Отчет по домашнему заданию

Еремейкин Пётр
мНоД16 ТМСС

Постановка задачи:

- Для данных из UCI репозитория Mashroom Data Set рассмотреть различные варианты применения классификаторов, основанных на FCA. Данные: http://archive.ics.uci.edu/ml/datasets/mushroom
- Для каждого варианта классификатора оценить качество и производительность
- Проанализировать возможные усовершенствования FCA методов
- Разработать классификатор, позволяющий предсказывать класс гриба с высокой точностью (около 90%)

Способ тестирования классификаторов

Для тестирования классификатора применяется метод кросс-валидации, реализованный в файле `validator.py`. Как известно, при кросс-валидации все данные разбиваются на несколько частей, каждая из которых поочередно выступает в роли обучающей. Результат тестирования агрегируется при помощи классов `ScoreCounter` (для отдельного запуска) и `Aggregator` (для запусков по всем частям), описанных в файле `aggregator.py`.

Представление данных

Для заданного набора данных опробуем метод бинаризации, реализованный функцией `get_data` в файле `data_preparation.py`. Применение этого метода приводит к существенному увеличению размерности данных с [8124 x 23] до [8124 x 118]. Такой рост вызывает значительное снижение производительности, поэтому было принято решение отказаться от метода бинаризации и при последующем анализе рассматривать узорные структуры (pattern structures).

Возможность применения FCA методов, основанных на решетке понятий

Рассмотрим возможность построения решетки узорных понятий (pattern concepts) для (+) и (–) контекста в зависимости от размера. Алгоритм close-by-one реализован в файле `close_by_one.py`. Как показали эксперименты, время работы данной реализации имеет высокую скорость роста. На рисунке 1 показана зависимость времени работы алгоритма в секундах от числа объектов с описанием из 22 многозначных признаков. Уже при количестве объектов в несколько десятков станет невозможным построение решетки целиком. Таким образом, методы, основанные на решетке понятий в данном случае не применимы или требуют определенных упрощений.

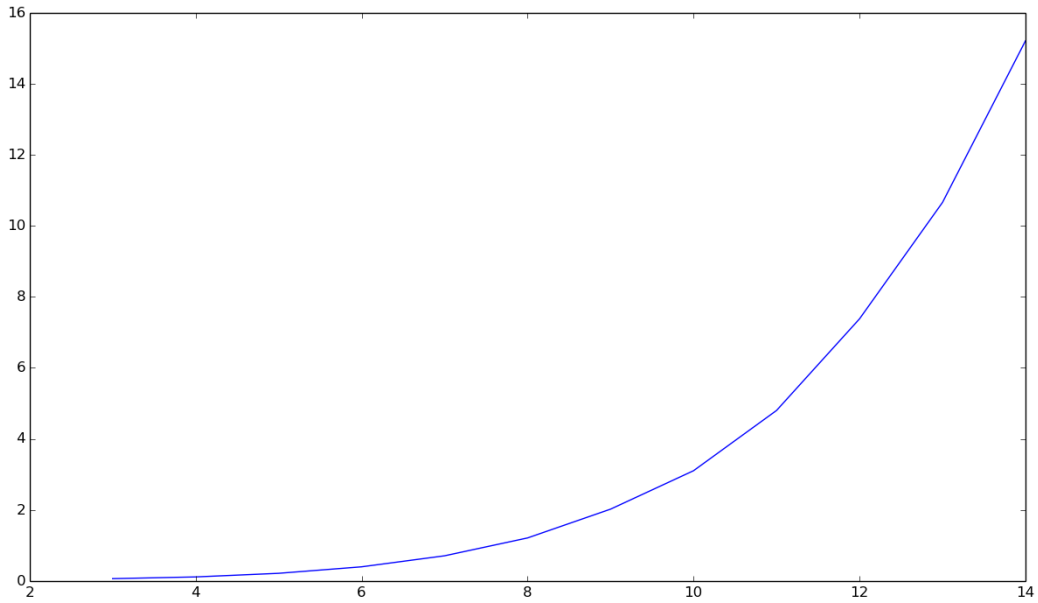

Рис 1.

Алгоритм ленивой классификации

Проанализируем характеристики одного из базовых алгоритмов, предложенных в задании: ленивая классификация с применением импликаций. Реализация данного алгоритма, адаптированная для работы с данными, сохраненными в `pandas.DataFrame` находится в файле `implication_classifier.py`. Применим для него кросс-валидацию. Уже при длине данных в 500 объектов (450 обучающих и 50 тестовых) алгоритм работает долго (полная кросс-валидация за 34 мин 23 сек или около 4 сек/объект) и показывает относительно невысокую точность предсказания (78.22% правильно классифицированных объекта при пороге равным 1). Результаты кросс-валидации каждой части:

p_p: 20 p_n: 0 n_p: 9 n_n: 20 u: 1 b: 0
p_p: 25 p_n: 2 n_p: 9 n_n: 14 u: 0 b: 0
p_p: 26 p_n: 2 n_p: 7 n_n: 14 u: 1 b: 0
p_p: 18 p_n: 0 n_p: 16 n_n: 16 u: 0 b: 0
p_p: 26 p_n: 1 n_p: 11 n_n: 12 u: 0 b: 0
p_p: 25 p_n: 1 n_p: 5 n_n: 19 u: 0 b: 0
p_p: 19 p_n: 2 n_p: 10 n_n: 16 u: 3 b: 0
p_p: 24 p_n: 0 n_p: 8 n_n: 18 u: 0 b: 0
p_p: 24 p_n: 0 n_p: 10 n_n: 16 u: 0 b: 0

По результатам видно, что кроме того, что точность предсказания классификатора мала, большую долю ошибок (91.40%) составляют n_p ошибки. Таким образом, в случае ошибки классификатора, несъедобный гриб может быть с большой вероятностью признан съедобным, что очень опасно. Из-за низкой производительности подбор порога потребует неоправданно много времени, тем более что при дальнейших исследованиях удалось найти намного более точный

и эффективный алгоритм. Далее рассмотрим возможные усовершенствования алгоритма.

Первое усовершенствование ленивой классификации

Первое рассмотренное усовершенствование – добавление кэша. Реализация этого усовершенствования приведена в файле `cached_implication_classifier.py`. Перед проверкой каждой гипотезы проверяется наличие результата в кэше, если он отсутствует, происходит вычисление результата. В данном случае удаление объектов из кэша не рассмотрено, так как общее число сохранённых результатов за все время валидации не велико, и предположительно скорость работы кэша, основанного на Python словаре, в среднем не зависит от его размера. Такой подход позволил сократить время работы при кросс-валидации до 23 мин 19 сек (или примерно до 3.1 сек/объект в среднем). Тем не менее такой результат также не удовлетворителен. Как и требовалось, точность предсказания после добавления кэша не изменилась.

Второе усовершенствование ленивой классификации

Второе рассмотренное усовершенствование – быстрая эвристическая проверка гипотезы. Это усовершенствование предлагает не определять фальсифицируемость гипотезы непосредственно по каждому объекту в противоположном контексте, а вычислять эвристическое значение, характеризующее возможность подтверждения гипотезы в противоположном контексте. Эвристическое значение вычисляется по агрегированному представлению контекста. Если вычисленное значение больше некоторого порога, считать гипотезу выполненной в противоположном контексте (т.е. сфальсифицированной), иначе гипотеза не выполнена. Остальная методика остается неизменной: для классифицируемого объекта для каждого объекта из соответствующего контекста строится гипотеза о принадлежности целевого объекта к классу, и эта гипотеза проверяется с помощью быстрой эвристической функции. По результатам проверки подсчитывается суммарное нормированное число выполненных гипотез и в зависимости от порога принимается решение о присвоении метки класса.

Эвристическая проверка осуществляется следующим образом: по каждой паре признак-значение в гипотезе суммируется частота данного значения относительно данного признака в соответствующем контексте. Полученная сумма нормируется на максимальную величину суммы для всех признаков гипотезы. Реализация приведена в файле `heuristics_classifier.py`.
Результаты кросс-валидации на 500 объектах (50 обучающих) приведены ниже:

p_p: 21 p_n: 0 n_p: 2 n_n: 26 u: 1 b: 0
p_p: 26 p_n: 0 n_p: 2 n_n: 19 u: 3 b: 0
p_p: 29 p_n: 0 n_p: 3 n_n: 15 u: 3 b: 0
p_p: 18 p_n: 0 n_p: 11 n_n: 20 u: 1 b: 0
p_p: 27 p_n: 0 n_p: 5 n_n: 16 u: 2 b: 0

p_p: 25 p_n: 0 n_p: 1 n_n: 23 u: 1 b: 0
p_p: 21 p_n: 0 n_p: 6 n_n: 20 u: 3 b: 0
p_p: 23 p_n: 0 n_p: 3 n_n: 22 u: 2 b: 0
p_p: 24 p_n: 0 n_p: 2 n_n: 21 u: 3 b: 0

Скорость существенно возросла относительно первой реализации (до 10 мин 35 сек на кросс-валидацию или 1.4 сек/объект). Средняя точность предсказания составила 88%, но все ошибки относятся к типу n_p.

Эвристический алгоритм классификации

Как показали эксперименты по предыдущему усовершенствованию, эвристическая оценка по агрегированному контексту может обеспечить не только меньшую скорость работы, но и более высокую точность предсказания. Рассмотрим полностью эвристический алгоритм.

Для классифицируемого объекта вычисляется сумма относительных частот по каждой паре признак-значение в соответствующем контексте. Объекту присваивается та метка, для класса контекста которого набрана больше сумма. Реализация приведена в файле `my_first_classifier.py`. Результаты кросс-валидации на 500 объектах:

p_p: 21 p_n: 0 n_p: 4 n_n: 25 u: 0 b: 0
p_p: 27 p_n: 0 n_p: 4 n_n: 19 u: 0 b: 0
p_p: 29 p_n: 0 n_p: 7 n_n: 14 u: 0 b: 0
p_p: 18 p_n: 0 n_p: 12 n_n: 20 u: 0 b: 0
p_p: 27 p_n: 0 n_p: 10 n_n: 13 u: 0 b: 0
p_p: 26 p_n: 0 n_p: 3 n_n: 21 u: 0 b: 0
p_p: 21 p_n: 0 n_p: 10 n_n: 19 u: 0 b: 0
p_p: 24 p_n: 0 n_p: 4 n_n: 22 u: 0 b: 0
p_p: 24 p_n: 0 n_p: 6 n_n: 20 u: 0 b: 0

Средняя точность предсказания составила 89.33% без настройки порога. Время выполнения кросс-валидации составила около 1.2 сек, что позволяет проводить тестирование на всех 8124 объектах. Результат кросс-валидации на всех объектах:

p_p: 395 p_n: 12 n_p: 68 n_n: 337 u: 0 b: 0
p_p: 415 p_n: 11 n_p: 62 n_n: 324 u: 0 b: 0
p_p: 397 p_n: 14 n_p: 76 n_n: 325 u: 0 b: 0
p_p: 412 p_n: 19 n_p: 59 n_n: 322 u: 0 b: 0
p_p: 406 p_n: 17 n_p: 65 n_n: 324 u: 0 b: 0
p_p: 404 p_n: 19 n_p: 68 n_n: 321 u: 0 b: 0
p_p: 406 p_n: 11 n_p: 67 n_n: 328 u: 0 b: 0
p_p: 407 p_n: 20 n_p: 73 n_n: 312 u: 0 b: 0
p_p: 403 p_n: 14 n_p: 69 n_n: 326 u: 0 b: 0
p_p: 402 p_n: 20 n_p: 77 n_n: 313 u: 0 b: 0

Точность предсказания: 89.64%, Время: 10.9 сек. Большинство ошибок типа n_p. Точность предсказания по результатам кросс-валидации на всех объектах не существенно отличается от точности предсказания по 500 объектам, поэтому

далее рассмотрим какое влияние окажут усовершенствования на выборке из 500 объектов. Результирующие характеристики подсчитаем на всех 8124 объектах.

Рассмотрим усовершенствование, связанное с “штрафованием” результата, если какое-то значение определенного признака, характерное для классифицируемого объекта вообще не встречается в контексте. Реализация приведена в файле `my_second_classifier.py`. Определим наилучшее значение штрафа с точки зрения точности. Зависимость точности предсказания от штрафа:

pen quality
1.0: 91.11
1.5: 94.00
2.0: 96.44
2.5: 97.55
3.0: 97.77
3.5: 97.77
4.0: 97.77
4.5: 97.77
5.0: 97.77

Наибольшая точность предсказания достигается при больших значениях штрафа, это значит, что лучше не применять штраф, а сразу возвращать противоположное значение если проявилась ситуация, в которой значение определенного признака, характерное для классифицируемого объекта вообще не встречается в заданном контексте.

Третья версия учитывает предыдущие усовершенствования и предлагает возводить в некоторую степень частоты значений признаков. Так как частоты принимают значение от 0 до 1, то при возведении в степень значения, близкие к 1 будут практически сохраняться, а маленькие значения около 0 практически обнуляться. Результат варьирования степени:

pow quality
1.0 : 98.44
1.5 : 98.44
2.0 : 98.44
2.5 : 98.44
3.0 : 98.44
3.5 : 98.44
4.0 : 97.77

Классификатор с возведением частот в степень описан в файле my_third_classifier.py. Как видно из результатов выше, возведение в степень не приводит к увеличению точности.

Рассчитаем параметры полученного классификатора (`my_classifier.py`) путем кросс-валидации по всем 8124 объектам.

Точность классификатора: Precision= p_p/(p_p+n_p)
Полнота классификатора: Recall = p_p/(p_p+p_n)
Сбалансированная F-мера: F = 2* Precision*Recall / (Precision + Recall)

Итерация	Precision	Recall	F
p_p: 407 p_n: 0 n_p: 1 n_n: 404 u: 0 b: 0	99.7	100.0	99.8
p_p: 426 p_n: 0 n_p: 0 n_n: 386 u: 0 b: 0	100.0	100.0	100.0
p_p: 411 p_n: 0 n_p: 4 n_n: 397 u: 0 b: 0	99.0	100.0	99.4
p_p: 431 p_n: 0 n_p: 3 n_n: 378 u: 0 b: 0	99.3	100.0	99.6
p_p: 423 p_n: 0 n_p: 2 n_n: 387 u: 0 b: 0	99.5	100.0	99.7
p_p: 423 p_n: 0 n_p: 2 n_n: 387 u: 0 b: 0	99.5	100.0	99.7
p_p: 417 p_n: 0 n_p: 0 n_n: 395 u: 0 b: 0	100.0	100.0	100.0
p_p: 427 p_n: 0 n_p: 2 n_n: 383 u: 0 b: 0	99.5	100.0	99.7
p_p: 417 p_n: 0 n_p: 3 n_n: 392 u: 0 b: 0	99.2	100.0	99.5
p_p: 422 p_n: 0 n_p: 2 n_n: 388 u: 0 b: 0	99.5	100.0	99.7
Средняя	99.52	100.0	99.7

В данном случае от классификатора более желательно свойство точности, так как требуется исключить ситуации классификации ядовитых грибов как съедобных. Тем не менее, точность полученного эвристического классификатора выше чем у классификаторов, основанных на FCA методах при существенно большей скорости работы.

Выводы

- Бинарное представление признаков зачастую может оказаться неприменимым для реальных данных.
- FCA методы классификации, основанные на решетке понятий (в т. ч. без применения бинаризации) работают за приемлемое время только на небольших объемах данных или требуют значительных усилий для их модификации.
- Для базового алгоритма классификации, основанного на импликациях, были рассмотрены некоторые улучшения, позволяющие сократить время классификации одного объекта с примерно 4 сек до 1.4 сек без потери точности.
- Предложенные улучшения все равно не позволяют добиться времени работы, пригодного для кросс-валидации по всем объектам и итеративного подбора параметров.
- В качестве альтернативы рассмотрен эвристический алгоритм, для которого были предложены улучшения и проведен подбор параметров.
- Итоговый классификатор работает намного быстрее и с большей точностью относительно рассмотренных FCA классификаторов.
- Таким образом, для конкретного случая стоит внимательно подойти к вопросу о выборе типа классификатора: более простое решение, без строгой теоретической основы может обеспечивать значительно лучший результат относительно сложного и математически

обоснованного. Однако, в некоторых случаях (например, для принятия решения о выдаче кредита) обоснованность может являться весомым критерием. В этом случае следует подробно изучить все возможные улучшения и тонкости реализации имеющихся математических методов.