DOI: 10.18137/RNU.V9187.21.04.P.161 УДК 004.855.5

П.А. Загороднюк

DATA MINING НА ЯЗЫКЕ GO

Целью данной статьи является оценка языка программирования Go как инструмента для реализации методов data mining. Для этого проводится анализ задачи классификации и метода *k*-ближайших соседей, затем предлагается способ программирования данного метода и организации процесс управления и подготовки исходных данных. В заключение на основе проведенной работы делается вывод, насколько Go подходит для решения подобных задач, и есть ли потенциал для реализации остальных методов.

Ключевые слова: язык программирования Go, добыча знаний, машинное обучение, задача классификации, метод *k*-ближайших соседей.

P.A. Zagorodnyuk

DATA MINING IN GO

The purpose of this article is to evaluate the Go programming language as a tool for implementing data mining methods. To do this, an analysis of the classification problem and the k-nearest neighbors' algorithm is carried out, then a method is proposed for how this method can be programmed and the process of managing and preparing the initial data can be organized. In conclusion, based on the work carried out, it sums up how well Go is suitable for solving such problems and whether there is potential for the implementation of other methods.

Keywords: Go programming language, data mining, machine learning, data classification, k-nearest neighbor algorithm.

Введение

Data mining – мощная и эффективная технология обработки больших массивов данных, позволяющая открывать новые, ранее неизвестные факты и зависимости, а также прогнозировать различные показатели объектов и событий [4; 5]. Сегодня data mining играет весомую роль. Область применения этой технологии не ограничена – она может внедряться везде, где есть большие накопления каких-либо ретроспективных данных: бизнес, веб-технологии, научные исследования, государственная сфера и др.

Примером использования data mining в бизнесе могут быть различные корпоративные информационные системы (далее – КИС) и системы поддержки принятия решений (далее – СППР). В сфере научных исследований технологии работы с данными широко применяются в биологии и медицине. С их помощью анализируется генетическая информация, прогнозируются вероятности различных заболеваний для пациентов и составляются индивидуальные программы лечения.

С использованием веб-технологий решаются задачи определения поведения и предпочтений пользователей, выявления мошеннических и опасных действий; в государственной сфере – задачи поиска лиц, скрывающихся от налогов, отслеживания действий иностранцев, прибывших в страну, и др.

Go – компилируемый язык с открытым исходным кодом, мощной библиотекой и новыми возможностями, которые выводят качество и процесс разработки на совершенно новый уровень [1–3]. Этот язык был разработан в Google Робертом Грисемером, Робом

Загороднюк Павел Анатольевич
магистрант Московского энергетического института (национальный исследовательский университет), Москва. Сфера научных интересов: разработка сетевых приложений, анализ данных, data mining.
Электронный адрес: pavel.zagorodnuik@gmail.com

Пайком и Кеном Томпсоном и официально представлен в ноябре 2009 года. На данный момент Go очень востребован – в корпорации Google он является пятым официальным языком. Используется в IBM, Adobe, Intel, Яндекс, ВКонтакте. На нем написаны такие инструменты, как Docker и Kubernetes.

Go является универсальным языком для решения общих задач, но из-за своих особенностей обрел большую популярность в разработке информационных систем. К его сильным сторонам можно отнести простой и удобный Си-подобный синтаксис, многопоточность, производительность, мощную библиотеку, новый подход к реализации объектно-ориентированного программирования и др.

Использование методов data mining в информационных системах существенно повышает их эффективность, добавляя ранее недоступный функционал, а также в целом улучшает впечатления пользователей от их применения. Очевидными примерами могут быть рекомендательные системы интернет-магазинов, онлайн-кинотеатров и стриминговых сервисов, где происходит изучение интересов пользователей и последующее предложение им тех или иных продуктов и услуг. Этот функционал основан на методах data mining и оказывает полезный эффект не только на клиентов, но и на компании, которые владеют подобными системами.

Как уже было сказано, Go часто используется в разработке информационных систем, но основной «конфликт» заключается в том, что реализаций методов data mining в Go попросту нет. В данной статье предлагается рассмотреть, как можно реализовать метод классификации в Go, а затем оценить, насколько Go подходит для подобных задач, есть ли потенциал для реализации остальных методов.

Задача классификации

В data mining все решаемые задачи разделены на десять категорий. Рассмотрим довольно часто встречающуюся в реальной практике категорию задач классификации.

Классификация – это процесс распределения объектов на заранее известные группы, называемые классами. Классификация объектов происходит заранее обученной моделью. Именно она решает, к какому классу необходимо отнести тот или иной объект. Чтобы построить такую модель, необходимо ее *обучить* на основе заранее классифицированных объектов и затем *протестировать*. Далее этот процесс будет представлен более детально, но сначала необходимо рассмотреть необходимую базу для представления исходных для методов данных.

Набор данных (или выборка данных) – это данные, которые можно представить в виде двумерной таблицы, где каждая строка – это какой-либо *объект*, а столбец – *атрибут объекта*. То есть выборка данных состоит из объектов (наблюдений, событий), а объекты – из атрибутов (свойств, характеристик). Например, лампочка – это объект, и у нее есть

такие характеристики, как потребляемая мощность, световая температура, тип цоколя и др. Атрибуты могут быть представлены разными шкалами: номинальными, порядковыми, интервальными, относительными.

Представляя объекты как множества числовых значений, например, $A=\{3.4,91,-8\}$ и $B=\{2.8,105,-5\}$, можно вычислить меру сходства пары объектов. Для этого используется евклидово расстояние, квадрат евклидова расстояния, манхэттенское расстояние и др. Если использовать евклидово расстояние, то мера сходства A и B будет $d^2(A,B)=1,002$.

Метод k-ближайших соседей реализует задачу классификации. Это базовый, достаточно простой и распространенный алгоритм. Суть его заключается в следующем: для определения класса нового объекта выделяется *k* наиболее схожих с ним уже классифицированных объектов (тех, что использовались при обучении модели); затем с применением невзвешенного или взвешенного голосования определяется класс нового объекта. Визуально действие алгоритма представлено на Рисунке 1, где крестиком обозначен объект, подлежащий классификации. Осями системы координат являются атрибуты объектов.

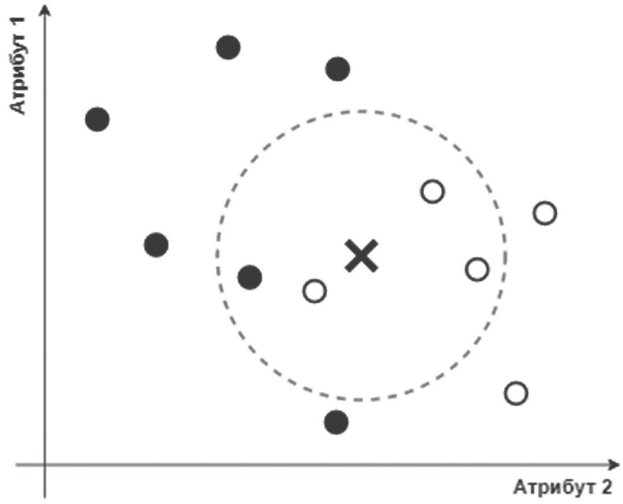

Рисунок 1. Классификация с помощью метода *k*-ближайших соседей при *k* = 4

На представленном Рисунке наблюдаем, что в окружении классифицируемого объекта имеется три белых объекта и один черный. В соответствии с невзвешенным методом голосования относим классифицируемый объект к белым, так как их больше. Если количество объектов каждого класса в окружении равно, то применяется *метод взвешенного голосования*, который учитывает расстояние между объектами в весах их голосов. Формула вычисления голосов взвешенным методом для определенного класса

$$votes(class)=\sum_{i=1}^{n}\frac{1}{d^2(X,Y_i)},$$

где $class$ – обозначение класса, для которого подсчитываются голоса; n – количество объектов класса $class$ в множестве *k*-ближайших; $d^2(X,Y_i)$ – квадрат расстояния между новым объектом X и известным объектом Y_i класса $class$.

Метод k-ближайших соседей на языке Go

Реализовать данный процесс классификации можно на языке Go. Первый этап классификации, *построение модели*, включает в себя обучение и тестирование. Обучение проводится на основе *обучающей выборки данных* – совокупности заранее классифицированных объектов. Результат обучения – *модель* (классификатор). Выборка данных представляется в виде некоторого типа данных *Data*, представляющего совокупность объектов без указания их классов; примем, что *Data* – это обычная выборка данных.

Задачу обучения модели будем выполнять в виде обыкновенной функции Go, вторым параметром которой будет срез целочисленных значений *[]int*, обозначающий классы объектов выборки. Можно было бы указать классы как совокупность значений какого-либо атрибута параметра *Data*, но это накладывает определенные сложности.

Тип данных (класс), которым будет представлена модель в Go, для простоты назовем *Model*. Экземпляр этого типа данных будет возвращать функция обучения модели. В Go при реализации объектно ориентированного программирования методы можно описывать для всех типов данных, поэтому любой тип данных является классом.

У метода *k*-ближайших соседей имеется два параметра: параметр *k*, определяющий количество ближайших объектов, участвующих в голосовании, и функция, определяющая алгоритм вычисления меры сходства двух объектов. Эти параметры будем задавать как целочисленное значение и указатель на функцию с сигнатурой *func (obj Object) Object*.

С учетом всего вышесказанного функция, которая создает и обучает модель, будет иметь вид *func Make Model(data Data, classes []int, k int, spacing func (obj Object) Object)Model*.

Любую обученную модель необходимо тестировать, чтобы проверить, насколько корректно она классифицирует объекты. Для тестирования необходима выборка данных с указанием классов объектов – модель классифицирует объекты этой выборки, а потом происходит сравнение ее решения с тем, которое как верное. Результатом тестирования является коэффициент, показывающий долю правильно классифицированных объектов. Вид сигнатуры тестирующего метода *func(m Model) Testing(data Data, classes []int)float64*.

Второй этап – использование модели для классификации объектов. Сигнатура метода классификации очевидная – *func(m Model)Classify(obj Object)int* – на входе объект, который необходимо классифицировать, на выходе индекс класса.

Необходимо обратить внимание на сам алгоритм классификации. Нужно заметить, что в нем имеется этап поиска *k* ближайших к классифицируемому объектов, который представляет собой множество однотипных, независимых и часто повторяющихся операций поиска минимальных расстояний. В этом процессе можно и нужно воспользоваться многопоточностью языка Go. Потоки в Go называются *go-подпрограммами*; они очень легковесные, поэтому можно не сомневаться в целесообразности их использования.

Вариантом решения данной задачи может быть использование go-подпрограмм в количестве, кратном числу ядер процессора, которые будут параллельно вычислять расстояния между объектами и отыскивать множество *k*-ближайших. Чтобы избежать взаимной блокировки, можно использовать мьютексы. Аналогичное использование многопоточности для повышения производительности также следует применить в методе *Testing*, чтобы классификация объектов происходила параллельно.

Выборки данных и ноды

Теперь перейдем к вопросу внутреннего устройства типа данных *Data*, с которым работают вышеописанные сущности. Он должен быть таким, чтобы пользователям

было удобно с ним работать, и чтобы вся система в целом была достаточно производительной. В процессе работы с исходными данными могут возникнуть следующие проблемы.

1. Исходные данные пользователя физически могут быть представлены в абсолютно любом формате: срезы, структуры, экземпляры классов с приватными полями и др. Если данных много, то задача конвертирования в общий формат будет ресурсозатратной.

2. Исходные данные пользователя могут находиться где угодно. Это может быть оперативная память, база данных, файл на жестком диске, микросервис и др.

3. Исходные данные необходимо подвергать процедуре предварительной обработки. Здесь имеется в виду нормализация (сведение значений всех атрибутов к единому масштабу), повышение качества путем очистки от пробелов, дублирований, выбросов и множество других действий. Каждый этап такой обработки – это новая версия данных в оперативной памяти.

Решить приведенные проблемы можно с помощью интерфейсов языка Go, а также нод. *Интерфейс в Go* – это абстрактный тип данных, скрывающий внутреннюю структуру объектов, но позволяющий вызывать относительно них методы, которые описаны в объявлении интерфейса. Если мы представим *Data* как интерфейсный тип, то внутренняя структура выборки данных может быть любой. Главное, чтобы относительно нее можно было вызвать методы, объявленные в *Data*.

Чтобы определить минимальный перечень необходимых методов, нужно понять, какая информация из выборки может понадобиться методам data mining. Очевидно, что это значения атрибутов каждого объекта, а также информация о том, сколько объектов имеется в выборке, каким количеством атрибутов они описываются. Определение интерфейса *Data* может иметь следующий вид:

```
type Data interface {
        GetValue(obj, attr int) float64
        Length() int
        Dimension() int
}
```

Эффективным дополнением к предложенному интерфейсу являются *ноды*. Как уже было сказано, перед тем как направить данные конкретному методу data mining, им необходимо пройти процедуру предварительной обработки, состоящей из очистки, нормализации и других различных преобразований. Важным фактом является то, что для всех этих действий входными и выходными параметрами являются выборки *Data*. *Нода* (или узел) – это функция, которая в качестве одного или нескольких параметров принимает значения *Data*, обрабатывает их и возвращает одно или несколько новых значений *Data*. Причем этой функции выделять память и записывать в нее преобразованную выборку совершенно необязательно. Так как входными и выходными значениями ноды являются интерфейсы, то обработка может выполняться *динамически*.

Примерами нод могут быть операции фильтрации данных, объединения и выделения, операции удаления и добавления атрибутов, объектов, а также нормализация и очистка.

Таким образом, с помощью интерфейсного типа *Data* и *нод* решаются все описанные выше проблемы. Пользователь может обрабатывать данные любого формата. Для этого нужно всего лишь определить три простых описанных выше метода, чтобы выборка соответствовала интерфейсу.

Заключение

В данной статье рассмотрен вариант реализации на языке *Go* выборок данных и метод *k*-ближайших соседей. В положительном ключе проявились такие его достоинства, как многопоточность и гибкий синтаксис. Также стоит отменить, что наличие в *Go* автоматической сборки мусора (очистки памяти от неиспользуемых данных), значительно повышает безопасность и упрощает работу при написании кода, а компиляция программ непосредственно в машинный код обеспечивает более высокую производительность, чем некоторые другие языки.

На основе всего вышесказанного можно сделать вывод, что *Go* отлично подходит для сферы data mining, и это делает его еще более мощным инструментом в разработке информационных систем.

Литература

1. *Батчер М., Фарина М.* Go на практике. М.: Пресс, 2017. 374 с.
2. *Доновaн Алан А.А., Керниган Брайан У.* Язык программирования Go. М.: Вильямс, 2016. 432 с.
3. *Саммерфильд М.* Программирование на языке Go. Разработка приложений XXI века. М.: Пресс, 2013. 582 с.
4. *Стивен С. Скиена.* Наука о данных: учебный курс. М.: Вильямс, 2020. 544 с.
5. *Чубукова И.* Data mining [Электронный ресурс]. URL: https://intuit.ru/studies/courses/6/6/info (дата обращения: 29.10.2021).

References

1. Butcher M., Farina M. (2017) *Go na praktike* [Go in Practice]. Moscow, Press Publishing, 374 p. (in Russian).
2. Donovan Alan A.A., Kernighan Brian W. (2016) *Yazyk programmirovaniya Go* [The Go Programming Language]. Moscow, Vil'yams Publishing, 432 p. (in Russian).
3. Summerfield M. (2012) *Programmirovanie na yazyke Go. Razrabotka prilozhenii XXI veka* [Programming in Go: Creating Applications for the 21st Century]. Moscow, Press Publishing, 582 p. (in Russian).
4. Steven S. Skiena (2020) *Nauka o dannykh: uchebnyi kurs* [Data Science: Training Course]. Moscow, Vil'yams Publishing, 544 p. (in Russian).
5. Chubukova I. (2021) *Sbor dannykh* [Data mining]. Available at: https://intuit.ru/studies/courses/6/6/info (date of the application: 29.10.2021).