в 2013 году [1] обрела большую популярность среди исследователей в области машинного обучения. Векторные представления слов, получаемые с помощью word2vec, нашли широкое применение в различных областях обработки естественного языка. Такое распространение обусловлено тем, что векторы word2vec довольно точно передают семантические значения слов. В данной работе рассматривается принцип работы одной из моделей word2vec – Skip-Gram.

Основная идея word2vec – слова, находящиеся в похожих контекстах, являются семантически близкими. Для вычисления векторного представления слов используется искусственная нейронная сеть. Во время обучения сети с помощью модели Skip-Gram формируются оптимальные векторы для каждого слова.

Модель Skip-Gram позволяет предсказывать близлежащие слова на основании центрального слова. Задача нейронной сети заключается в следующем: для заданного слова вычислить у всех остальных слов в словаре вероятности их появления рядом с этим словом. Под словом “рядом” понимается фиксированный размер окна, к примеру 5 слов слева и 5 слов справа.

Для обучения нейронной сети используются большие наборы текстов. В качестве тренировочных примеров используются пары “центральное слово – контекстное слово”. На рисунке 1 показаны примеры тренировочных данных полученные из предложения “Thequickbrownfoxjumpsoverthelazydog.” при размере окна, равном двум (центральное слово выделено синим) [2].

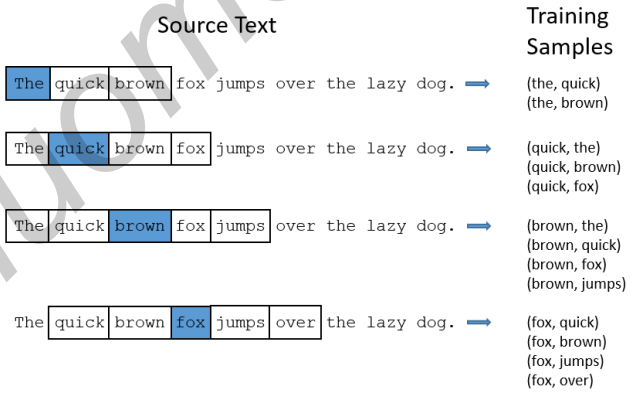

Рис. 1 – Примеры тренировочных данных

Предположим, что в наборе текстов находится 10000 уникальных слов. Каждое слово изначально представлено как one-hot вектор $I \in R^{10000}$. Выходной слой нейронной сети формирует вектор $O \in R^{10000}$, содержащий вероятности появления каждого из 10000 слов рядом с входным словом. Архитектура нейронной сети представлена на рисунке 2.