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 БПМИ186 _____ Болотин Арсений Алексеевич
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

Аннотация

Данная работа выполнялась, как командный проект, нашей целью было создание системы с помощью которой можно предсказывать успешность завершения учебной дисциплины по данным студента. Для этого были изучены различные методы кластеризации и поиска ассоциативных правил. В данной работе описаны алгоритм кластеризации Affinity Propagation и алгоритм поиска ассоциативных правил PrePost. Также был подготовлен REST API для работы с удаленной базой данных. К сожалению, в ходе нашей работы нам не удалось получить доступ к базе данных студентов ВШЭ, однако оба алгоритма были успешно имплементированы, настроено взаимодействие с API, а также алгоритмы были протестированы на оригинальном датасете репетиторов, собранного с сайта https://repetitors.info/.
Ключевые слова: Алгоритмы кластеризации, Алгоритмы поиска ассоциативных правил, Affinity Propagation, PrePost, REST API.

Содержание

1 Введение

Целью нашего проекта было создание системы предсказания успешного завершения учебной дисциплины обучающимися. Цель обусловлена довольно естественными причинами - если заранее понять, что у студента могут быть проблемы с освоением предмета, то на это можно повлиять. В частности, у нас на факультете есть дополнительные адаптационные факультативы, но как понять надо ли их посещать первокурснику или нет. В этом могла бы помочь такая система.
Заранее стоит сказать, что систему предсказания, как планировалось изначально создать не удалось, в связи с тем, что база данных студентов ВШЭ была не доступна. Однако, свою систему мы тестировали на датасете репетиторов, собранного с сайта https://repetitors.info/.

1.1 Задачи

- Описание алгоритмов Affinity Propagation и PrePost
- Реализация Affinity Propagation и PrePost
- Создание RESTful API для работы с удаленной базой данных
- Тестирование алгоритмов на датасете репетиторов
- Тестирование RESTful API

1.2 Алгоритмы кластеризации и поиска ассоциативных правил

Алгоритм кластеризации - алгоритм, разбивающий выборку данных на сравнительно однородные группы.
Алгоритм поиска ассоциативных правил - алгоритм для обнаружения интересующих нас связей между переменными в большой базе данных.

1.3 Мотивация использования алгоритмов

Для данного проекта вполне разумно использование алгоритмов кластеризации и поиска ассоциативных правил. Таким образом для нового ученика мы могли бы отнести его к какому-то кластеру, и оценить насколько студенты из данного кластера справляются с соответствующей дисциплиной. В случае же поиска ассоциативных правил, можно понять, что если часть ассоциативного правила выполняется, то скорее всего выполнится и другая часть ассоциативного правила. Также алгоритмы поиска ассоциативных

правил могут выявлять какие-то общие проблемы, а не для конкретного ученика. Например, вполне вероятно можно обнаружить правило вида (студент живёт в общежитии; лекции по предмету первой парой; низкая оценка) или (студент работает; низкая посещаемость).
В команде было разобрано и реализовано большое количество алгоритмов, некоторые из которых были крайне специфичны. В частности, выбранный мной алгоритм PrePost, по запросу в гугле либо выдаст статью авторов алгоритма, либо сайт, на котором только ссылка на данную статью.

1.4 Инструменты

Алгоритмы были реализованы на языке Python, а API было сделано с помощью фреймворка Flask и двух дополнений к Flask: Flask-REST-JSONAPI и Flask-SQLAlchemy.

2 Алгоритм Affinity Propagation

2.1 Использование алгоритма

- Одной из главных особенностей алгоритма является то, что он не требует указывать количество кластеров на которые надо разбить данные.
- Также у каждого кластера в результате алгоритма есть представитель, являющийся точкой из изначальных данных.
- Датасет не очень большой ($N \leq 10^6$) или в меру большой, но разреженный ($N \leq 10^7$)
- Алгоритм является детерминированным, занимает $O(N^2)$ памяти и имеет сложность $O(N^2 * MAX\_ITERATIONS)$ по времени.
- В различных источниках утверждается, что алгоритм успешно применялся при сегментировании изображений, выделении групп в геноме, группировании фильмов на Netflix, моделировании экономических процессов.

2.2 Шаги алгоритма

1. Заполним Similarity Matrix для входных данных алгоритма.
2. Заведем матрицы R (Responsibility) и A (Availability). A = R = 0
3. Обновляем матрицы R и A поочередно по формулам $MAX\_ITERATIONS$ раз. Обновление может быть прервано, если изменения стали меньше фиксированного значения.
4. Заводим Criterion Matrix: C = A + R.
5. В каждой строке матрицы C индекс максимального значения соответствует представителю данной точки. (Соответственно точки с одинаковым индексом максимального значения отнесем к одному кластеру)

2.3 Similarity Matrix

$S(i, j)$ – такая функция, что :

- при i != j : $S(i, j) = -1 *$ (расстояние между точками i и j)^2

- при i == j : Элементы на диагонали заполняются одним значением. От выбранного числа зависит количество кластеров, которые получатся. Если выбрано маленькое число, то получится маленькое число кластеров и наоборот. Чаще всего число выбирается минимальным или медианой из уже посчитанных S(i, j), i != j.

Примечание: за S(i, j) может быть взята любая функция такая, что S(i, j) > S(i, k), если i ближе к j чем к k. Чаще используется именно минус квадрат расстояния между точками.

2.4 Responsibility Matrix

Responsobility Matrix рассчитывается по формуле :

$$r(i,k) \leftarrow s(i,k) - \max_{k' \text{ such that } k' \neq k} \{a(i,k') + s(i,k')\}$$

2.5 Availability Matrix

Availability Matrix рассчитывается по формулам :

1. Элементы на диагонали - $$a(k,k) \leftarrow \sum_{i' \text{ such that } i' \neq k} \max\{0, r(i',k)\}$$

2. Элементы вне диагонали - $$a(i,k) \leftarrow \min\left\{0, r(k,k) + \sum_{i' \text{ such that } i' \notin \{i,k\}} \max\{0, r(i',k)\}\right\}$$

2.6 Интуитивное объяснение

- Функция S (как следует из названия) - функция схожести 2 точек.
- Матрица R - матрица ответственности, насколько одна точка "хочет чтобы ее представителем была другая точка.
- Матрица A - матрица доступности, насколько одна точка готова быть представителем другой точки.

Представители - это итоговые центры кластеров, как обсуждалось ранее алгоритм центром кластера выбирает одну из изначальных точек.
Таким образом, алгоритм организует так называемые "обмен сообщениями"между точками, из-за чего и меняются соответствующие матрицы.

2.7 Оптимизации

Можно ввести ограничение на число вершин в дереве:

- В матрицу S добавляют шум для предотвращения вычислительных осцилляций в случаях, когда есть несколько хороших разбиений на кластеры
- При обновлении R и A используется не простое присваивание, а присваивание с экспоненциальным сглаживанием. Авторы советуют использовать параметр сглаживания со значением от 0.5 до 1 (по умолчанию 0.5)
- Часто AP нуждается в постобработке — дополнительной кластеризации лидеров групп. Подходит любой другой метод, может помочь дополнительная информация о задаче.

3 Алгоритм PrePost

3.1 Постановка задачи

- Вход - набор транзакций T = T1, T2 ... (название из-за применимости в поиске ассоциативных правил в чеке клиентов магазинов) и число eps.
- Выход - Набор itemset'ов, которые часто встречаются в транзакциях.

3.2 Основные понятия

- I - множество i1, i2 ... , состоящие из всех встречаемых элементов транзакций.
- eps - число, поданное на вход алгоритму, по которому мы будем определять встречается ли правило часто или нет.
- itemset : некоторое подмножество I.
- $supp(X) = \frac{|\{t \in T; X \in t\}|}{|T|}$, иными словами - отношение числа транзакций, содержащих множество X, ко всем транзакциям.

  X будем считать часто встречаемым, если supp(x) >= eps.

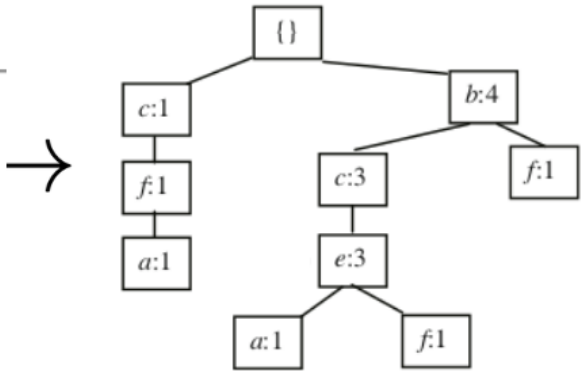
3.3 Шаги алгоритма

1. Построение PPC-tree
   (a) Построение FP-tree
   (b) Подсчет pre-order и post-order для каждой вершины дерева.
2. Формирование из дерева NL1
3. Формирование из NL1 NL2
4. Формирование NL-K
5. На каждом шаге, сохраняем часто встречаемые itemsets, для которых supp(x) >= eps.

3.4 FP-tree

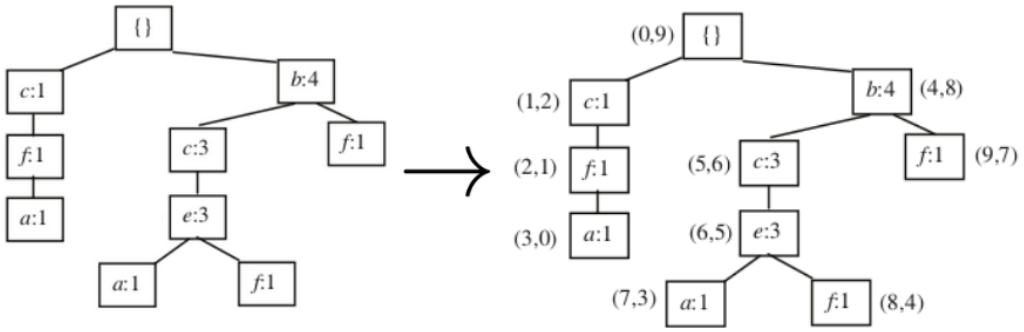
Сначала подсчитаем сколько раз каждый из элементов встречается в транзакциях. Отбросим все элементы, которые встречаются меньше $\epsilon * |T|$ раз. Затем внутри каждой транзакции отсортируем оставшиеся элементы по убыванию частоты. После этого строится префиксное дерево по полученным "словам" из транзакций. Также подсчитаем для каждой вершины количество слов, проходящих через неё.
Пример построения FP-Tree по траназкциям:

Items	Ordered frequent items
a, c, g, f	c, f, a
e, a, c, b	b, c, e, a
e, c, b, i	b, c, e
b, f, h	b, f
b, f, e, c, d	b, c, e, f



3.5 PPC-Tree

PPC-Tree - является модификацией FP-Tree. После построения FP-Tree подсчитываются pre-order и post-order для каждой вершины дерева. Pre-order - порядок обхода вершин: вершина, левое поддерево, правое поддерево. Post-order - порядок обхода вершин: левое поддерево, правое поддерево, вершина. Pre-order и post-order легко подсчитываются с помощью любого обхода графа(BFS, DFS).
Пример построения PPC-Tree из FP-Tree:



3.6 NL-1

NL1 для элемента $i \in I$ назовем множество всех вершин в дереве, которые хранят данный символ, представленные в виде троек:
<(pre-order вершины, post-order вершины) : кол-во заканчивающихся слов в данной вершине>. (<x , y> :z)

3.7 NL-k

Пусть мы хотим построить NL-k для подмножества I мощности k - i1, i2, .. ik, в котором выполнено $supp(i)1 \geq supp(i2) \geq ... \geq supp(ik)$. Тогда NL-k определяется рекурсивно через NL-(k-1) следующим образом:

- Пусть NL(K-1) для i2, i3, .. ik (P1 = <x11, y11> : z11, ...<x1m, y1m> : z1m) и NL(K-1) для i1, i3, .. ik (P2 = <x21, y21> : z21, ...<x2n, y2n> : z2n)
- $\forall i \in [1..m] \forall j \in [1..n]$, если <x1i, y1i : z1i> является предком <x2j, y2j: z2j>, то положим в NL-k <x1i, y1i : z2j>

Если в NL-k для подмножества IS = i1, i2, .. ik просуммировать zi, то мы получим количество транзакций в которых встречается itemset IS. Таким образом мы будем узнавать часто встречаемые itemset, сравнивая supp(IS) с $\epsilon * |T|$. А это то, чего мы и хотели добиться.

3.8 Способ нахождения NL-k

Теперь обсудим как же находить NL-k. Мы будем находить их все поуровнево, то есть сначала найдем все NL-1, затем все NL-2 и т. д. Пусть мы нашли все NL для itemsets размера меньших k, и теперь хотим найти все NL-k. Это задача разбивается на два случая.
1 случай. k = 2. Заведем квадратную таблицу размером T, равным количеству частых NL-1.Для каждой вершины N из PPC-tree и для каждого предка этой вершины Na, T[N][Na] += (кол-во слов, проходящих через N; это число записано в вершине N PPC-tree). Затем для каждой пары элементов i1, i2 добавим NL-2 от itemset i1, i2, только если $T[i1][i2] \geq \epsilon * |T|$.
2 случай. k > 2.
Пусть L_{k-1} - список частых itemsets мощности k-1, отсортированных по убыванию supp этих itemsets. Тогда для каждой пары из L_{k-1}, для которых совпадают все элементы, кроме первых построим NL-k. Пусть элементы в паре были $xx_2...x_{k-1}$ и $yx_2...x_{k-1}$ и $supp(x) \geq supp(y)$, а их NL-(k-1) были P1 и P2 соответственно. Тогда будем итерироваться 2 указателями по P1 и P2.

Если элемент из P1 является предком элемента из P2, то добавляем в NL-k <x1i, y1i : z2j>, как это и обсуждалось ранее, иначе если x1i > x2j то сдвигаем второй итератор, а если y1i < y2j то сдвигаем первый итератор. После добавления в NL-k всех вершин, подсчитываем общий supp($xyx_2...x_{k-1}$) и если он $\geq \epsilon * |T|$, то добавляем $xyx_2...x_{k-1}$ в L_k.

4 REST API

4.1 Что такое REST?

REST (Representational State Transfer — передача состояния представления) - согласованный набор архитектурных принципов для создания более масштабируемой и гибкой сети.
Основные принципы:

- Клиент-серверная архитектура
- Любые данные — ресурс
- Используются стандартные методы HTTP (GET, POST, PUT, DELETE)

4.2 Инструменты реализации

Реализация REST API была написана с помощью фреймворка Flask, а точнее с помощью двух дополнений к Flask: Flask-REST-JSONAPI и Flask-SQLAlchemy. Flask-REST-JSONAPI поможет нам в созаднии RESTful API с поддержкой JSON. Flask-SQLAlchemy использована для работы с базами данных.

4.3 Менеджер Ресурсов

Как уже было сказано выше, один из принципов REST: "Данные - это ресурс". Для обеспечения такого принципа использовались 2 класса абстракции - Experiment и Parameters, для которых были поддержаны HTTP методы GET и POST.
Experiment:

- GET: возвращает результат конкретного эксперимента по id
- POST: запуск конкретного алгоритма по id

Parameters:

- GET: возвращает словарь всех экспериментов, ключ - id эксперимента, значение - параметры эксперимента
- POST: установка параметров эксперимента

В результате получаются следующие URL:

- http://127.0.0.1:5000/get_ids (Parameters-GET)
- http://127.0.0.1:5000/experiments/{exp_id}/set_params?{params} (Parameters-POST)
- http://127.0.0.1:5000/experiments/{exp_id}/get_results (Experiment-GET)
- http://127.0.0.1:5000/experiments/{exp_id}/start (Experiment-POST)

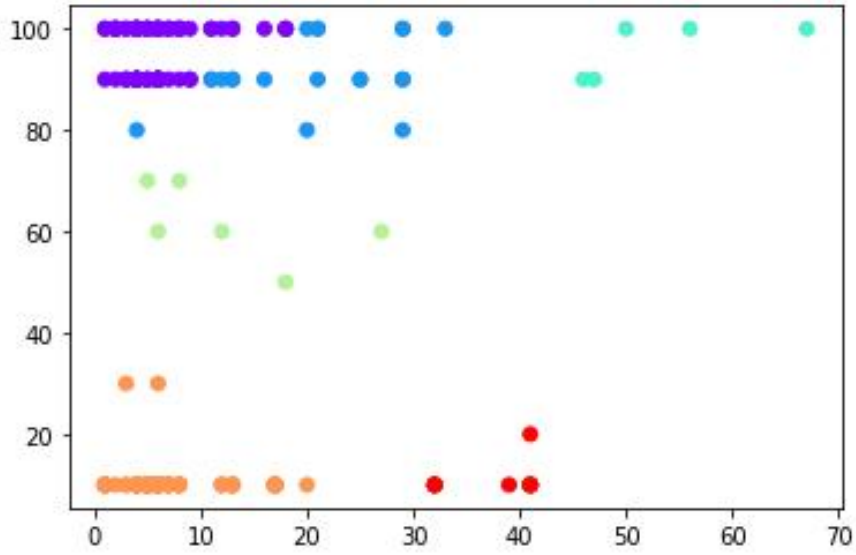
5 Результаты

5.1 Реализации

Описанное API и алгоритмы Affinity Propagation и PrePost были реализованы для дальнейшей работы с данными.
С результатами можно ознакомиться на гитхабе (Имплементация алгоритмов и реализация API):
https://github.com/ArseniyBolotin/AP-PrePost

5.2 Алгоритмы

Для тестирования алгоритмов был использован датасет оценки репетиторов и отзывов по ним. Датасет насчитывает порядка 300.000 записей.
В датасете представляли интерес только три поля: оценка, предмет, отзыв о репетиторе. В алгоритме кластеризации были использованы первые 2 поля, а для алгоритма поиска ассоциативных правил я парсил отзыв в набор слов и искал ассоциативные правила по ним.
Результаты тестирования алгоритмов на датасете репетиторов можно найти в соответствующих файлах на гитхабе - AffinityPropagationRepetitors.ipynb и PrePostRepetitors.ipynb соответственно.
Как было ранее отмечено, Affinity Propagation работает с не очень большими датасетами, поэтому для тестирования пришлось работать с меньшим размером поддатасетом, равномерно порезав данный. Связано это в первую очередь с тем, что алгоритм требует квадрат памяти. Также оценки и нумерацию предметов пришлось отмасштабировать. Результаты работы кластеризации для оценки и предмета:



Чтобы кластеризация была более целесообразной требуется поразбираться с отзывом, я добавил новые параметры есть ли слово ЕГЭ и ОГЭ в отзыве, а также исключим предмет из параметров кластеризации, так как это все таки не количественный параметр. Результат добавлен на гитхаб (AffinityPropagationRepetitorsResults.csv)
Алгоритм PrePost достаточно быстро работал, полный список ассоциативных правил можно опять же увидеть на гитхабе, не может не радовать правило - ('ужасно', 'математика', 'огэ'). Датасет не особо приспособлен для алгоритмов этого типа, по этому было необходимо парсить руками отзыв, исключая бесполезные слова. Вот часть полученных ассоциативных правил:

```
('ужасно', 'преподаватель'),
('ужасно', 'занятия'),
('ужасно', 'математика'),
('ужасно', 'огэ'),
('ужасно', 'репетитор'),
('ужасно', 'заниматься'),
('ужасно', 'занятий'),
('ужасно', 'егэ'),
('ужасно', '5'),
('ужасно', 'только'),
('ужасно', 'просто'),
('преподаватель', 'преподаватель'),
('преподаватель', 'занятия'),
('преподаватель', 'заниматься'),
('преподаватель', 'довольны'),
('преподаватель', 'если'),
('преподаватель', 'ребенок'),
('занятия', 'огэ'),
('занятия', 'просто'),
('математика', 'огэ'),
('огэ', '5'),
```

5.3 REST API

Продемонстрируем работу API.
Локально поднимем сервер, будем посылать запросы через Postman. Данные для алгоритмов были искусственно сгенерированы для тестирования: для Affinity Propagation - нормальное распределение вокруг центров (1, 0) и (10, 0), для PrePost - пример, использованный в теоретическом объяснении алгоритма.

Создали 2 эксперимента с разными алгоритмами, теперь их можно запустить, а затем получить результаты [Приложение 1].

Таким образом наше API готово для работы с удаленной базой данной, остальная работа зависит только от самой базы данных и ее специфик. В ходе работы могут возникуть проблемы с тем, что мы не имеем прямой доступ к данным и не можем извлекать необходимые поля.

6 Источники

Affinity Propagation:

- Clustering by Passing Messages Between Data Points (Science, 16 FEBRUARY 2007, Brendan J. Frey and Delbert Dueck)
- Статья на Хабре
- Статья на Towards Data Science

PrePost:

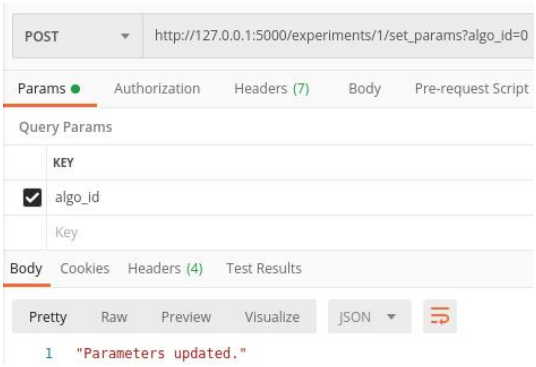
- A New Algorithm for Fast Mining Frequent Itemsets Using N-Lists(Sciece China. Information Sciences · September 2012, DENG ZhiHong, WANG ZhongHui & JIANG JiaJian)

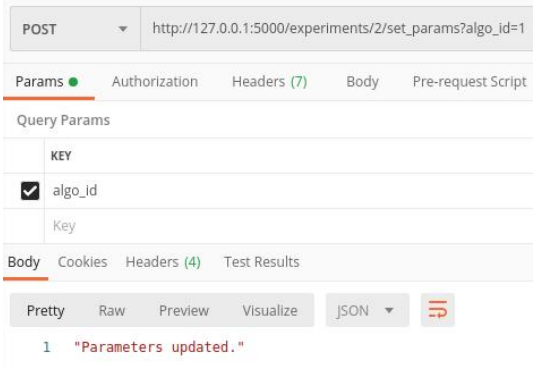
REST API:

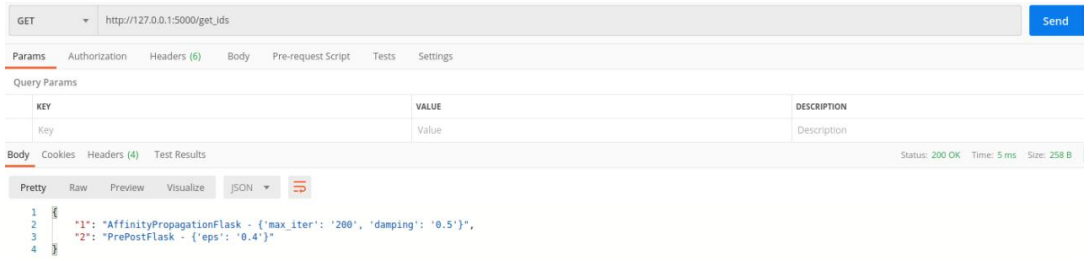
- Flask-restful
- Полезный туториал

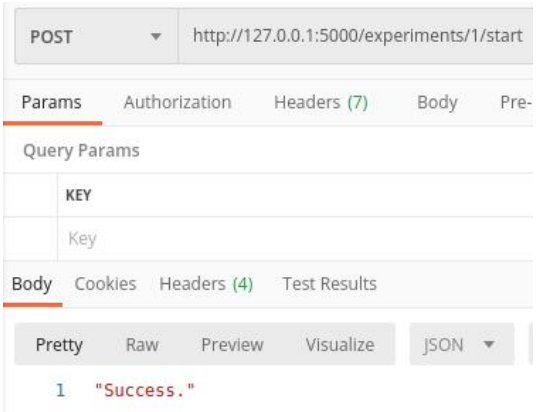
7 Приложение

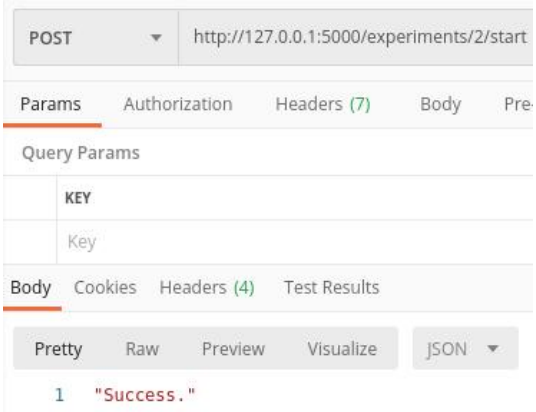
7.1 Результаты тестирования REST API











Полученные результаты: