ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ

НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Факультет компьютерных наук
Образовательная программа «Прикладная математика и информатика»

Отчет о программном проекте

на тему «Разработка системы предсказания успешного завершения учебной _____дисциплины» _____

Выполнил:
студент группы БПМИ185_ _____ _____А. И. Лежанкина_____
Подпись И.О. Фамилия
_____23.06.2020_____
Дата

Принял:
руководитель проекта_____Андрей Андреевич Паринов_____
Имя, Отчество, Фамилия
_____ м.н.с. МНУЛ ИССА ФКН НИУ ВШЭ_____
Должность Место работы

Дата ___23.06___2020 _____ _____
Оценка (по 10-тибалльной шкале) Подпись

Москва 2020

Содержание

Введение

Данный проект является результатом командной работы. Каждому члену команды был предъявлен алгоритм из области кластеризации и поиска ассоциативных правил.

Актуальность данной работы обоснована тем, что студент часто не может оценить сложность отдельной дисциплины, вследствие чего уделяет недостаточно времени на её изучение, поэтому система, помогающая предсказать вероятность того, что определённый студент завершит прохождение дисциплины успешно, оказывается достаточно полезной при обучении и распределении времени. Также можно добавить, что возможность кластеризовывать данные, объединяя их по определённым признакам, помогает более наглядно взглянуть на них и выявить некие взаимосвязи. Например, между оценкой и временем изучения предмета или названием дисциплины и успешностью её преподавания. То есть, такая система применима не только к базе данных студентов и их оценок, а к любой структурированной информации, где необходимо сравнить объекты, выбрать наиболее подходящий по каким-то параметрам. В современном мире информации становится всё больше, и человек нуждается в средствах упрощения её обработки и взаимодействия с ней. Чтобы собственноручно не сравнивать показатели в каких-то таблицах, гораздо удобнее предоставить эту работу программе, особенно, если она представлена через удобный веб-сервис.

Цель работы – разработка системы предсказания успешного завершения дисциплины или обработка информации о репетиторах.

Задачи:

- Первого этапа – Изучение и сравнение алгоритмов кластеризации. Реализация алгоритма Ward Hierarchical Clustering.
- Второго этапа – Проектирование своего RESTful API.
- Третьего этапа – Реализация алгоритма поиска ассоциативных правил Equivalence Classification Algorithm. Тестирование алгоритмов на базе данных репетиторов.

Объект исследования – алгоритмы кластеризации и поиска ассоциативных правил.

Предмет исследования – обработка представленной базы данных, используя упомянутые алгоритмы.

Обращаясь к терминам, необходимо пояснить краткую суть алгоритмов.

Кластеризация (или кластерный анализ) – это задача разбиения набора объектов на группы (кластеры). Идея заключается в том, что внутри одного кластеры должны находить наиболее схожие объекты, в то время, как в различных группах – строго отличные друг от друга объекты. Изначально число кластеров не задано, его оптимальное число определяется непосредственно в процессе работы алгоритма. Задача кластерного анализа принадлежит множеству задач обучения без учителя (unsupervised learning), то есть выполняется без вмешательства со стороны экспериментатора. Можно сказать, что кластеризация является хорошим способом обработки информации на начальных этапах взаимодействия с ней. Результат работы алгоритма наглядно структурирует данные и показывает общую тенденцию, чтобы человек мог далее проанализировать результат работы самостоятельно. Среди самых известных алгоритмов, а по совместительству среди тех, которыми занимается наша группа, можно перечислить OPTICS, PrefixSpan, SpectralClustering, the PAM Clustering Algorithm, Clara, Affinity propagation и Ward Hierarchical Clustering. Я занималась реализацией последнего, о котором подробнее будет написано ниже в отдельной главе.

Поиск ассоциативных правил (Associations rules learning или ARL) – метод определения взаимосвязей в базе данных. В основе работы алгоритма – анализ подмножеств (транзакций) – уникальных наборов элементов. Благодаря подобным алгоритмам поиска

утверждаются определённые правила попадания объектов в одну транзакцию. В дальнейшем, относительно введённых пользователем данных, они сортируются по своей «силе». Подобный метод обработки данных можно использовать в бизнесе, например в расположении продуктов на полках магазина. Самым известным примером применения этого алгоритма является случай[1] в 1992 году, когда группа компании Teradata под руководством Томаса Блишока исследовала 1,2 миллиона транзакций в 25 магазинах Osco Drug. Самым сильным выявленным правилом оказалась закономерность в покупке пива вместе с подгузниками в промежутке между 17.00 и 19.00. Опять же, подробнее об устройстве такого типа алгоритмов будет сказано в отдельной главе ниже. Я занималась реализацией алгоритма Equivalence Classification Algorithm (ECLAT).

Для реализации представленных задач использовался язык программирования Python.

Во время работы над проектом использовалось большое количество научно-практической литературы и технической документации, в частности различных сайтов и статей. Среди основных этапов работы можно выделить разработку и сравнение своего алгоритма с аналогичными алгоритмами других членов группы, выделение плюсов и минусов своего. Также необходимо было изучить базу данных, чтобы понять, относительно каких параметров её лучше классифицировать и обрабатывать.

1 http://www.dssresources.com/newsletters/66.php

Теоретическая часть
Ward hierarchical clustering

Теория

Данный вид кластеризации является восходящим. Изначально каждый студент – это отдельный кластер, затем объекты сливаются во всё более крупные кластеры: на каждом шаге выбирается пара кластеров с минимальным расстоянием друг от друга и объединяется. При этом данное расстояние хранится для последующего подсчёта оптимального числа кластеров. Объединение происходит до тех пор, пока все студенты не сольются в один кластер.

Структура данных

Для хранения и обработки данных использую структуру Непересекающихся множеств, которая позволяет легко объединять множества (кластеры), хранит информацию в виде списка родителей, где каждый i-й индекс соответствует i-ому студенты. Реализую функцию как класс с методами

- «find» – поиск родителя;
- «union_sets» – объединение множеств;
- «get_item» – возвращение элемента, соответствующего определённому индексу;
- «count_sets» – поиск числа кластеров;
- «count_elems» – поиск числа элементов в кластере;
- «result» – кластеры.

Формула расстояния

Формула для подсчёта расстояния между кластерам U и V. ρ – расстояние в Евклидовой метрике

- **Метод Уорда** (англ. *Ward's method*)

$$\mathrm{R}_{\mathrm{ward}}(U,V) = \frac{|U|\cdot|V|}{|U|+|V|}\rho^2\left(\sum_{u\in U}\frac{u}{|U|},\sum_{v\in V}\frac{v}{|V|}\right)$$

Оптимальное число кластеров

Изначально запускается функция кластеризации с единичным параметром на месте числа желаемых кластеров, при этом минимальное расстояние, найденное на каждом шаге алгоритма, записывается в список.

Далее мы проходим по образованному списку в поисках максимальной разницы между соседними расстояниями. Шаг, на котором будет найдено это расстояние, будет являться переменной I в формуле num = N – i + 1, которая отвечает за число кластеров, где N – число студентов.

Затем функция кластеризации запускается повторно, но уже с параметром num. Итоговое разбиение является ответом.

Входные данные

Хранятся вводимые данные в виде списка списков, где каждому студенту соответствует определённый индекс.
Среди характеристик полезны:

- Оценки за дисциплины;
- Количество часов, затраченных на освоение предмета;
- Идентификаторы посещения различных курсов или другие полезные скиллы (1 при наличии, 0 иначе).

Выходные данные

В итоге мы имеем набор кластеров оптимального размера. Данные выводятся в виде ключа – номера кластера и набора студентов, которые вошли в кластер. Пока в реализации это просто индекс, но в зависимости от ввода это могут быть имена студентов или набор характеристик.

Стандартизация

Так как данные могут иметь различный масштаб: например, оценки колеблются от 1 до 10, а число часов может доходить до сотни и более, будет полезно стандартизировать данные так, что их среднее значение будет равно 0, а отклонение 1. Для этого можно использовать StandartScaler и функцию fit_transform.

Equivalence Classification Algorithm (Eclat)

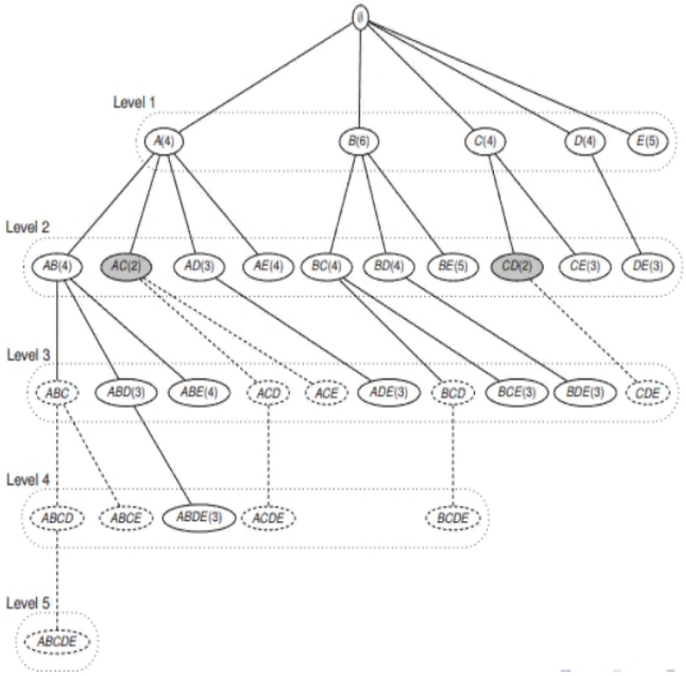
Теория

В отличие от других алгоритмов поиска ассоциативных правил данный использует поиск в глубину. Также в данном алгоритме база данных сканируется только единожды.

Для работы нам понадобится ввести несколько определений:

- Support (Поддержка) – идентификатор того, насколько часто встречается объект среди транзакций. Вычисляется, как число транзакций, содержащих те объекты, чью support мы ищем, делённое на количество транзакций. Измеряется в процентах (*100). Набор характеристик будет представлять интерес, если его Support выше, введённого ранее значения. Этот параметр (Минимальная поддержка) будет зависеть от того, с какой базой данных мы работаем, и сколько в ней студентов. На практике будет понятнее, какой размер минимальной поддержки делает работу эффективнее. Условно можно считать его равным 20%. Представленный тип хранения данных для алгоритма Eclat позволяет быстро рассчитывать поддержку. Суммарное число транзакций нам дано изначально, а число транзакций, в которых содержится наш объект item хранится как множество TID set. Чтобы посчитать число транзакций, в которых встречается несколько объектов, мы просто объединяем их и считаем мощность. Это можно сделать очень просто, если хранить данные в контейнере set.
- Confidence (Достоверность/Вероятность) – идентификатор корректности работы ассоциативного правила. Confidence(i | j) = Support(ij) / Support(i). Также необходимо задать какие-то ограничения максимальной достоверности, иначе правила окажутся слишком очевидными, и алгоритм будет не таким полезным. Мне кажется, можно зафиксировать этот параметр равным 60.

В ходе алгоритма мы поочередно просматриваем все возможные объединения объектов с некоторым ограничением. Сначала берем каждый объект сам по себе и считаем его поддержку, если она меньше заданной минимальной, отбрасываем его. Из оставшихся



образовываем пары, считаем поддержку, отбрасываем. Затем тройки, четверки и так до тех пор, пока число рассматриваемых объектов в объединении не превзойдёт максимальное

число характеристик для некоторого студента. Будет образовано подобное дерево при помощи поиска в глубину.

Входные данные

{item : TID set}, item – название объекта, TID set – множество номеров транзакций, содержащих данный объект item. В нашем случае объектами являются характеристики, относящиеся к студентам, а транзакциями, соответственно, наборы этих характеристик для каждого студента. Номер транзакции – номер студента в базе данных. Например, пусть исходная база данных содержит несколько студентов и набор характеристик для каждого. Характеристикой может быть оценка за дисциплину, затраченное на её прохождение число часов и т.д. Тогда для успешной работы алгоритма Eclat необходимо преобразовать каждую характеристику в объект item (Например, 10 по матанализу и 8 по программированию это два разных объекта-item). Номером транзакций будет являться номер студента, который имеет данную оценку среди своих характеристик.

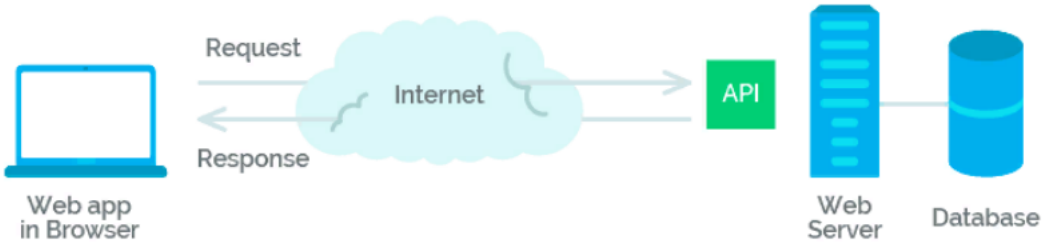
Данные удобно будет хранить в виде дерева префиксов, а также отдельно словарь со значениями Support и Confidence для каждого ключа-узла из образованного дерева. Это обеспечит наглядный показатель того, какие отношения будут образовывать правила.

Выходные данные

Если выводить Confidence от введённых пользователем собственных характеристик, то это будет вероятность получения желаемой оценки. Иначе можно вывести вероятный результат работы нашего ассоциативного правила для заданных характеристик, тогда студент получит наиболее вероятный результат своей деятельности.

Проектирование API

По завершении реализации алгоритмов необходимо было разработать собственный веб-API. Это полезно для спецификации структуры запросов и ответов между клиентом и сервером. API позволяет взаимодействовать пользователю с лежащей внутри базой данных благодаря паре уровней между ними – уровень абстракции данных и уровень менеджера ресурсов.



Теория

Для решения задачи проектирования API (application programming interface) использовался довольно стандартный подход REST (representational state transfer) – архитектура при дизайне веб-сервисов и веб-API. Важно, что в ней внутренняя реализация скрыта от клиента. Его принципы:

- клиент-серверная архитектура – существует разделение между клиентом, использующим сервер, и самим сервером. Клиенты взаимодействуют с сервером, обмениваясь представлениями ресурсов;
- любые данные – ресурс (унификация) – программный интерфейс сервера стандартизирован;
- любой ресурс имеет ID – универсальный код ресурса URI, который однозначно идентифицирует этот ресурс;
- ресурсы связаны между собой;
- используются стандартные методы (запросы) протокола HTTP – они должны быть независимы друг от друга и могут задаваться в любом порядке. Для каждого запроса существует свой статус, необходимый для грамотного оформления ответа и предоставления его клиенту;
- кэшируемость;
- сервер не хранит все сессии в памяти – никакие данные клиента не сохраняются на сервере.

Этот подход позволяет предоставить данные для пользователя в понятном виде. Основная идея заключается в том, что каждый запрос на сервер преобразует приложение клиента. RESTful API – практическое воплощение архитектурного стиля REST. При разработке RESTful веб-сервиса у нас имеется некоторый список url-адресов, с помощью которых можно с ним взаимодействовать.

Поподробнее о методах HTTP. Зачастую каждый из методов служит для выполнения какого-то действия из концепции программирования CRUD (create, read, update, delete). Всего их существует 7: OPTIONS, GET, HEAD, PUT, POST, DELETE, PATCH. Основными являются только 4:

- GET (read) – получает информацию о ресурсе в удобном для пользователя формате. Возможно применение как для полного списка ресурсов (в нашем случае для всех экспериментом), так и для какого-то конкретного. Не меняет состояние ресурса;
- POST (create) – создаёт новый ресурс;
- PUT (update) – обновляет ресурсов. Изменяет состояние ресурса, но допускается неоднократное повторение;
- DELETE (delete) – удаляет ресурс. Также изменяет состояние.

Для взаимодействия с пользователем в стандарте HTTP существует около 70 статус кодов. Это информация о статусе завершения операции. Нужно заметить, что ошибки из серии – 4xx client error – должны отлавливаться, обрабатываться, но их нельзя возвращать в ответ пользователю. В моей реализации применялось всего 5:

- 200 – OK – успешный запрос;
- 201 – OK – в результате запроса был создан новый ресурс (используется в методе POST);
- 204 – OK – ресурс удалён (в методе DELETE);
- 400 – Bad Request – невалидный запрос;
- 404 – Not found – ресурс не найден (в методах GET, PUT и DELETE).

Реализация и тестирование

Стоит начать с того, что ввиду некоторых проблем с получением прав доступа для работы с базой данных студентов, как изначально планировалось, все тестирования проводились с базой данных репетиторов[2], поэтому цели проекта немного изменились. Необходимо было кластеризовать представленный датасет, относительно предмета и оценки; разработать API для работы с алгоритмом кластеризации; с помощью алгоритма поиска ассоциативных правил исследовать базу данных и установить взаимосвязь между словами в отзывах клиентов.

Основная часть проекта написана на языке Python и представлена в едином Jupyter Notebook. Для работы использовались библиотеки pandas, numpy, flask, flask_restful и matplotlib.

Обработка входных данных

База данных находилась в открытом доступе и изначально представлена в виде xlsx файла. С помощью библиотеки pandas данные преобразовывались в формат DataFrame. Оценка работы репетиторов именовалась словесно, то есть «лучше не бывает; отлично; почти хорошо» и тд. Для удобства работы такие обозначения переводились в числовой формат оценки от 1 до 10:

- 'лучше не бывает!': 10;
- 'отлично': 9;
- 'почти отлично': 8;
- 'вполне хорошо': 7;
- 'хорошо': 6;
- 'почти хорошо': 5;
- 'более-менее': 4;
- 'так себе...': 3;
- 'плохо': 2;
- 'ужасно': 1

Название дисциплины так же переводилось в числовой вид, соответствующий порядковому номера встречи той или иной дисциплины.

Всего в выбранной базе данных обнаружено 288724 объекта, но нужно заметить, что алгоритмы слишком долго обрабатывали базу целиком, поэтому работа велась с урезанной таблицей с определённым шагом, вследствие чего, результат работы является неточным, но приблизительным. Так как оба алгоритма направлены на выявление общей тенденции, то нет необходимости в излишней детализации, и сокращённая база данных достаточна для предоставления необходимой информации и последующего анализа.

Кластеризация

На практике реализованный алгоритм кластеризации Ward Hierarchical Clustering оказался достаточно долго работающим, так как по сути работает дважды, чтобы определить оптимальное число кластеров, поэтому он применялся к урезанной базе данных.

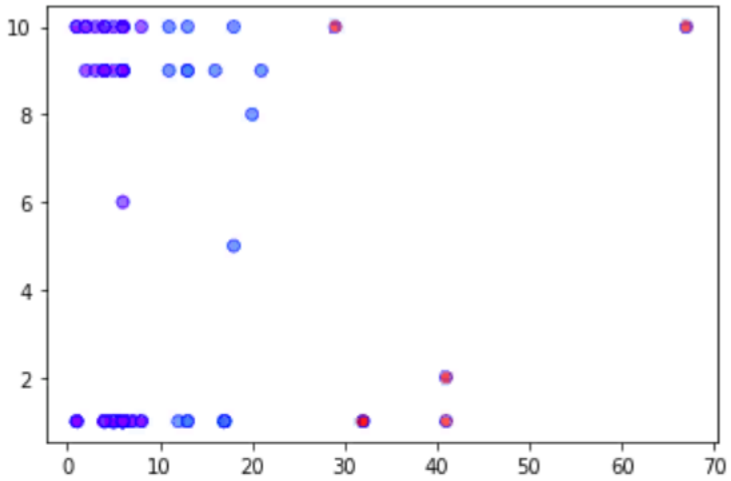
Особенных обработок данных перед применением алгоритма не понадобилось. В качестве аргументов использовались столбцы с оценками деятельности репетиторов, а также с номером дисциплины. Если бы для работы применялась предполагаемая база данных со студентами, то использовалось бы большее число столбцов с информацией об оценках, тогда наблюдение за различиями принадлежности объектов кластерам было бы интереснее, чем

2 Данные собраны с сайта https://repetitors.info/

сейчас.

В результате применения алгоритма к каждому тысячному репетитору имеем следующее разделение объектов на кластеры и соответствующую визуализацию:

```
Кластер №0: 0 1 3 4 5 6 8 10 11 12 15 16 17 18 20 21 22 23 24
28 29 30 31 32 33 35 36 37 38 39 40 41 43 44 45 46 47 48 49 5
1 52 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75
76 77 78 79 82 83 84 86 88 89 90 91 92 95
Кластер №1: 2 7 9 19 25 26 27 34 42 53 54 55 56 85 87 94 96
Кластер №2: 13 14 50 80 81 93
```



На представленной схеме на оси X отмечены номера дисциплин, а на оси Y оценки за их преподавание. Судя по результатам работы можно сказать, что на принадлежность объекта кластеру больше влиял номер дисциплины, о котором идёт речь, а не оценка. То есть дисциплины с номерами от 0 до 10 принадлежат первому кластеру, от 10 до 30 – второму, от 30 и далее – третьему. Алгоритм сам определил, что оптимальное число кластеров – 3. Добавлю, что при тестировании на базе данных разного размера, алгоритм всё время предлагал разделение на 3 кластера.

RESTful Api

При практической реализации использовались: Python3, Flask, Flask-RESTful – расширение для Flask. Помимо Flask также предлагались такие фреймворки как Eve или Django, но выбранный оказался наиболее удобным и обладающим обширной и подробной документацией.

В нашем случае ресурсом является объект JSON, суммарно представляющий из себя базу данных. Это довольно лёгкий и понятный для работы формат. Процесс заключался в обработке JSON файла: на сервер передаётся информация о студентах: идентификатор, год поступления, пол, город; информация об оценках: идентификатор студента, идентификатор оценки, название дисциплины, год получения оценки, дата получения, тип, оценка Важно заметить то, что идентификатор студента и оценки не изменяется даже при изменении самого объекта, он сохраняется постоянным. На выходе пользователь получает идентификаторы студента и эксперимента и кластер, к которому отнесён студент.

Flask – это библиотека Python, представляющая из себя фреймворк для создания веб-приложений. Она идеально подходит для быстрого и качественного проектирования благодаря простому, но легко расширяемому функционалу.

Из расширения flask_restful нам потребуются классы Api и Resourse, а также интерфейс парсинга запросов reqparse.

В моей реализации использовалось 3 HTTP-метода. Все они принимают на вход идентификатор эксперимента и работают с алгоритмом Ward Hierarchical clustering. Подробнее о методах. Они применялись как для целого списка экспериментов, так и для конкретных с определённым идентификатором. Для списка:

- GET – возвращает список экспериментов;
- POST – добавляет новый эксперимент;
- DELETE – удаляет эксперимент.
  Для отдельного эксперимента:
- GET – возвращает результат эксперимента;
- POST – запускает алгоритм с полученными параметрами.

Поиск ассоциативных правил

Сначала стоит сказать, что представленная база данных с информацией о репетиторах не лучшим образом подходит для тестирования данного алгоритма Equivalence Classification Algorithm., так как здесь не достаточно фич, взаимосвязь между которыми можно найти. Возможный вариант использования данного алгоритма заключается в работе с комментариями к каждому репетитору. Столбец с текстом разделялся на отдельные слова, таким образом преобразовываясь в список слов. Алгоритм запускался для столбца с названием дисциплины, изменённым на числовое значение, и для обновлённого столбца с текстом. Суть заключалась в поиске взаимосвязи между словами, например, если речь идёт об определённом предмете, и ученик занимался конкретное число часов, то впечатление от занятий было «хорошо» или наоборот «плохо».

Так же как алгоритм кластеризации, данный алгоритм долго работал на цельной базе данных и её пришлось обрезать.

Ввиду того, что алгоритм направлен на работу с оптимизированным датасетом, база данных, представленная в проекте, не позволила получить обширных наглядных результатов, было найдено небольшое количество правил, которые являются логичными и могут хоть что-либо отразить. Алгоритм показал:

	word	counter
0	-	6
1	балла	196
2	викторовичем	196
3	всем	33
4	грамотный	196
5	доходчиво	33
6	егэ	201
7	замечательный	33
8	занимались	196
9	интересно	69
10	как	33
11	материал	1

```
('балла',) ==> ('преподаватель',) support: 2 confidence: 100.0

('балла', 'грамотный') ==> ('на',) support: 2 confidence: 100.0

('балла', 'грамотный') ==> ('очень',) support: 2 confidence: 100.0

('балла', 'грамотный') ==> ('преподаватель',) support: 2 confidence: 100.0
```

Таким образом, задачу поиска ассоциативных правил в данных условиях можно свести к взаимосвязи экзамена, впечатлений от занятий или результата и предмету, по которому он сдавался. Здесь следует обратить внимание на предобработку текста, так как изначально среди задач проекта этого не было, то на это не было затрачено достаточное количество времени, чтобы результаты были показательными, не учитываются различия в падежах, числе и регистре. Также алгоритм пока выявил много «мусора», то есть правила относительно местоимений, союзов и прочих вспомогательных частей речи.

Результат работы алгоритма представлялся в виде прикрепленного выше «следствия» наличия определённых «items» в транзакции, а также непосредственного списка наиболее часто встречающихся транзакций. По поводу support и confidence объяснялось в теоретической части, но можно пояснить, что для данной задачи минимальное значение support и необходимое confidence подбиралось экспериментально, пока не было найдено подходящее число правил. Minsupport = 2, confidence = 60.

Заключение

В результате работы над проектом были получены следующие результаты:

- Реализован и протестирован алгоритм кластеризации Ward Hierarchical Clustering;
- Разработан индивидуальный RESTful API для работы с алгоритмом кластеризации;
- Реализован и протестирован алгоритм поиска ассоциативных правил Equivalence Classification Algorithm.

В ходе работы было изучено большое количество научной литературы и технической документации, опробовано множество различных библиотек, использована клиент-серверная архитектура JSON API, исследована философия RESTful API.

На данный момент работа над проектом продолжается. В первую очередь, необходимо опробовать алгоритмы на базе данных с информацией о студентах, а для этого получить к ней доступ. И продолжить создание веб-сервиса. Таким образом, уже будет готова заявленная система успешного предсказания дисциплины.

Также можно продолжить модифицирование алгоритма поиска ассоциативных правил для уже разобранного датасета. Предлагается усовершенствовать обработку слов в комментариях: отсеивать очевидно ненужные слова, в виде местоимений, междометий и тп, учитывать падежи, чтобы сократить и оптимизировать правила.

Использованные материалы

Ward hierarchical clustering

- Ward's Hierarchical Clustering Method: Clustering Criterion and Agglomerative Algorithm https://www.researchgate.net/publication/51962445_Ward's_Hierarchical_Clustering_Method_Clustering_Criterion_andAgglomerative_Algorithm
- Техническая документация https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html
- «Hierarchical Clustering with Python and Scikit-Learn» https://stackabuse.com/hierarchical-clustering-with-python-and-scikit-learn/

Equivalence Classification Algorithm (Eclat)

- International Journal of Computer Techniques -– Volume 2 Issue 3, May – June 2015 FREQUENT ITEMSET MINING USING ECLAT WITH RELATIVE PROFIT AND PRICE http://oaji.net/articles/2015/1948-1434984559.pdf
- Основная теория https://www.geeksforgeeks.org/ml-eclat-algorithm/
- «Поиск ассоциативныхх правил» https://habr.com/ru/company/ods/blog/353502/
- Псевдокод и пример реализации http://mlwiki.org/index.php/Eclat

Api

- «Пишем API на Python» https://temofeev.ru/info/articles/pishem-api-na-python-s-flask-i-rapidapi/
- Проектирование RESTful API с помощью Python и Flask https://habr.com/ru/post/246699/
- How to build JSON API with Python https://www.freecodecamp.org/news/build-a-simple-json-api-in-python/
- Руководство по проектированию API https://docs.microsoft.com/ru-ru/azure/architecture/best-practices/api-design
- Подходы к проектированию RESTful API https://habr.com/ru/company/dataart/blog/277419/
- Лучшие практики проектирования REST API https://jazzteam.org/ru/technical-articles/restful-services-manual/

Приложение

- База данных репетиторов https://repetitors.info/
- Репозиторий https://github.com/LezhankinaAI/Project2020