

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 успешности завершения дисциплины»

(промежуточный, этап 1)

Выполнил:

студент группы БПМИ182



Подпись

Аюпов Ш.И.

И.О. Фамилия

01.02.2020

Дата

Принял:

руководитель проекта

Паринов Андрей Андреевич

Имя, Отчество, Фамилия

м.и.с. МНУЛ ИССА ФКН НИУ ВШЭ

Должность

Место работы

Дата 11.02. 2020

Оценка (по 10-тибалльной шкале)

10



Подпись

Москва 2020

СОДЕРЖАНИЕ

Оглавление

ВВЕДЕНИЕ.....	3
Задачи	4
The PAM Clustering algorithm	5
AprioriDP	6
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ.....	7
ПРИЛОЖЕНИЕ 1.....	8
ПРИЛОЖЕНИЕ 2.....	10

ВВЕДЕНИЕ

Часто студент не может точно и разумно оценить сложность каждой конкретной учебной дисциплины, в связи с чем уделяет недостаточно внимания важным или трудным предметам. Смысл системы состоит в выявлении студентов, находящихся в группе риска невыполнения конкретной дисциплины. Нахождение таких проблемных мест помогло бы как самим студентам, которые могли бы своевременно уделить такому предмету больше внимания, так и Университету, который, зная дисциплины, которые вызывают проблемы, мог бы организовывать адаптационные курсы и/или консультации. Таким образом, актуальность данной проблемы видна невооруженным взглядом.

Универсального решения данной проблемы найдено не было, поэтому было принято решение использовать методы кластеризации данных и методы поиска ассоциативных правил.

Напомним, что задача кластеризации состоит в том, чтобы, имея большое количество объектов и функцию похожести (или непохожести), объединить эти объекты в кластеры, так, чтобы в объекты в пределах одного кластера были максимально схожи, а между кластерами максимально различались. Отметим, что определение довольно размыто, и непонятно какие кластеризации считать хорошими, а какие плохими. Для этого существуют различные метрики [7] При этом функция похожести, вообще говоря, может быть произвольной, и существует огромное количество метрик. [1][2]

Поиск ассоциативных правил заключается в том, что, имея базу данных, элементы которой – это подмножества некоторого множества объектов (здесь объекты понимаются в другом относительно кластеризации смысле), найти в ней правила вида: (подмножество1) → (подмножество2), которое означает, что если в элементе базы данных есть подмножество1, то скорее всего в этом же элементе есть подмножество2. Для оценки правил используются разные величины, такие как доверие и поддержка.

Задачи

Целью поставлена разработка системы предсказания успешности завершения дисциплины. Были выделены следующие задачи:

1. Изучение и сравнение алгоритмов кластеризации;
2. Изучение и сравнение алгоритмов поиска ассоциативных правил;
3. Реализация алгоритма кластеризации на языке Python;
4. Реализация алгоритма поиска ассоциативных правил на языке Python;
5. Создание web-сервиса;
6. Тестирование алгоритмов на данных;

Проект является групповым; задачи распределены так: каждый выбирает один алгоритм кластеризации и алгоритм поиска ассоциативных правил, разбирает их и представляет на общем собрании. Таким образом, каждый участник понимает и представляет, как работает большое количество алгоритмов. Затем выбранные алгоритмы реализуются на языке Python с возможным добавлением библиотек.

Мной были выбраны алгоритмы: The PAM Clustering algorithm и AprioriDP – алгоритм поиска ассоциативных правил. Описание и подробности в работе этих алгоритмах в соответствующих разделах.

Для реализации был выбран язык Python 3.7, а также используются библиотеки NumPy, SciPy, Matplotlib.

При этом встает вопрос об актуальности реализации данных алгоритмов. Может быть они уже где-то реализованы? Ответ: да, но не в библиотеках Python – большинство часто используемых библиотек не содержат реализации, например, SciPy [8].

The PAM Clustering algorithm

Нам даны n объектов, функция "непохожести" $d(i, j)$ и число k . Мы хотим кластеризовать объекты в k кластеров: отнести каждый объект к одному из кластеров так, чтобы объекты в одном кластере были похожи друг на друга и не похожи на объекты из других кластеров. Данный алгоритм работает по следующему принципу: он пытается найти для каждого кластера представителя (medoid) - "центральный" объект в кластере, отнести каждый объект к ближайшему представителю и минимизировать суммарное расстояние от объектов до назначенных представителей.[3] При этом все комбинации проверить слишком сложно ($O(nk)$), поэтому ищет он по-другому, а именно является алгоритмом локального поиска: имея какую-то кластеризацию, пытается улучшить результат, просматривая "соседние" кластеризации и выбирая лучшую, то есть ту, которая больше всего минимизирует целевую функцию (суммарное расстояние).

Для алгоритмов локального поиска нужно определить 2 вещи: начальную конфигурацию и определение соседей. Начальную конфигурацию алгоритм задает жадно (фаза BUILD), а соседи определены так: берем в конфигурации представителя и объект, не являющийся представителем, меняем их роли и переназначаем все объекты (фаза SWAP). Иными словами, две конфигурации соседние, если могут быть получены swap-ом представителя и не-представителя. Дополнительно храним для каждого объекта i числа D_i - расстояние до ближайшего представителя и E_i - расстояние до второго ближайшего представителя. Введем также множества S и U - соответственно, представители и не-представители. Для лучшего понимания см. Приложение 1.

BUILD работает жадно: сначала добавляет объект, который ближе всех к другим, затем добавляет по одному объекту такому, который максимизирует вклад - сумму расстояний до него невыбранных объектов, которые должны быть назначены ему.

SEARCH организован так: мы просматриваем все пары представителей и не-представителей ($(S \times U)$), и считаем разницу, которая произойдет при замене в выбранной паре. Затем пытаемся выбрать такую пару, разница при которой самая маленькая: если разница отрицательна, значит целевую функцию можно уменьшить, применяем эту замену и ищем дальше, иначе целевую функцию уменьшить нельзя и алгоритм завершает работу. Дополнительно можно задать количество максимальных итераций.

Данный алгоритм дает хорошие результаты, но существенной проблемой является время работы (до сходимости SWAP фазы). Существует несколько способов решения этой проблемы: семплирование и оптимизации. Первый означает, что мы берем из всей кучи данных только часть (семплы) и запускаем на них изначальный алгоритм и возвращаем ответ. Данный метод формально уже является другим алгоритмом и называется CLARA/CLARANS и разбираться не будет. Другой состоит в нескольких оптимизациях и позволяет уменьшить время работы, его название FastPAM. [4] Суть его в следующем:

1. Сохранять для каждой точки самого близкого представителя, а также расстояния до него и до второго ближайшего, чтобы не пересчитывать целевую функцию целиком.
2. Для каждого не-представителя считать рассматривать все представители разом, а не по очереди - экономия лишних вычислений одного и того же - ожидается $O(k)$ speedup с тем же результатом.
3. Сохранять для каждого представителя лучший swap и в конце итерации применить все улучшающие - уменьшение количества итераций в сравнении с предыдущей оптимизацией за счет возможного ухудшения результата.
4. Улучшение BUILD фазы - линейное приближение - использование sample выборки.

AprioriDP

Пусть есть множество предметов $I = \{I_1, \dots, I_n\}$ и база данных $T = \{T_1, \dots, T_p\}$, где $T_i \subseteq I$ - транзакция. Определим ассоциативное правило, как $X \Rightarrow Y$, где $X, Y \subseteq I$. Введем также понятия поддержка набора: $supp(X) := |\{t \in T | X \subseteq t\}|/|T|$, то есть доля транзакций, содержащих данный набор; доверие правила: $conf(X \Rightarrow Y) := supp(X \cup Y)/supp(X)$ - отношение количества транзакций, содержащих как X , так и Y , к количеству транзакций, содержащих X . Тогда задача формулируется так: по заданной базе данных и ограничениям на минимальную поддержку и минимальное доверие найти такие правила $X \Rightarrow Y$, что $supp(X \cup Y) > min_supp$ и $conf(X \Rightarrow Y) > min_conf$. Алгоритм Apriori [5] работает в 2 этапа: сначала находит часто встречаемые подмножества (согласно ограничению), затем строит из полученных подмножеств правила (согласно ограничению). Рассмотрим каждый этап.

Построение частых множеств. Алгоритм берет за основу следующую идею: если множество встречается часто, то есть условие выполняется, то любое его подмножество встречается часто, и наоборот, если множество не встречается часто, то и его супермножества не встречаются часто. Поэтому алгоритм работает так: начинаем с 1-элементных множеств, находим частые - пусть они входят в L_1 , затем на их основе формируем кандидаты - 2-х элементные множества (Назовем C_2), которые могут быть частыми, и отсекаем те из них, которые встречаются нечасто, получаем L_2 . И так далее - формируем C_k на основе L_{k-1} и проходим по базе данных получаем L_k . Продолжаем до тех пор, пока $L_k \neq \emptyset$. Остается понять, как формировать C_k на основе L_{k-1} . У элемента из C_k должно выполняться следующее свойство: любое его $k-1$ элементное подмножество содержится в L_{k-1} . Это довольно сильное свойство и его сложно проверять. Вместо этого можно рассматривать все пары элементов из L_{k-1} , и если множества в паре отличаются всего одним элементом, то берем их объединение в C_k . И в первом, и во втором случае все элементы надо проверить, то есть пройти по базе данных, поэтому оставим второй вариант. Наконец, ответом будет объединение L_i .

Построение ассоциативных правил. Имея, какое-то частое множество F , мы хотим построить для него правила. Делаем так: пусть X, Y разбиение F , тогда надо проверить условие: $conf(X \Rightarrow Y) = supp(X \cup Y)/supp(X) > min_conf$. Поскольку F частое множество, то и его подмножества частые, а для частых множеств мы знаем их поддержку. Итак, рассматриваем все непустые подмножества $s \subseteq F$ и считаем $conf(s \Rightarrow (F \setminus s)) = supp(F)/supp(s)$. Теперь заметим, что числитель не зависит от s . Поэтому мы можем рассматривать максимальные значения $supp(s)$, которые достигаются на одноэлементных множествах. Из этого также следует, что если для s значение доверия нам подходит, то и на всех его супермножествах значение доверия нам подойдет, и наоборот, если для s не подходит, то не подойдет и для всех подмножеств. Тогда можно организовать постройку правил рекурсивно.

Наконец, алгоритм AprioriDP [6] является оптимизацией первого этапа алгоритма Apriori. Вместо того, чтобы каждый раз заново проходить по базе данных, можно запомнить количество комбинаций заранее за один проход. В первом приближении, за первый проход, можно посчитать количество встречаемых 1- и 2-х элементных подмножеств и запомнить данные в какую-нибудь структуру. Поэтому построение C_2 и L_2 будет значительно быстрее. Дальнейшие шаги аналогичны простому алгоритму. По аналогии можно запоминать вхождения 3-х, 4-х, . . . k -элементных подмножеств, но следует учитывать, что тогда этот проход будет требовать больше времени на предобработку - $O(pnk)$ - и на запрос - $O(k)$ при наивной реализации.

Для лучшего понимания см. Приложение 2.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Haq, Irfan Ul & Caballero, Juan. (2019). A Survey of Binary Code Similarity.
2. Xu, D., Tian, Y. A Comprehensive Survey of Clustering Algorithms. *Ann. Data. Sci.* **2**, 165–193 (2015). <https://doi.org/10.1007/s40745-015-0040-1>
3. Kaufmann, Leonard and Rousseeuw, Peter. (1987). Clustering by Means of Medoids. *Data Analysis based on the L1-Norm and Related Methods*. 405-416.
4. Schubert, Erich, and Peter J. Rousseeuw. “Faster k-Medoids Clustering: Improving the PAM, CLARA, and CLARANS Algorithms.” *Lecture Notes in Computer Science* (2019): 171–187. Crossref. Web.
5. Rakesh Agrawal and Ramakrishnan Srikant. 1994. Fast Algorithms for Mining Association Rules in Large Databases. In *Proceedings of the 20th International Conference on Very Large Data Bases (VLDB'94)*, Jorge B. Bocca, Matthias Jarke, and Carlo Zaniolo (Eds.). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 487-499.
6. Bhalodiya, Dharmesh and Patel, Kamlesh and Patel, Chhaya. (2013). An Efficient way to Find Frequent Pattern with Dynamic Programming Approach.. 10.1109/NUiCONE.2013.6780102.
7. Spencer Norris, Assessment Metrics for Clustering Algorithms, <https://opendatascience.com/assessment-metrics-clustering-algorithms/> (Дата обращения 01.02.2020)
8. An overview on clustering methods, <https://scikit-learn.org/stable/modules/clustering.html>. (Дата обращения 01.02.2020)

ПРИЛОЖЕНИЕ 1

Псевдокод алгоритма The PAM Clustering

Input: n объектов, $d(i, j)$ - функция расстояния, $k \in \mathbb{N}$
Output: $c = \{c_1, c_2, \dots, c_k\}$ - представители, $C = \{C_1, C_2, \dots, C_k\}$ - кластеры (разбиение $\{1, \dots, n\}$)

```

1 инициализация  $S \leftarrow \emptyset, U \leftarrow \emptyset, c \leftarrow \emptyset, C_i \leftarrow \emptyset$ ;
2  $S, U, D \leftarrow \text{BUILD}(n, d, k)$ ;
3 посчитать  $E$ ;
4  $S, U \leftarrow \text{SEARCH}(n, d, D, E, S, U)$ ;
5  $c \leftarrow S$ ;
6 for  $u \in U \cup S$  do
7    $i \leftarrow \text{argmin}_{j \in S} d(u, j)$ ;
8    $C_i \leftarrow C_i \cup \{u\}$ ;
9 end
10 return  $c, C$ ;
```

Algorithm 1: Скелет алгоритма

Input: n объектов, $d(i, j)$ - функция расстояния, $k \in \mathbb{N}$
Output: S - множество представителей, $U = \{1, \dots, n\} \setminus S$, D - расстояния от объектов до ближайшего представителя.

```

1 инициализация  $S \leftarrow \emptyset, U \leftarrow \{1, \dots, n\}$ ;
2  $s \leftarrow \text{argmin}_i \sum_j d(i, j)$ ;
3  $S \leftarrow S \cup \{s\}$ ;
4  $U \leftarrow U \setminus \{s\}$ ;
5 посчитать  $D$ ;
6 while  $|S| < k$  do
7    $s \leftarrow \text{argmax}_{i \in U} \sum_{j \in U} \max(D_j - d(i, j), 0)$ ;
8    $S \leftarrow S \cup \{s\}$ ;
9    $U \leftarrow U \setminus \{s\}$ ;
10  обновить  $D$ ;
11 end
12 return  $S, U, D$ ;
```

Algorithm 2: BUILD

Input: n объектов, $d(i, j)$ - функция расстояния, D, E - расстояния от объектов до ближайшего и второго ближайшего представителей, S - множество представителей, $U = \{1, \dots, n\} \setminus S$
Output: S - множество представителей, $U = \{1, \dots, n\} \setminus S$

```

1 инициализация gets_better  $\leftarrow$  true;
2 while gets_better do
3     gets_better  $\leftarrow$  false;
4     for  $(s, u) \in S \times U$  do
5         for  $j \in U \cup \{s\}$  do
6             if  $d(j, s) > D_j$  then
7                 if  $d(j, u) \geq D_j$  then
8                      $K_{jsu} \leftarrow 0$ ;
9                 else
10                     $K_{jsu} \leftarrow d(j, u) - D_j$ ;
11                end
12            else
13                if  $d(j, u) < E_j$  then
14                     $K_{jsu} \leftarrow d(j, u) - D_j$ ;
15                else
16                     $K_{jsu} \leftarrow E_j - D_j$ ;
17                end
18            end
19        end
20         $T_{su} \leftarrow \sum_{j \in U \cup \{s\}} K_{jsu}$ ;
21    end
22     $s_{best}, u_{best} = \operatorname{argmin}_{(s, u) \in S \times U} T_{su}$ ;
23    if  $T_{s_{best}u_{best}} < 0$  then
24        gets_better = true;
25         $S \leftarrow (S \cup \{u_{best}\}) \setminus \{s_{best}\}$ ;
26         $U \leftarrow (U \cup \{s_{best}\}) \setminus \{u_{best}\}$ ;
27        обновить  $D, E$ ;
28    end
29 end
30 return  $S, U$ ;

```

Algorithm 3: SEARCH

ПРИЛОЖЕНИЕ 2

Псевдокод алгоритма AprioriDP

Input: T - база данных, min_supp - ограничение на поддержку, min_conf - ограничение на доверие

Output: L - множество частых подмножеств, R - множество правил

```

1  $L \leftarrow ConstructFrequentSets(T, min\_supp);$ 
2  $R \leftarrow ConstructRules(L, min\_conf);$ 
3 return  $L, R;$ 

```

Algorithm 4: AprioriDP

Input: T - база данных, min_supp - ограничение на поддержку

Output: L - множество частых подмножеств

```

1 инициализация  $count\_table[n][n]$  нулями;
2 for  $t \in T$  do
3   for  $i \leftarrow 1$  to  $len(t)$  do
4     for  $j \leftarrow i$  to  $len(t)$  do
5        $count\_table[t_i][t_j] \leftarrow count\_table[t_i][t_j] + 1;$ 
6     end
7   end
8 end
9 for  $i \leftarrow 1$  to  $n$  do
10  if  $count\_table[i][i] \geq min\_supp \cdot DB\_size$  then
11     $L_1 \leftarrow L_1 \cup \{i\};$ 
12    for  $j \leftarrow i + 1$  to  $n$  do
13      if  $count\_table[i][j] \geq min\_supp \cdot DB\_size$  then
14         $L_2 \leftarrow L_2 \cup \{\{i, j\}\};$ 
15      end
16    end
17  end
18 end
19  $k \leftarrow 3;$ 
20 while  $L_{k-1} \neq \emptyset$  do
21    $C_k \leftarrow \{a \cup b \mid a, b \in L_{k-1}, |a \cup b| = k\};$ 
22   for  $t \in T$  do
23      $S \leftarrow \{c \in C_k \mid c \subseteq t\};$ 
24     for  $s \in S$  do
25        $count[s] \leftarrow count[s] + 1;$ 
26     end
27   end
28    $L_k \leftarrow \{c \in C_k \mid count[c] \geq min\_supp \cdot DB\_size\};$ 
29    $k \leftarrow k + 1;$ 
30 end
31 return  $\bigcup_i L_i;$ 

```

Algorithm 5: ConstructFrequentSets

Input: L - множество частых подмножеств, min_conf - ограничение на доверие
Output: R - множество правил

```

1 for  $F \in L$  do
2    $r \leftarrow \emptyset$ ;
3    $CheckSubset(F, F, r, min\_conf)$ ;
4    $R \leftarrow R \cup r$ ;
5 end
6 return  $R$ ;
```

Algorithm 6: ConstructRules

Input: F - общее подмножество, s - текущее подмножество, r - множество правил (изменяемый объект), min_conf - ограничение на доверие

```

1 if  $\frac{supp(F)}{supp(s)} \geq min\_conf$  then
2    $r \leftarrow r \cup \{s \Rightarrow (F \setminus s)\}$ ;
3   for  $\hat{s} \in \{a \subset s \mid |a| = |s| - 1\}$  do
4      $CheckSubset(F, \hat{s}, r, min\_conf)$ ;
5   end
6 end
7 return;
```

Algorithm 7: CheckSubset