УДК 20.53.19, 28.23.13

И.И. Холод

Функциональная модель для параллелизации алгоритмов интеллектуального анализа данных

Аннотация. В данной статье описывается подход создания параллельных алгоритмов интеллектуального анализа данных с помощью функциональной модели, расширяющей λ-исчисления встроенными функциями необходимыми для описания таких алгоритмов. В результате она позволяет представить алгоритмы в виде последовательности вызовов унифицированных, потокобезопасных функций - функциональных блоков. Алгоритмы, построенные из таких функциональных блоков, могут легко перестраиваться, для оптимизации и формирования новых алгоритмов, а также распараллеливаться (как следствие теоремы Черча-Россера).

В качестве примера рассматривается алгоритм Naïve Bayes и варианты его параллельного выполнения.

Ключевые слова и фразы: параллельные вычисления, parallel data mining, интеллектуальный анализ данных, data mining.

Введение

Алгоритмы интеллектуального анализа данных (ИАД), выполняющие анализ сырых данных и извлечение из них полезных знаний [1], наиболее эффективны при применении их к большим объемам информации. Увеличение объема, приводят к увеличению временных затрат на анализ. Эта проблема может быть решена за счет параллельного выполнения таких алгоритмов

Можно выделить два основных подхода к созданию параллельных алгоритмов:

- разработка алгоритма с явным выделением параллельно выполняющихся ветвей и обеспечение их совместной работы;
- формирование алгоритма из блоков, обладающих свойством потокобезопастности, и последующая их (в том числе автоматическая) параллелизация.

Первый подход требует от программиста ответить на вопрос «как распараллелить программу». Он имеет явный недостаток, связанный с высокой трудоемкостью и большими затратами, как на разработку, так и на отладку.

Второй подход позволяет программисту ответить на вопрос «что нужно выполнить параллельно». Он позволяет трансформировать после-

довательный алгоритм, построенный из потокобезопастных блоков, в параллельный алгоритм, распределив такие блоки по параллельным ветвям. Основным достоинством такого подхода является простота его параллелизации и хорошая масштабируемость. Это позволяет компенсировать недостаток, связанный с возможно низкой эффективностью такого параллельного алгоритма в определенной среде выполнения.

Второй подход используется функциональными языками программирования. В них, в качестве блоков, из которых строятся программы, используются чистые функции.

Использование основных принципов заложенных в функциональных языках программирования позволяет перенести их достоинства и на итеративные алгоритмы.

1. Функциональная модель алгоритма интеллектуального анализа данных

1.1. Требования к блокам алгоритма

Для создания алгоритмов, как последовательности произвольных блоков, такие блоки должны обладать следующими свойствами:

- взаимозаменяемость;
- исполнение в любом порядке.

Первое требование взаимозаменяемости реализуется за счет унификации входного и выходного интерфейса блока. Все блоки, для создания алгоритмов, должны получать одинаковый входной набор аргументов и возвращать одинаковый набор результатов.

Возможностью выполнения в произвольном порядке обладают функции в функциональных языках программирования. Такие языки основаны на теории λ- исчислений, в рамках которой известна теорема Черча-Россера [2]: *состояния, которые могут быть достигнуты при применении редукций разным порядком к какому либо терму в λ- исчислениях могут быть сведены к одинаковому результату.*

В λ- исчислении редукцией называют преобразование некоторого λ-терма. Сам λ-терм описывает в том числе и функциональную зависимость (λ-выражение) и может иметь вид:

$$\lambda x.e\ v, \text{ где}$$

x – переменная - аргумент функции;
e – выражение, вычисляющее значение на основании аргумента x;
v – значение аргумента, подставляемое вместо аргумента x.

В λ-исчислении определены четыре вида редукций (α -, β -, δ - *и* η - *редукции*) [2]. Наиболее значимыми редукциями с практической точки зрения являются δ - *редукция* и β-*редукция:*

β - редукция - редукция – называется преобразование в результате которого выполняется подстановка вместо переменной ее значения:

$$\lambda x.e(x)\, y \rightarrow e(y)$$

δ - редукция – называется преобразование в результате которого вместо λ-выражения выполняется подстановка результата вычисления:

$$e(y) \rightarrow z, \text{ где } z \text{ результат вычисления } e(y)$$

Редукции могут применяться к исходному выражению последовательно, меняя его форму. Применение редукций к исходному выражению, по сути, является выполнением программы, а само выражение в этом случае можно рассматривать как программу (алгоритм). При этом, согласно теореме Черча-Россера применение редукций (т.е. выполнение программы) может выполняться в любом порядке и даже параллельно [3].

Это возможно за счет того что функции в λ-исчислении являются чистыми. Функция является чистой, если она обладает следующими свойствами [2]:

- детерминирована, т.е. возвращает один и тот же результат для одинакового набора данных;
- не имеет побочных эффектов, т.е. не изменяет и не использует глобальных переменных.

Таким образом, для того, чтобы можно было построить алгоритм из набора блоков эти блоки должны быть:

- унифицированы, т.е. иметь одинаковый входной и выходной интерфейс;
- реализованы как чистые функции.

Блоки, удовлетворяющие этим требованиям, будем называть функциональными блоками.

1.2. Функциональное представление алгоритма интеллектуального анализа данных

Опишем функциональную модель представления алгоритмов ИАД, представив сам алгоритм, как композицию функций и расширив λ- исчисления необходимыми встроенными функциями.

Формально алгоритм ИАД можно представить как функцию, принимающую на вход набор данных D и строящую модель знаний M [4,5]:

$$f: D \rightarrow M^1$$

Утверждение. Алгоритм ИАД можно представить функциональным выражением в виде композиции функций:

$$f = fb_n \circ fb_{n-1} \circ \ldots \circ fb_i \circ \ldots fb_1 = fb_n\,(d, fb_{n-1}\,(d, \ldots. fb_i\,(d, \ldots. fb_1(d)\ldots)..)),$$

где fb_i –функция, выполняющая некоторые вычисления (часть алгоритма) и строящая модель знаний m_i на основе набора данных d и модели знаний построенной предыдущей функцией m_{i-1} .

$$fb_i: D, M \rightarrow M$$

Каждая такая функция, также может быть представлена композицией функций:

$$fb_i = fb_{i.k} \circ \ldots \circ fb_{i.r} \circ \ldots fb_{i.1} = fb_{i.k}\,(d, \ldots. fb_{i.r}\,(d, \ldots. fb_{i.1}(d)\ldots)..),$$

Доказательство. Такая форма соответствует правилам λ-исчислений и может быть записана в форме λ-выражения следующим образом:

$$(\lambda d\ m.fb_n(d,m))\ ((d\ \lambda d\ m.fb_{n-1}(d,m))\ \ldots.\ (\lambda d\ m.\ fb_i(d,m))\ \ldots.\ (\lambda d.fb_1(d))\ d)..)),$$

где d - набор данных, m – модель знаний.

Применение β-редукции позволит преобразовать представленное выше функциональное выражение к результату в виде модели знаний. Например, для алгоритма из 3х функциональных блоков λ-выражение будет выглядеть следующим образом:

$$(\lambda d\ m.\ fb_2(d,m))\ ((d\ \lambda d\ m.\ fb_1(d,m))\ (d\ (\lambda d\ m.\ fb_0(d,\ m))\ d\ null)))$$

Использовав, например аппликативный порядок применения редукции мы получим:

$$(\lambda d\ m.\ fb_2(d,m))\ ((d\ \lambda d\ m.\ fb_1(d,m))\ (d\ (\lambda d\ m.\ fb_0(d,\ m))\ d\ null))) \rightarrow$$

$$(\lambda d\ m.\ fb_2(d,m))\ ((d\ \lambda d\ m.\ fb_1(d,m))\ (d\ fb_0(d,\ null))) \rightarrow$$

$$(\lambda d\ m.\ fb_2(d,m))\ (d\ b_1(d,\ fb_0(d,\ null))) \rightarrow$$

$$fb_2(d,\ fb_1(d,\ fb_0(d,\ null))) \rightarrow m$$

В результате модель знаний m будет вычислена последовательным выполнением блоков fb_0 , fb_1 и fb_2

[1] Здесь и далее заглавными буквами будем обозначать типы, а прописными переменные этих типов (например, переменная d типа D).

Для унификации функций, будем считать, что первая функция fb_1 (а следовательно и алгоритм) в качестве входного аргумента получает пустую модель знаний $m_0 = \varnothing$, следовательно алгоритм ИАД в виде функции будет описан следующим образом:

$$f: D, M \rightarrow M$$

Унификация интерфейсов функций и использование ими для вычисления только входных аргументов (т.е. наличие у них свойств чистых функций) позволяет называть функции fb_i функциональными блоками.

1.3. Расширение функциональной модели встроенными функциями

Алгоритмы ИАД (как и другие алгоритмы), могут включать в себя такие структурные элементы как: условные операторы, циклы (в том числе циклы по векторам и по атрибутам [6]), параллельные ветви. Для описания этих элементов в виде функциональных выражений добавим встроенные функции и покажем, как с их помощью можно записать перечисленные структурные элементы в функциональном виде.

Утверждение. Условный оператор в функциональном виде может быть записан как функция высшего порядка, принимающая в качестве аргументов другие функции:

$$dec: D, M, cf, fb, fb \rightarrow M,$$

$$dec(d, m) = if\ cf(d, m)\ then\ fb_t(d, m)\ else\ fb_f(d, m)\ ,$$ где

cf – функция, вычисляющая условное выражение, вида:

$$cf: D, M \rightarrow BOOLEAN\ ;$$

fb_t – функциональный блок (4й аргумент функции), выполняющийся если результат функции cf – истина;

fb_f - функциональный блок (5й аргумент функции), выполняющийся если результат функции cf – ложь.

Доказательство.

В теории λ-исчислений булевские типы и условные выражения описываются с помощью следующих λ-выражений [2]:

$$IF = \lambda p.\lambda t.\lambda e.p\ t\ e,$$ где

p логическое выражение, которое может быть представлено как λ-выражение в виде:

$TRUE = \lambda x.\lambda y.x$

$FALSE = \lambda x.\lambda y.y;$

t – выражение выполняющееся если p=TRUE;

e – выражение, выполняющееся если p=FALSE.

Таким образом, описанную выше функцию условного оператора можно записать в виде следующего λ-выражения:

$$IF = \lambda p.\lambda t.\lambda e.\ (cf\ d\ m)\ (fb_t\ d\ m)\ (fb_f\ d\ m)$$

Для использования функции *dec* как функционального блока у нее должны быть заданы все функции аргументы.

В функциональных языках программирования циклы отсутствуют, в виду отсутствия состояния программы и операторов присваивания. Для выполнения итерационных действий без сохранения состояния программы используется рекурсивный вызов функции. С его помощью и запишем циклы алгоритмов ИАД.

Утверждение. Цикл может быть записан как рекурсивная функция высшего порядка вида:

$$loop: D, M, fb, cf, fb, fb \rightarrow M,$$

$$loop = loop' \circ fb_{init},\ \text{где}$$

fb_{init} – функциональный блок (3й аргумент), выполняющийся перед циклом (инициализирующий цикл);

loop' – вспомогательная рекурсивная функция:

$$loop'(d,m)=if\ cf(d,m)\ then\ loop' \circ fb_{iter}(d,m) \circ fb_{pre}(d,m)\ else\ fb_{iter}(d,m) \circ fb_{pre}(d,m),$$

где fb_{pre} – функциональный блок (5й аргумент), выполняющийся перед каждой итерацией;

fb_{iter} – функциональный блок (6й аргумент), выполняющийся итерационно.

Доказательство. Докажем, корректность записи применив последовательно редукции, например, для трех итераций. В результате получим следующее выражение, которое соответствует логике выполнения цикла:

$loop\ (d, m, fb_{init}, cf, fb_{pre}, fb_{iter},) \rightarrow$
$loop'\ (d, (fb_{init}\ (d, m)), cf, fb_{pre}, fb_{iter})) \rightarrow$
if $(cf(d, (fb_{init}\ (d, m)))$ *then // условие истинно*
$(loop'\ (d, (fb_{iter}\ (d, (fb_{pre}\ (d,(fb_{init}\ (d, m)))\ cf, fb_{pre}, fb_{iter}) \rightarrow$
if $(cf\ (d, (fb_{init}\ (d, m)))$ *then // условие истинно*
if $(cf\ (d, (fb_{init}\ (d, (fb_{init}\ (d, m)))))$ *then // условие истинно*
$loop'(d,(fb_{iter}(d,(fb_{pre}(d,(fb_{iter}(d,(fb_{pre}(d,(fb_{init}(d, m))))))))))),cf,fb_{pre},fb_{iter}) \rightarrow$
if $cf\ (d,(fb_{init}(d, m)))$ *then // условие истинно*
if $cf\ (d, (fb_{init}\ (d,(fb_{init}(d, m)))))$ *then // условие истинно*

if cf(d,(fb_{iter}*(d,(*fb_{pre}*(d,(*fb_{iter}*(d,(*fb_{pre}*(d,(*fb_{init} *(d,m))))))))))) // условие ложно*
else

$$\boldsymbol{fb_{iter}(d,(fb_{pre}(d,(fb_{iter}(d,(fb_{pre}(d,(fb_{iter}(d,(fb_{pre}(d,(fb_{init}(d,m))))))))))))}$$

В результате будет выполнено выражение, записанное в последней строке. Можно убедиться в том что:

- инициализация цикла (функция fb_{init}) будет выполнена один раз в самом начале выполнения выражения;
- основной блок цикла (функция fb_{iter}) будет выполнен три раза и результаты каждого выполнения будут переданы следующему блоку с предварительной обработкой (функция fb_{pre}).

Для использования цикла *loop* как функционального блока у него должны быть заданы все функции аргументы. В частности для алгоритмов ИАД характерными являются циклы по векторам и атрибутам [6]. Для них можно заранее определить необходимые некоторые функции. Для цикла по векторам $loop_W$:

- $fb_{init} = fb_{initW}$ – функциональный блок, инициализирующий проход по векторам набора данных (например, устанавливающий позицию курсора набора данных на первый вектор);
- $fb_{pre} = fb_{preW}$ – функциональный блок, определяющий следующий вектор для обработки (например, сдвигающий позицию курсора набора данных на следующий вектор);
- $cf = cf_W$ – условная функция, проверяющая, что все вектора в наборе данных были обработаны.

Для цикла по атрибутам $loop_A$ можно определить следующие функции аргументы:

- $fb_{init} = fb_{initA}$ – функциональный блок, инициализирующий проход по атрибутам вектора (например, устанавливающий позицию курсора вектора на первый атрибут);
- $fb_{pre} = fb_{preA}$ – функциональный блок, определяющий следующий атрибут для обработки (например, сдвигающий позицию курсора вектора на следующий атрибут);
- $cf = cf_A$ – условная функция, проверяющая, что все атрибуты у вектора были обработаны.

Таким образом, для вызова циклов по векторам и по атрибутам, как функционального блока, достаточно в качестве аргумента передать только функциональный блок для итерационного выполнения fb_{iter}:

$$loop_W(d, m, fb_{iter}) \text{ и } loop_A(d, m, fb_{iter}).$$

1.4. Использование функциональной модели для описания параллельной формы

Как указывалось выше, одним из основных преимуществ построения алгоритмов из функциональных блоков является возможность их параллельного выполнения. При этом практическое значение имеет параллельное выполнение функциональных блоков вычисляющих аргументы при аппликативном порядке редукции. Таким образом, для параллельного выполнения блоков алгоритмов ИАД они должны вызываться для вычисления аргументов одной функции. Например, в выражении:

$$fb_i\,(d,\ fb_j(d,\ m),\ fb_p(d,\ m),\ fb_q(d,\ m))$$

блоки fb_j, fb_p и fb_q могут быть вычислены параллельно. Такого рода параллельные вычисления соответствуют параллелизму по задачам.

Алгоритмы ИАД в структуре, которых есть такие блоки, обладают внутренним параллелизмом и могут быть распараллелены. Однако алгоритмам ИАД в основном присущ параллелизм по данным. В этом случае требуется явное преобразование функционального выражения с добавлением функции разделения данных и последующим объединением результатов. Добавим в модель функцию высшего порядка, позволяющую выполнить параллелизацию по данным в алгоритмах ИАД:

$$parall:\ D,\ M,\ fb,\ split,\ join \rightarrow M$$

$$parall = join \circ fb \circ split, \text{ где}$$

– *split* - функция, выполняющая разделение набора данных D (возможно в зависимости от модели знаний M) и возвращающая список из разделенных наборов данных:

$$split:\ D,\ M \rightarrow [D]$$

– *join* - функция, объединяющая модели знаний M из списка и возвращающая объединенную модель M:

$$join:\ [M] \rightarrow M.$$

В функции *join* элементы списка моделей знаний *[M]* могут вычисляться функциональными блоками, аргументами которых будут наборы данных из списка, построенного блоком *split*. Таким образом, результат композиции функции *parall* можно записать следующим образом:

$$parall = join\ ([fb_i(m,\ split(d,\ m)[0])\ \ldots\ fb_i(m,\ split(d,\ m)[n])]).$$

В таком выражении функциональный блок fb_i вызывается для вычисления аргументов функции *join*, а значит (следуя теореме Черча-Россера) может быть выполнен параллельно. Функция *parall* может быть добавлена для любого функционального блока в функциональном выражении.

2. Пример использования функциональной модели для параллелизации алгоритма Naïve Bayes

В качестве примера представления алгоритма ИАД с помощью функциональной модели рассмотрим алгоритм Naïve Bayes (листинг 1).

```
1. for i=0 to |W|-1 // для всех векторов
2.         incCountOfVectorsOfClass() // увеличить на 1 счетчик количества векторов, принадлежащих классу i-го вектора.
3.         for j=1 to |A|  // для всех независимых атрибутов
4.                 incCountOfVectorsOfAttr()   // увеличить на 1 счетчик количество векторов с значениями j-го и целевого атрибутов i-го вектора.
5.         end for j;
6. end for i;
```

Листинг 1. Алгоритм Naive Bayes

Опишем представленный выше алгоритм Naïve Bayes с помощью предложенной функциональной модели для последующей его параллелизации. Для этого представим каждую операцию, выполняющуюся в алгоритме, с помощью предложенной модели.

Основное место в структуре алгоритма занимает цикл по векторам (строки 1-10). Его можно записать с помощью функции высшего порядка $loop_W$ описанной в модели. Функциональный блок, выполняющийся итерационно в цикле fb_{iter}, включает в себя операции изменения модели знаний в строках 2 (в зависимости от значения целевого атрибута i-го вектора) и цикле по атрибутам i-го вектора в строках 3-9:

$$fb_{iter}=fb_1 = loop_A \circ fb_2, \text{ где}$$

fb_2 - функциональный блок, соответствующий 2-й строке, формирует модель знаний, в которой счетчик количества векторов для класса целевого атрибута вектора i увеличен на 1. Таким образом:

$$f_{NB}(d,m_0) = loop_W(d, m_0, fb_{iter}) = loop_W(d, m_0, loop_A \circ fb_2)$$

Цикл по атрибутам (строки 3-9) представляется функцией высшего порядка $loop_A$. Функциональный блок, выполняющийся итерационно в цикле, включает операции изменения модели знаний в строках 4-8 в зависимости от значений j-го и целевого атрибута i-го вектора:

$$fb_{iter} = fb_4, \text{ где}$$

fb_4 – функциональный блок, выполняющий увеличение на 1 счетчика количества векторов с значениями j-го и целевого атрибутов i-го вектора. Таким образом:

$$loop_A(d, m_i, fb_{iter}) = loop_A(d, m_i, fb_4)$$

В результате алгоритм содержит следующие функциональные блоки: $loop_W$, fb_2, $loop_A$, $loop_A \circ fb_2$, fb_4. Каждый из них можно выполнить параллельно с помощью функции *parall*. Рассмотрим каждый из вариантов:

1.Параллелизация функционального блока $loop_W$:

$$parall(d, m, loop_W, split, join).$$

В этом случае весь алгоритм будет выполняться параллельно. Учитывая, что распараллеливаемая часть является циклом по векторам $loop_W$, это означает, что параллельно будет выполняться обработка векторов в соответствии с распределением данных (функцией *split*). Таким образом, в каждой параллельной ветви будет построена модель знаний по той части векторов, которые будут распределены ей. Построенные модели знаний будут объединены функцией *join*. Учитывая особенности модели знаний Naïve Bayes Model [7, 8] после объединения будет получена модель знаний совпадающая с моделью знаний построенной последовательным выполнением алгоритма. Учитывая, что время выполнения обработки векторов напрямую зависит от их количества, выполнение в каждой ветви блока $loop_W$ применительно к подмножеству векторов будет занимать меньше времени, чем применительно ко всему множеству. Это в итоге позволит сократить время выполнения алгоритма при рассмотренном варианте параллелизации.

2.Параллелизация блока fb_2:

$$loop_W(d, m_0, loop_A \circ parall(d, m, fb_2, split, join))$$

В этом случае параллельно будет выполнена инкрементация счетчика количества векторов с соответствующим (одним и тем же) значением целевого атрибута. Учитывая, что эти действия не зависят от данных, в каждой ветки будет выполнено абсолютно одинаковые действия. По-

этому такой вариант не снизит время выполнения алгоритма и поэтому не имеет практического смысла.

3.Параллелизация блока $loop_A$:

$$loop_W(d, m_0, parall(\ d, m, loop_A, split, join)\ ^\circ fb_2)$$

При таком варианте параллелизации параллельно будет выполняться цикл по атрибутам $loop_A$. При этом если разделение данных будет выполняться по атрибутам (т.е. на разные ветви будут передавать разные атрибуты векторов), то строящиеся модели будут отличаться. Учитывая особенности модели знаний Naïve Bayes Model [7,8] после объединения будет получена модель знаний, совпадающая с моделью знаний построенной последовательным алгоритмом. Учитывая, что время выполнения обработки атрибутов напрямую зависит от количества обрабатываемых атрибутов, то выполнение в каждой ветви блока $loop_A$ применительно к подмножеству атрибутов будет занимать меньше времени. В случае большого числа атрибутов такой вариант параллелизации в итоге позволит сократить время выполнения алгоритма.

4.Параллелизация блока $loop_A\ ^\circ fb_2$

$$loop_W(d, m_0, parall(\ d, m, loop_A\ ^\circ fb_2, split, join))$$

При данном варианте параллелизации, учитывая, что в каждой параллельной ветви будет дублироваться обработка i-го вектора (как в варианте 2), то получаем преимущества варианта 3 .

5.Параллелизация блока fb_4:

$$loop_A(d, m_i, parall(\ d, m, fb_4, split, join))$$

Данный вариант параллелизации как и во 2м варианте предполагает параллельную инкрементацию счетчика количества векторов соответствующих анализируемому вектору. Учитывая, что данная операция не зависит от данных, она будет выполняться абсолютно идентично в каждой параллельной ветки, а следовательно такой вариант параллелизации не имеет смысла.

В результате можно утверждать, что имеет смысл параллельное выполнение цикла по векторам (вариант 1) и при большом количестве атрибутов параллельное выполнение цикла по атрибутам (вариант 3). Для более точной оценки возможно экспериментальное выполнение каждого из вариантов и последующая оценка эффективности для него. На основе полученных оценок можно сделать более точный выбор более эффективного варианта. Учитывая, что при таком способе записи алгоритма интеллектуального анализа данных(с помощью предложенной функциональной модели), определение вариантов паррлелизации строго

формализовано (подстановка функции *parall* выполняется для каждого функционального блока), то сформировать все возможные варианты параллельного выполнения алгоритма можно автоматически. Следовательно, выполнив автоматически оценку каждого варианта, можно автоматически построить наиболее эффективный вариант параллельного выполнения алгоритма из его последовательной версии.

Заключение

В статье предложена функциональная модель описания алгоритма интеллектуального анализа данных, опирающаяся на принципы теории λ-исчислений. Такое представление позволяет декомпозировать алгоритмы на функциональные блоки: имеющие унифицированный интерфейс и обладающие свойствами чистых функций. Для описания структурных элементов алгоритмов интеллектуального анализа данных с помощью функциональных блоков в модель были введены встроенные функции высшего порядка: цикл *loop*, цикл по векторам *loopW*, цикл по атрибутам *loopA*, условное выражения *dec*. Для описания параллельной формы алгоритма с помощью функциональной модели в нее добавлена функция распараллеливания по данным *parall*.

Представление алгоритма с помощью предложенной функциональной модели и явное выделение в нем функциональных блоков, позволяет не только построить алгоритм, но и преобразовать его к параллельной форме минимальными изменениями.

На примере алгоритма Naive Bayes, показана возможность описания алгоритма интеллектуального анализа данных с помощью предложенной модели. На основе данного описания показано формирование различных вариантов параллельного выполнения алгоритма.

Список литературы

[1] Анализ данных и процессов. /А.А.Барсегян, М.С.Куприянов, И.И.Холод, М.Д.Тесс, С.И.Елизаров. СПб.: БХВ-Петербург, 2009. 512 с.: ил.

[2] Barendregt, H. P. The Lambda Calculus: Its Syntax and Semantics, Volume 103 of Studies in Logic and the Foundations of Mathematics. North-Holland, 1981

[3] Топорков В.В. Модели распределенных вычислений. М.: ФИЗМАТЛИТ, 2004. 320 стр

[4] Common Warehouse Metamodel (CWM) Specification. http://www.omg.org/spec/CWM/1.1/

[5] Java Specification Request 73: Java Data Mining (JDM) –JDM Public review Draft 2003/11/25 : JSR-73 Expert Group

[6] М. С. Куприянов, И. И. Холод, З. А. Каршиев, И. А. Голубев. Интеллектуальный анализ данных в распределенных системах. СПб.: Изд-во СПбГЭТУ «ЛЭТИ», 2012. 110 с..
[7] PMML Specification.: Data Mining Group. http://www.dmg.org/PMML-4_0
[8] И.И. Холод // Унифицированная модель Data Mining // Сборник докладов XV Международной конференции по мягким вычислениям и измерениям SCM`2012, Санкт-Петербург, 25-27 июня, 2012 г., Том 1, с. 237-240.

Об авторе:

Холод Иван Иванович

К.т.н., доцент кафедры вычислительная техника Санкт Петербургского государственного электротехнического университета «ЛЭТИ» им. В.И. Ульянова (Ленина)

e-mail: iiholod@mail.ru