Список использованных источников:

1. Johnson, J. Topological graph clustering with thin position / J. Johnson // Cornell University Library – Topological graph clustering with thin position. – Stillwater, 2012. – 2 с.
2. Хатчер, А. Алгебраическая топология / Хатчер А. // Алгебраическая Топология. Пер. с англ. В. В. Прасолова под ред. Т.Е. Панова – Москва, 2011. – 138 с.
3. Carlson, G. Topology and data / G. Carlson // American mathematical society. Topology and data. – California, 2009. – 256 с.
4. Edelsbrunner, H. Topological Persistence and Simplification / H. Edelsbrunner, D. Letscher, A. Zomorodian // Topological Persistence and Simplification. – Illinois, 2002. – 3 с.
5. Edelsbrunner, H. Computational topology: an introduction / H. Edelsbrunner, J. Harer // American mathematical society. Computational topology: an introduction. – Duke, 2010 – 182 с.

ЛОГИЧЕСКОЕ ПРОГРАММИРОВАНИЕ: ПРИНЦИПЫ И ПРИЛОЖЕНИЯ К РЕШЕНИЮ АКТУАЛЬНЫХ ЗАДАЧ

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

Хвалько Д. В.

Искра Н. А. – старший преподаватель

Логическое программирование радикально отличается от других парадигм программирования. Сегодня в связи с техническим развитием в области параллельных вычислений и BigData стало возможным эффективное практическое применение логического программирования.

Отличительной чертой логической парадигмы программирования является наличие кванторов существования - утверждений о том, что существуют некоторые объекты с заданными свойствами. При работе с этой парадигмой используется конструктивный метод доказательства целевых утверждений: если требуемое было доказано, то в процессе доказательства находятся элементы, соответствующие ранее не определённым объектам, которые, однако, входят в целевое утверждение. Это соответствие и создаёт результат вычислений. Буквально это можно воспринимать как два выражения: программа – множество аксиом, вычисление – конструктивный вывод целевого утверждения из программы (множества аксиом).

Парадигма логического программирования имеет следующие особенности:

1. В логических языках отсутствуют операторы присваивания и побочные эффекты.
2. Применяется естественная математическая модель вычислений.
3. Есть возможность возвратов и перебора.
4. В язык встроены возможности по представлению списков, деревьев и др. коллекций.
5. Высоко развиты возможности мета-программирования.

Аппарат логического программирования, как и первые языки, основанные на этой парадигме, были разработаны в шестидесятые годы двадцатого века. С тех пор, за неимением необходимых технологий и достаточных ресурсов, про эту парадигму почти забыли, но в наши дни, в след за появлением параллельных вычислений, облачных вычислений и больших данных, благодаря увеличению вычислительных мощностей компьютеров, парадигма логического программирования вышла на новый виток развития. Логическое программирование в наши дни позволяет частично решать проблемы неоднозначности человеческого языка и проблему обобщения знаний в задачах искусственного интеллекта и построения экспертных систем [1].

Языки логического программирования представляют собой формализованные языки, где программа есть описание требуемого решения в терминах математической логики, а решение задачи строится в процессе логического вывода по заданному описанию. Существует много подходов к разработке программ на логических языках программирования: например, с использованием индуктивной логики или с использованием ограничений.

Исторически первым языком логического программирования был язык Planner. Этот язык был основан на автоматическом выводе результатов из данных и заданных правил перебора вариантов (которые и назывались планом, отсюда и название языка). Planner использовался, например, для снижения требований к вычислительным ресурсам с помощью поиска с возвратом.

Немного позднее был разработан язык Prolog, который не требовал плана перебора вариантов и был, в некотором роде, упрощением языка Planner.

От языка Planner в свою очередь произошли такие языки, как Conniver, Popler, QA-4, и QLISP. Языки программирования Datalog, Mercury, Visual Prolog были выделены как подмножества языка Prolog. Существуют также альтернативные языки логического программирования, которые не используют поиск с возвратом, например, Ether.

В последние годы, Datalog, который является синтаксическим подмножеством языка Prolog для представления и обработки данных, вновь возник в центре внимания создателей широкого спектра новых приложений, в том числе в таких сферах как:

- интеграция данных;

- мониторинг компьютерных сетей;
- извлечение информации (Data Mining);
- защита информации;
- облачные вычисления [2].

Общим для всех этих сфер является использование языка Datalog как более высокого уровня абстракции данных в выполнении запросов и построении дедуктивных баз данных [3].

Список использованных источников:

1. Цуканова, Н.И. Теория и практика логического программирования на языке Visual Prolog 7/ Н.И. Цуканова, Т.А. Дмитриева // Международный журнал экспериментального образования, №2, 2012. – стр. 77.
2. Datalog and Emerging Applications: An Interactive Tutorial / Shan Shan Huang, Todd J. Green, Boon Thau Loo // SIGMOD'11. – Athens, 2011.
3. Катериненко, Р. Дедуктивные базы данных: построение продукционной модели с помощью современных СУБД / Lambert Academic Publishing, 2012.

СИСТЕМА ОЦЕНКИ КАЧЕСТВА АЛГОРИТМОВ КЛАССИФИКАЦИИ

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

Михалькевич Е. Ф.

Лукашевич М. М. − канд. техн. наук

Система предназначена для оценки качества различных алгоритмов классификации. Оценка может производиться с использованием различных методик и метрик оценки, а также с использованием любых входных данных, что позволяет подобрать алгоритм классификации под конкретную задачу.

Классификатор – это абстрактный автомат (математическая модель), принимающий решение об отнесении вектора информативных признаков к одному из классов. Предполагается, что информативные признаки объединяются по некоторому правилу в интегрированный параметр, по которому и принимается решение. Самым тривиальным решением является дуальное решение - ДА/НЕТ.
Для правильной работы классификатора его необходимо обучать и тестировать. Чтобы настроить классификатор под определенную задачу, необходима репрезентативная обучающая последовательность, по которой можно достаточно точно восстановить параметры классификатора. Обычно, для тестирования и обучения классификаторов, базы берутся в различных открытых репозиториях. Самым известным из них является UCI [1].

При решении любой проблемы связанной с машинным обучением в первую очередь производится анализ исходных данных, специфики требований к интеллектуальной системе, а также может строиться модель системы. Это все требуется для выбора методики классификации или кластеризации.

На сегодняшний день существует большое множество различных классификаторов. Каждый из них имеет свои сильные и слабые стороны. Поэтому основной проблемой является выбор классификатора под конкретную задачу или под конкретные данные.

Как известно, классификация не сводится к методам разделения наборов подмножеств в признаковом пространстве. Для заказчика важно не только получить алгоритм, реализующий (возможно с некоторыми ошибками) требуемое разделение классов, но и иметь оценку надежности решения поставленной задачи, т.е. знать, как часто данный алгоритм будет ошибаться при классификации вновь предъявляемых объектов. Ясно, что указанная оценка напрямую определяет качество решения поставленной задачи. На практике же дать такую обоснованную оценку часто оказывается затруднительным. [2]

Процесс обучения классификатора важен не менее чем построение модели. Хотя не существует общих алгоритмов обучения для всех моделей, используются схожие методы для отдельных методов классификации [4].

Предложенная система позволяет решить проблему подбора классификатора. Однако она может использоваться и для других приложений. Например, исследование влияния каждого из признаков на качество классификаторов и уменьшение размерности пространства признаков.

Система оценки качества классификаторов обладает гибкостью конфигурации тестирования алгоритмов. На сегодняшний день существует огромное количество классификаторов. Многие исследователи адаптируют и модифицируют алгоритмы для своего приложения [3]. Из всех классификаторов можно выделить несколько основных и наиболее используемых.

При работе с системой можно выбрать:

- Количество признаков образов;
- Несколько алгоритмов классификации для сравнения;
- Алгоритмы обучения для каждого алгоритма классификации;
- Критерии остановки обучения;