В ходе первичных экспериментов было показано, что при помощи описанных сценариев можно осуществлять как извлечение уникальных характеристик каждой схемы ДОЗУ, так и производить генерацию истинно случайных последовательностей.

Список использованных источников:

1. Lao, Y. Reliable PUF-Based Local Authentication With Self-Correction / Y. Lao, B. Yuan, C. H. Kim, K. K. Parhi // IEEE transactions on Computer-Aided Design of Integrated Circuits and Systems. –2017. – Vol. 36, № 2. –P. 201-213.
2. Tehranipoor, F. DRAM based Intrinsic Physical Unclonable Functions for System Level Security / F. Tehranipoor, N. Karimian, K. Xiao, J. Chandy // Proc. of the 25th edition on Great Lakes Symposium on VLSI. – Pittsburgh, Pennsylvania, USA, 2015. – P. 15-20.

КЛАССИФИКАЦИЯ ТОНАЛЬНОСТИ ТЕКСТОВЫХ ДОКУМЕНТОВ С ПОМОЩЬЮ МЕТОДА ОПОРНЫХ ВЕКТОРОВ

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

Романов А.А.
Пилецкий И. И. – канд. физ.-мат. наук, доцент

С развитием сети Интернет, объём информации, создаваемой человечеством, значительно увеличился. Информация накапливается в различных источниках, таких как социальные сети, форумы, площадки для отзывов, блоги и новостные сайты, причём преимущественно она хранится в виде текстовых данных. Большой объём и слабая структурированность таких данных определяют необходимость создания систем автоматической обработки. Одной из актуальных задач обработки, является определение тональности текстовых документов. Определение тональности текстов востребовано, например, при анализе эффективности рекламных кампаний, сборе откликов о проведённых мероприятиях, определении репутации брендов, построении системы поддержки пользователей.

Текст на естественном языке может нести в себе не только информацию, но и её эмоциональную оценку. Эмоциональная оценка, выраженная в текстовом документе, называется тональностью или сентиментом (англ. sentiment – чувство, настроение). В понятии машинного обучения, задача определения эмоциональной оценки текста сводится к задаче классификации. В формальном виде задача классификации определяется следующим образом [1]. Пусть существует описание документа $d \in X$, где X – векторное пространство документов, и конечное множество классов $C = \{c_1, c_2, \dots, c_j\}$. Из множества документов с заранее известными классами $D = \{(d, c)$, где $(d, c) \in X \times C\}$, используя обучающий алгоритм, необходимо получить классифицирующую функцию γ, которая отображает документы в классы $\gamma: X \rightarrow C$. В решаемой задаче определения тональности множество C состоит из двух элементов: положительной и отрицательной эмоциональной оценки.

Задачу классификации на два класса успешно решают с использованием различных методов машинного обучения. Для применения алгоритмов машинного обучения текстовый документ необходимо представить в виде математического вектора. В качестве векторной модели используется «мешок термов» [1]. Текст в данной модели рассматривается как неупорядоченное множество термов. Термом может являться любое символьное выражение текста, например, слова, словосочетания, знаки пунктуации. Каждому терму сопоставляется некий вес. Вектор же формируется при упорядочивании всех уникальных термов в пространстве. Размерность вектора определяется числом уникальных термов во всей коллекции и является постоянной для всех документов.

Перед взвешиванием документов проводится предварительная очистка коллекции: приведение всех символов текстовых документов к нижнему регистру и удаление пунктуационных знаков. Далее из коллекции извлекаются уникальные термы, для каждого из которых рассчитываются статистические данные, необходимые для построения весовых схем. При необходимости, для повышения результатов классификации настраивается система фильтров, используя извлеченную статистику. Для взвешивания опробованы различные весовые схемы: бинарная, TF, TF-IDF, TF-RF и др. [2-5].

Для решения задачи был выбран один из алгоритмов машинного обучения с учителем – метод опорных векторов (supportvectormachine, SVM). Выбор данного алгоритма основан на его высокой точности в решении задач бинарной классификации коллекций текстовых документов различных тематик [6]. Основная идея метода SVM – поиск разделяющей гиперплоскости, максимально удалённой от ближайших к ней точек обоих классов в пространстве признаков [7]. В качестве ядра SVM взято линейное ядро, как наиболее эффективное при больших размерностях векторов и большом количестве объектов для обучения [8].

В качестве оценок качества результатов обучения и работы алгоритма выбраны четыре общепринятые характеристики: accuracy, precision, recall и f-measure. Для несмещенной оценки вероятности ошибки и избегания проблемы переобучения используется кросс-валидация по 5 блокам.

В процессе исследования тестируются различные коллекции документов на английском и русском языках. Коллекции содержат сотни тысяч документов и сопоставимое количество уникальных термов. Для эффективной обработки такого объёма информации был выбран фреймворк ApacheSpark. Библиотека MLlib данного фреймворка поддерживает реализацию метода опорных векторов с линейным ядром [9]. Используя данную библиотеку, на всех тестируемых коллекциях удалось достичь показателей качества более 80%.

Список использованных источников:

1. Manning D., Raghavan P., Schutze H. Introduction to Information Retrieval. Cambridge University Press. 2008.
2. Lan M., Tan C.L., Su J., Lu Y. Supervised and Traditional Term Weighting Methods for Automatic Text Categorization. IEEE Transactions on Pattern Analysis and Machine Intelligence. vol. 31. pp. 721–735. 2009.
3. Wang D., Zhang H. Inverse-category-frequency based supervised term weighting scheme for text categorization. 2010.
4. Mori, T. Information gain ratio as term weight: the case of summarization of ir results. In Proceedings of the 19th International Conference on Computational Linguistics, 688–694. Association for Computational Linguistics Publisher. 2002.
5. Lan, M.; Sung, S.-Y.; Low, H.-B.; and Tan, C.-L. A comparative study on term weighting schemes for text categorization. In Proceedings of the International Joint Conference on Neural Networks 2005. 2005
6. Рубцова Ю. В. Разработка и исследование предметно независимого классификатора текстов по тональности. Труды СПИИРАН. – 2014. – Т. 5. – №. 36. – С. 59-77.
7. Воронцов К. В. Лекции по методу опорных векторов. 2007.
8. Chih-Wei Hsu, Chih-Chung Chang, and Chih-Jen Lin. A Practical Guide to Support Vector Classification. 2016.
9. Spark MLlib linear methods. https://spark.apache.org/docs/latest/mllib-linear-methods.html

КЕШИРОВАНИЕ КАК СПОСОБ ОПТИМИЗАЦИИ ROR ПРИЛОЖЕНИЙ

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

Свито А.И., Пырлог Е.С., Дорошкевич П.Е.

Стержанов М.В. – канд. техн. наук, доцент

Ruby On Rails (RoR) представляет собой фреймворк для быстрой разработки динамических Web-приложений, написанный на языке Ruby. Платформа обладает огромным списком достоинств, таких как обширное комьюнити, самая «чистая» реализация паттерна MVC и огромное количество готовых инструментов для тестирования, локализации и обеспечения безопасности приложения. Благодаря чему при написании на RoR достигается высокая скорость разработки поддерживаемых и масштабируемых приложений. Однако одним из недостатков данной платформы является её низкая производительность. Нами рассматривается кэширование как один из способов оптимизации производительности RoR приложений.

Наиболее часто используемым способом оптимизации приложения является кэширование, суть которого заключается в следующем:

Зачастую приложению приходится решать по запросу пользователя одни и те же задачи. Если для решения этой задачи требуется большое количество времени, то многократное её выполнение может значительно снизить производительность приложения. Чтобы избежать решения одной и той же задачи много раз используют кэширование. Вместо того чтобы многократно выполнить процедуру, она выполняет всего один раз, но после сохраняет результат своего выполнения. Таким образом, при необходимости повторного решения одной и тоже задачи нам достаточно просто передать тот результат, который был сохранен при первом запросе решения. В Ruby on Rails наиболее активно используются четыре вида кэширования: кэширование объектов, страниц, действий и фрагментов. Рассмотрим подробнее каждый из них:

1. *Кэширование объектов*. Самым простым в понимании является кэширование объектов, которое заключается в выделении в приложении наиболее часто используемых объектов и помещении их во внешнее хранилище, доступ к которому осуществляется быстрее чем получение и создание объекта внутри приложения. Недостаток данного подхода в том, что не всегда удаётся выделить часто используемые сущности в приложении. RubyOnRails является объектно-ориентированным языком и зачастую вся работа приложения заключается в извлечении из базы данных коллекции объектов, к котором пользователь имеет неограниченный доступ. Ввиду этого сложно предугадать объекты, с которыми приложению придётся оперировать чаще всего.

2. *Кэширование страниц*. Кэширование страниц – наиболее эффективная форма кэширования данных, имеющаяся в Rails. Когда пользователь посылает некоторый запрос по конкретному URL, приложение после получения данных и выполнения необходимых действий создает HTML-страницу. Код разметки этой страницы сохраняется в кэше. При необходимости повторной обработки запроса по тому же URL, страница не создаётся заново, а просто извлекается прямо из хэша. При этом фактически приложение не вступает в работу, всю работу в последнем случае выполняет веб-сервер. Таким образом, обработка подобных запросов занимает минимально возможное время, которое зависит исключительно от скорости работы веб-сервера. Однако такой подход, несмотря на простоту, также не лишён недостатков. Поскольку обработка запросов в данном случае происходит без участия приложения, мы не можем обработать один и тот же запрос по-разному, в зависимости от, например, роли пользователя, производящего запрос. Также, из-за того что приложение не включено в обработку запросов, фактически, доступ к содержимому страницы ничем не ограничен.

3. *Кэширование действий*. Для того, чтобы иметь возможность кэширования страниц, доступ к которым имеют лишь ограниченные категории привилегированных пользователей, используется кэширование действий. Как и в предыдущем случае, его целью является кэширование содержимого всей страницы целиком, однако данный подход дает возможность фильтрации пользователей, которым доступен просмотр страницы.