Dao Duy Nam, S. A. Ivanovskiy
Saint-Petersburg state electrotechnical university «LETI»

OPTIMIZATION ALGORITHM LOCALIZATION MOBILE ROBOT
WITH USING TRIANGULATION MAP

We consider approximation algorithm for the robot localization problem. The algorithm is based on triangulation of a simple polygon representing a map. On the basis of their program implementation conducted experimental studies of this algorithm. The numerical results and their interpretation are given.

Computational geometry, robotics, robot localization, polygon triangulation, overlay polygon, approximation algorithm, algorithm complexity

УДК 004.82, 004.89

А. В. Малов
Motorola Solutions

С. В. Родионов
Санкт-Петербургский государственный электротехнический университет «ЛЭТИ» им. В. И. Ульянова (Ленина)

Реализация упрощенного алгоритма Байеса в среде функционального программирования COMMON LISP

Рассматриваются основные задачи и особенности интеллектуального анализа данных. Иллюстрируются преимущества применения среды и языка функционального программирования COMMON LISP для решения типовых задач интеллектуального анализа данных на примере реализации упрощенного алгоритма Байеса.

Интеллектуальный анализ данных, функциональное программирование, упрощенный алгоритм Байеса

В настоящее время наблюдается интенсивное развитие информационных технологий. В частности, активно развиваются технологии искусственного интеллекта, в том числе интеллектуальный анализ данных (Data mining). Он является одним из современных подходов к анализу больших массивов данных и предназначен для выявления скрытых нетривиальных знаний, которые позволят сформулировать практически полезные гипотезы.

Для того чтобы сформулированные гипотезы не носили случайного характера и были характерными для заданной предметной области, они должны быть логически объяснимы и представлены в виде, доступном для дальнейшего анализа и использования. Такие представления называются моделями. Модель, представляющая собой формальное описание полученных из данных знаний, является результатом работы методов и алгоритмов Data mining и отражает вид представления полученных знаний.

Широко распространен подход к интеллектуальному анализу данных посредством первоначального построения или выбора модели с последующим ее применением для выполнения функций Data mining, таких, как классификация, аппроксимация, кластеризация, ассоциирование и построение последовательностей, определение атрибутов, прогнозирование временных рядов, определение аномалий [1]. При этом имеется

определенная последовательность построения модели и ее последующего использования, включающая следующие этапы [2]:

1) подготовка данных (конвертация физических данных к виду, удобному для применения методов data mining, и формирование логического представления);

2) собственно построение модели (задание ее параметров, настройка алгоритмов извлечения знаний, их накопления и интеграции);

3) анализ модели (проверка извлекаемых знаний на релевантность и настройка обмена знаниями);

4) применение полученных знаний вида модели отношений стимул–реакция для прогнозирования на базе новых данных.

Достаточной емкостью и гибкостью для реализации этапа 2 обладает множество языков программирования. Если необходимо реализовать программу для экспериментов, не требующую наличия излишеств наподобие графического интерфейса, а также работающую подчас с неизвестным заранее объемом данных, то для этой цели отлично подходит язык императивного программирования Java, в котором есть библиотеки с множеством полезных для надежной работы алгоритма классов. Приведем несколько примеров реализаций интеллектуального анализа данных на императивных языках.

Библиотека RapidMiner [3] является мощной свободно распространяемой аналитической платформой для сбора данных, машинного обучения и интеллектуального анализа. Анализ процессов осуществляется посредством перетаскивания операторов, настройки их параметров, дальнейшего их объединения и представления процессов в графическом виде. Структура процесса описывается внутри на языке XML и разрабатывается в среде GUI. Комбинации успешных операторов можно объединять в блоки с возможностью дальнейшего их использования в новых процессах. Библиотека содержит более 1500 операторов для задач профессионального анализа данных (маркетинговый анализ, методы text mining, web mining, анализ временных рядов и т. п.).

Библиотека Waikato Environment for Knowledge Analysis (Weka) [4] содержит коллекцию алгоритмов машинного обучения для задач интеллектуального анализа данных. Алгоритмы могут либо применяться непосредственно, либо быть встроены в Java-код. Библиотека содержит инструменты для функций классификации, регрессии, кластеризации, построения правил ассоциации и визуализации. Она может применяться для разработки новых схем машинного обучения.

Библиотека R [5] представляет собой системы для статистической графической обработки данных. Она содержит лингвистические средства языка R, среду выполнения кода с встроенной графикой, средства отладки, средства доступа к ряду системных функций и запуска программ с использованием файлов сценариев. Интерпретатор языка R позволяет использовать ветвления и циклы, а также модульное программирование с помощью функций. Возможна интеграция с модулями на языках *C*, *C++* или Fortran для повышения эффективности реализации. Библиотека может использоваться для анализа данных на основе линейных моделей, регрессии, временных рядов, кластеризации. Существует большой набор функций с графической средой для создания различных видов представления данных.

В работе [6] рассматривается реализация алгоритма выявления образцов на языке функционального программирования Haskell. В статье указывается, что встроенная поддержка функциональности сопоставления с образцом является сильной стороной языка Haskell. По мнению авторов, данная особенность языка не только способствует удобству разработки программ, но и должна дать выигрыш в быстродействии по сравнению с императивными языками, чаще используемыми при реализации данных алгоритмов. В работе сравнивается быстродействие реализаций алгоритма на языке императивного программирования Java и языке функционального программирования Haskell. Экспериментальные данные, приведенные в работе, показывают преимущество в быстродействии реализации на языке Haskell. Авторы исследования не приводят детали реализации алгоритма на языке Java. Следует отметить, что не существует причин, на основании которых можно было бы утверждать, что на любом императивном языке можно реализовать только менее эффективную функциональность сопоставления с образцом, по сравнению с реализацией данной функциональности в языке Haskell. Таким образом, можно сделать вывод, что данные результаты являются следствием неоптимальной реали-

зации на языке Java исследуемого алгоритма, а также могут наблюдаться по причине использования Java-платформы, так как это потенциально может снижать быстродействие.

Обзор существующих научных работ показывает, что область применения функционального программирования в рамках интеллектуального анализа данных не ограничивается алгоритмами выявления образцов. Так в работе [7] рассматривается применение языка функционального программирования Haskell при решении задач интеллектуального анализа данных в области биоинформатики. Авторы отмечают достоинства языка Haskell, присущие функциональным языкам. В число достоинств авторы включают ясность и удобочитаемость программного кода, простоту отладки, которая объясняется тем, что не имеющие побочных эффектов функции можно отлаживать независимо от остального кода, так как результат их работы всегда зависит только от входных параметров. Данные свойства позволяют снизить частоту появления ошибок в коде. Авторы работы обращают внимание на то, что стоящие перед ними задачи предъявляют высокие требования к корректности программного кода, так как полученные результаты могут быть значимыми, а ошибки, допущенные при обработке большого объема данных, приводят к большим потерям времени. Данные особенности повышают значимость достоинств функционального подхода. К числу недостатков языка Haskell авторы относят большое потребление памяти во время работы программы. Следует отметить, что разработанное авторами программное обеспечение имеет возможность работать в параллельном режиме на кластерах Beowulf.

В настоящей статье иллюстрируются преимущества применения среды и языка функционального программирования COMMON LISP для решения типовых задач интеллектуального анализа данных. Среди достоинств данного языка программирования можно выделить свойства, присущие языкам функционального программирования. К ним относятся возможность организации автоматического параллельного выполнения, отсутствие потенциальных ошибок, связанных с переменными и присваиванием им новых значений, удобство отладки и разработки тестов.

Как будет проиллюстрировано в статье далее, при решении задач интеллектуального анализа данных в виде списков удобно представлять и обрабатывать различные массивы данных. Для этих целей подходит встроенная в язык COMMON LISP поддержка списков и функционал для работы с ними. В частности, имеющийся функционал для работы со списками позволяет применять ко всем элементам списка различные функции, передаваемые как параметры в виде указателей. Данные возможности позволяют создавать для алгоритмов интеллектуального анализа данных решения, имеющие универсальный характер.

Упрощенный алгоритм Байеса представляет собой алгоритм классификации. Данный алгоритм позволяет достаточно эффективно сформировать модели интеллектуального анализа данных для определения отношений между входными и прогнозируемыми атрибутами.

Входные данные для интеллектуального анализа данных с помощью упрощенного алгоритма Байеса, или базу данных, можно представить в виде таблицы, каждая строка которой содержит значения входных атрибутов и значение прогнозируемого атрибута. В качестве примера базы данных в табл. 1 представлены данные клиентов страховой компании, на основании которых с помощью упрощенного алгоритм Байеса можно сформировать модель и предсказать вероятность страхового случая или его отсутствия у клиента в новом страховом периоде. Первая строка таблицы содержит названия соответствующих атрибутов, последний столбец первой строки – название прогнозируемого атрибута.

Модель интеллектуального анализа данных для упрощенного алгоритма Байеса можно представить в виде таблицы, содержащей рассчитанную вероятность состояния каждого входного

Таблица 1

Пол	Состоит в браке	Наличие детей	Частота страховых случаев	Возраст	Стаж вождения, лет	Наличие страхового случая
Ж	Да	Нет	Низкая	18	0	Нет
М	Да	Да	Низкая	35	13	Нет
М	Нет	Нет	Высокая	23	4	Да
…	…	…	…	…	…	…

атрибута при всех возможных значениях прогнозируемого атрибута.

Здесь может быть использован стандарт PMML (Predicted Model Markup Language) версии 2.0, предназначенный для обмена построенными mining-моделями между системами Data mining с использованием языка представления моделей в виде XML-документа. В стандарте PMML поддерживается тип модели NaiveBayesModel, являющийся набором матриц, каждая из которых содержит частоту (или вероятность) значений независимого атрибута относительно значений зависимого атрибута.

В рассматриваемом примере модель для упрощенного алгоритма Байеса можно представить в виде табл. 2. При этом каждая ячейка на пересечении строки, соответствующей возможному значению каждого входного атрибута, и столбца, соответствующего возможному значению прогнозируемого атрибута, должна содержать вероятность того, что значение входного атрибута соответствует строке при условии, что значение прогнозируемого атрибута соответствует столбцу. В примере для простоты вместо конкретных значений вероятностей в ячейках помещены троеточия. Данные значения вычисляются в процессе работы упрощенного алгоритма Байеса.

Рассматриваемая в настоящей статье реализация упрощенного алгоритма Байеса охватывает не все рассмотренные ранее этапы из последовательности формирования модели интеллектуального анализа данных и последующего ее применения. Цель разработки рассматриваемой реализации – обосновать выбор методов и средств на этапах построения модели в части создания и настройки алгоритмов извлечения знаний и их применения (фрагменты этапов 2 и 4). С целью упрощения рассматриваемая реализация извлекает только те знания, которые необходимы, чтобы определить значения прогнозируемого атрибута для новых данных, поступающих на вход в виде вектора входных атрибутов. Таким образом, второй этап в реализованной модели подразумевает, что модель содержит лишь значения, необходимые для вычисления конкретного входного вектора. При этом после определения прогнозируемых значений полученная модель не сохраняется.

Для реализации упрощенного алгоритма Байеса, рассматриваемой в настоящей статье, был выбран язык среды функционального программирования COMMON LISP. К числу достоинств данного языка, присущих и другим языкам функционального программирования, можно отнести:

– возможность организации автоматического параллельного выполнения. Параллельное выполнение программ в императивных языках трудно реализовать, так как ответственность за организацию параллельных вычислений возлагается на программиста. Поскольку результат работы функций при функциональном программировании зависит только от входных параметров, порядок вычисления функций не влияет на результат. Данные обстоятельства открывают широкие перспективы для автоматического параллельного выполнения;

– отсутствие потенциальных ошибок, связанных с переменными и присвоением им новых значений. На результат работы функций влияют только входные параметры, что исключает появление ошибок, которые трудно обнаружить. Подобные ошибки могут быть связаны, например, с присвоением значений одной и той же переменной в различных участках кода;

– удобство отладки и разработки тестов. На результат работы функций влияют только вход-

Таблица 2

Название входного атрибута	Значение атрибута	Наличие страхового случая в новом страховом периоде	
		Нет	Да
Пол	М	...	...
	Ж	...	...
Состоит в браке	Да	...	...
	Нет	...	...
Наличие детей	Да	...	...
	Нет	...	...
Частота страховых случаев	Низкая	...	...
	Средняя	...	...
	Высокая	...	...
...	...	...	...

ные параметры, поэтому для тестирования функций достаточно проверить их значения при различных входных параметрах.

Из недостатков языка COMMON LISP, а также функционального подхода в целом можно выделить:

– отсутствие циклов. При чистом функциональном подходе для организации повторений допустима только рекурсия;

– требуется сборщик мусора. Отсутствие присвоений приводит к необходимости их замены на создание новых данных, что, в свою очередь, требует наличия функционала автоматического выделения и освобождения памяти, т. е. обязательным компонентом становится сборщик мусора.

Данные в языке COMMON LISP, как и программный код, представляют собой списки. Как уже отмечалось, при решении задач интеллектуального анализа данных, в том числе при реализации упрощенного алгоритма Байеса, в виде списков удобно представлять и обрабатывать различные массивы данных, включая базу данных, а также результирующую модель. Для этих целей подходит встроенная в язык COMMON LISP поддержка списков и функционал для работы с ними. В императивных языках для работы со списками требуется подключение библиотек, что является дополнительным шагом, а также может вносить затруднения, например при смене версий библиотек, переходе на новые среды выполнения, компиляторы и т. п.

Встроенный в язык COMMON LISP функционал для работы со списковыми структурами данных предоставляет разработчикам достаточно широкие возможности. В частности, имеется функционал, позволяющий применять ко всем элементам списка различные функции, передаваемые как параметры в виде указателей. Следует отметить, что подобный функционал позволяет создавать для алгоритмов интеллектуального анализа данных решения, имеющие универсальный характер. Например, для решения задач интеллектуального анализа данных часто требуется обход массива данных в определенном порядке с целью обработки его элементов. Разные алгоритмы могут иметь общий набор функционала для обхода данных, но при этом передавать указатели на различные обрабатывающие данные функции.

При решении задач интеллектуального анализа данных часто необходимо обрабатывать различные массивы данных, что требует реализации в программах циклов для их обработки. Ранее отмечалось, что в языке COMMON LISP, как и во всех остальных языках функционального программирования, в случае использования чистого функционального подхода при реализации программы применять циклы запрещено. Для организации повторений и замены циклов допустима только рекурсия, которая позволяет выполнять повторные действия без использования переменных.

Ниже приведен пример функции, иллюстрирующей реализацию рекурсии на языке COMMON LISP. Функция class_entr_count выполняет обход всех векторов базы данных, которая передается в функцию с помощью параметра data_base_list. Обход выполняется с целью подсчета числа вхождений в базу данных класса, передающегося в функцию через параметр cur_class. Число вхождений заданного класса в базу данных является возвращаемым функцией значением.

```
(DEFUN class_entr_count (cur_class
                         data_base_list)
  (COND
   ((NULL data_base_list) 0)
   ((EQUAL (NTH g_class_number_in_row
                (CAR data_base_list))
           cur_class)
    (+ 1 (class_entr_count cur_class
                           (CDR data_base_list))))
   (T (class_entr_count cur_class
                        (CDR data_base_list)))))
```

Кроме рекурсивного вызова, обеспечивающего замену цикла, в функции используется условный оператор COND, позволяющий реализовать ветвление алгоритма. Выражение

```
(NULL data_base_list)
```

проверяет условие достижения окончания рекурсии. Если условие истинно, рекурсия прекращается и функция class_entr_count возвращает значение «ноль». В противном случае проверяются следующие условия. С помощью выражения

```
(EQUAL (NTH g_class_number_in_row
            (CAR data_base_list))
       cur_class)
```

проверяется присутствие заданного класса в обрабатываемом на данном шаге векторе базы данных. Если условие истинно, то происходит рекурсивный вызов функции class_entr_count, при этом возвращаемое значение увеличивается на единицу. В случае ложности рассматриваемого условия происходит переход к следующему условию, которое в языке COMMON LISP всегда истинно, так как задается константой T. В этом случае происходит рекурсивный вызов функции class_entr_count без увеличения на единицу возвращаемого значения.

Ниже приведен пример функции, иллюстрирующей удобство работы со списками на языке COMMON LISP.

```
(DEFUN normal_probability_list (probability_list)
  (LET ((probab_sum (APPLY #'+
                           probability_list)))
    (MAPCAR #'(LAMBDA (probab) (/ probab
probab_sum))
                    probability_list)))
```

В данном примере функция normal_probability_list принимает в качестве параметра числовой список probability_list и возвращает нормированный список, полученный из исходного делением каждого элемента исходного списка на сумму всех его элементов. В данном примере отсутствуют циклы, при этом все действия производятся с помощью встроенного в язык COMMON LISP функционала для работы со списками.

Выражение

```
(APPLY #'+ probability_list)
```

возвращает сумму всех элементов числового списка probability_list.

Выражение

```
(MAPCAR #'(LAMBDA (probab) (/ probab
probab_sum))
                probability_list))
```

вызывает для каждого элемента числового списка probability_list функцию, которая передается в качестве параметра и возвращает список, сформированный из результатов вызова функций для каждого из элементов. В примере в качестве указателя на функцию используется лямбда-выражение, которое принимает в качестве параметра число probab и возвращает результат деления числа probab на число probab_sum.

Актуальным направлением в разработке и реализации моделей Data mining является ориентация на выполнение алгоритмов анализа в распределенной вычислительной среде, которое поддерживается методами соответствующей декомпозиции алгоритмов Data mining с целью их распараллеливания [8].

Такая декомпозиция алгоритма и дальнейшая реализация способов распараллеливания возможна как на объектно-ориентированном языке программировании в виде набора классов, так и на языке функционального программирования.

СПИСОК ЛИТЕРАТУРЫ

1. Интеллектуальный анализ данных в распределенных системах / М. С. Куприянов, И. И. Холод, З. А. Каршиев, И. А. Голубев. СПб.: Изд-во СПбГЭТУ «ЛЭТИ», 2012.

2. Анализ данных и процессов: учеб. пособие / А. А. Барсегян, М. С. Куприянов, И. И. Холод и др. 3-е изд., перераб. и доп. СПб.: БХВ-Петербург, 2009.

3. RapidMiner Studio Manual // Rapidminer. 2014. URL: https://rapidminer.com/wp-content/uploads/2014/10/RapidMiner-v6-user-manual.pdf.

4. Witten I. H., Frank E., Hall M. A. Data Mining: Practical Machine Learning Tools and Techniques. Morgan Kaufmann Publishers, 2011.

5. Bloomfield V. A. Using R for Numerical Analysis in Science and Engineering. Chapman & Hall/CRC, 2014.

6. Kerdprasop N., Kerdprasop K. Mining Frequent Patterns with Functional Programming // Intern. J. of Computer, Information, Systems and Control Engineering. 2007. Vol. 1, № 1. P. 120-125. URL: http://www.waset.org/publications/6340/-mining-frequent-patterns-with-functional-programming.

7. Amanda C., King R. Data mining the yeast genome in a lazy functional language. URL: http://users.aber.ac.uk/afc/papers/ClareKingPADL.pdf.

8. Холод И. И. Способы параллелизации алгоритмов интеллектуального анализа // Изв. СПбГЭТУ «ЛЭТИ». 2012. № 9. С. 39–47.

A. V. Malov
Motorola Solutions

S. V. Rodionov
Saint-Petersburg state electrotechnical university «LETI»

THE REALIZATION OF NAIVE BAYES ALGORITHM IN THE FUNCTIONAL PROGRAMMING FRAMEWORK COMMON LISP

The main problems and features of data mining are observed. The conveniences providing by the usage of functional language COMMON LISP for the solving of typical data mining problems are observed through the example of the realization of Naive Bayes algorithm.

Data mining, functional programming, naive Bayes algorithm