Федеральное государственное автономное образовательное
учреждение высшего образования
«Национальный исследовательский университет «Высшая школа экономики»

Факультет компьютерных наук
Основная образовательная программа
Прикладная математика и информатика

КУРСОВАЯ РАБОТА

на тему

«Алгоритмы машинного обучения в задачах выявления предпочтений»

Выполнил студент группы БПМИ141, 2 курса,
Климкин Андрей Михайлович

Научный руководитель:
канд. тех. наук, доцент ФКН
Объедков Сергей Александрович

Москва 2016

Содержание

Аннотация. В данной работе описана основная идея задачи выявления предпочтений, а также представлены некоторые алгоритмы машинного обучения для её разрешимости. Первый из описанных алгоритмов основан на гипотезе компактности, а также на методах метрических классификаторов, а именно на методе ближайших соседей. Построенная модель алгоритма не имеет аналогов в мире. Второй алгоритм уже является достаточно известным и базируется на идее лексикографической структуры предпочтений пользователей. Смоделирована основная задача для данных алгоритмов, описан их принцип работы, а также их временная сложность. Представлены идеи применения различных эвристик для улучшения эффективности приведённых моделей. Также в работе проведено сравнение результатов алгоритмов между собой на экспериментальных данных.

Введение

Общая постановка задачи

Одним из широко развивающихся разделов машинного обучения являются методы обучения предпочтениям [1], основной задачей которых является обучение алгоритма по уже известным данным так, чтобы он смог предсказывать предпочтения объектов уже на новом множестве элементов, предпочтения на котором нам уже не известны. Данный раздел машинного обучения широко используется, например, в рекомендательных системах [2]. На данный момент известно множество эффективных алгоритмов выявления предпочтений[4]: коллаборативная фильтрация, разрешающие деревья, Байесовские методы и многие другие. Существует несколько постановок задачи выявления предпочтений, рассмотри одну из их.

Представим себе, что у нас есть конечное множество пользователей U, а также конечное множество объектов O. Каждый из объектов характеризуется вектором признаков $(x_1, x_2, ..., x_n)$. Например, если считать, что объекты – это автомобили, то их признаками могут быть цвет, стоимость, модель и т.д. Также любой из пользователей может обладать своими характеристики $(y_1, y_2, ..., y_m)$. Предпочтением будем называть тройку элементов (u, o_1, o_2), где $u \in U$, $o_1, o_2 \in O$. Скажем, что в данной тройке объект o_1 является более приоритетным, чем o_2 для u. Что же это означает? Вернёмся к примеру с машинами и представим, что u – это человек, выбирающий машину, а o_1, o_2 – это два предложенных ему автомобиля. Если говорить неформально, то o_1 имеет больший приоритет, чем o_2, если машина o_1 нравится человеку u больше, чем машина o_2. Главная задача алгоритмов выявления предпочтений заключается в том, чтобы по уже известным тройкам (u, o_1, o_2) восстановить все остальные или только нужную в конкретной постановке задачи часть.

Структура работы

В данной работе будут рассмотрены два алгоритма выявления предпочтений.
В П.2 будет рассмотрен первый алгоритм, основанный на гипотезе компактности [П.2] и методе ближайшего соседа [П.2]. Будет подробно описана идея построения модели, способы выбора метрики близости, этап обучение алгоритма, а также его работа на конкретном запросе тестовой выборки. В конце главы будет описана временная асимптотика построения данной модели.
В П.3 будет представлен второй алгоритм, основанный на идее лексикографического вида предпочтений [П. 3.1]. В начале главы изложена основная идея лексикографических моделей предпочтения, затем представлено краткое описание алгоритма обучения модели, а также работы на тестовом запросе. В конце главы указывается временная асимптотика работы данной модели.
Далее в П.4 приведено описание экспериментов для данных алгоритмов, с подробным описанием данных и результатами каждого из методов.
В приложении П.6 подробно описана программная реализация каждого из представленных алгоритмов.

Первый алгоритм

Для решения поставленной задачи была использована идея алгоритма ближайших соседей KNN (k nearest neighbours) с некоторыми модификациями(эвристиками). Большую роль в построении модели сыграла также теория графов.
Метод K ближайших соседей – это метрический алгоритм классификации, основанный на оценивании сходства объектов, использующий гипотезу компактности.
Гипотеза компактности – при удачно введённой мере сходства похожие объекты лежат в одном классе гораздо чаще, нежели в разных.
Основная идея метода ближайших соседей заключается в гипотезе о том, что близкие объекты имеют схожие предпочтения. Чтобы узнать предпочтение конкретного пользователя, про которого у нас мало информации, мы ищем некоторое количество ближайших соседей(пользователей), про которых у нас уже достаточно информации, смотрим на их предпочтения, усредняем их с некоторыми весами и выдаём получившиеся предпочтения как предпочтения исследуемого нами объекта.

Описание идеи алгоритма

Шаг 1 (Построение графа предпочтений)

Для каждого опрошенного человека строится граф его предпочтений. Вершины графа – предлагаемые объекты, например, автомобили. У i-го пользователя будет ребро (a, b), если в обучающей выборке присутствовало предпочтение (u_i, o_a, o_b) .

Предположение. Для i-го человека объект o_a лучше, чем объект o_b, если в графе присутствует путь от вершины a до вершины b, причём обратного пути в графе нет.

Назовём данный метод вычисления **явным вычислением предпочтений на графе**, данный метод также будет участвовать в экспериментах, представленных в П.4, хоть он и не включает никаких методов машинного обучения.

Шаг 2 (Метрики близости)

Теперь нужно рассчитать близость между двумя участниками опроса.

KNN для пользователей, близость по характеристикам

Во-первых, у каждого пользователя есть собственные характеристики, например: возраст, регион проживания, пол, образование, поэтому один из критериев близости двух участников – схожесть их характеристик. Первым шагом неплохо бы было сделать нормировку признаков, чтобы они имели один и тот же масштаб и не зависили от разброса номинальных значений их величин, поэтому для каждого признака вычтем его среднее

значение по всем объектам и поделим на его среднеквадратическое отклонение. Некоторые признаки могут быть более важными, чем остальные, поэтому введём вектор весов $w = (w_1, w_2, ..., w_n)$ для признаков, где n – количество признаков у объекта. Значения данного вектора будут подбираться на обучающей выборке при помощи кросс валидации и метода отжига[3].
Идея **кросс-валидации** заключается в следующем: наша исходная выборка известных предпочтений разбивается на обучающую и контрольную. Затем для каждого разбиения мы настраиваем алгоритм на обучающей выборке, а затем тестируем на контрольной. После этого берём среднее значение по всем разбиениям, тем самым получая некоторую оценку ошибки алгоритма.
Метод отжига[3] – общий алгоритмический метод решения задачи глобальной оптимизации.
Теперь нужно ввести некоторую метрику сходства. Подходящей метрикой является взвешенная метрика Минковского ($p = 2$):

$$\rho(x, x') = \left(\sum_{i=1}^{n} w_i (x_i - x'_i)^p \right)^{\frac{1}{p}}$$

Answer intersection method, близость по предыдущим ответам

Во-вторых, для каждой пары пользователей можно узнать, насколько похожи их предыдущие ответы. Для этого заведём для каждого пользователя множество его ответов – пары вида (o_a, o_b), и для расчёта близости между двумя людьми нужно найти мощность объединения множеств двух пользователей. Пусть мы рассматриваем пару людей (u_1, u_2). Для расчёта их близости нам потребуется две переменные – $same$ и dif. Если мы знаем, что для пользователя u_1 верно, что объект o_a предпочтительнее, чем объект o_b, и данное утверждение также верно и для пользователя u_2, то необходимо увеличить переменную близости $same$ на некоторую величину. Если же для пользователя u_2 справедливо, что объект o_b, наоборот, лучше, чем o_a, то увеличивается dif на некоторую величину. Тогда **антиблизостью** двух пользователей по их предпочтениям назовём величину равную $(dif - same)$. Чем данная величина меньше, тем предыдущие ответы(предпочтения) двух пользователей более схожи.

После того как мы знаем расстояние между двумя пользователями по характеристикам и расстояние, полученное при помощи метода близости по предыдущим ответам, для нахождения точной величины расстояния между двумя пользователями можно сложить два расстояния с некоторыми коэффициентами, данные коэффициента также подбираются методом отжига[3]. Если данная величина хотя бы для одной пары пользователей отрицательна, то нужно найти минимум расстояния между всеми парами пользователей и увеличить этот минимум до 0, также отнимая этот же минимум от всех остальных пар расстояния между всеми пользователями. Теперь все расстояния – это неотрицательные

числа.

Стоит также отметить, что величины метрики по предыдущим ответам можно рассчитывать лишь один раз после получения входных данных, в то время как расстояние по характеристикам зависит от вектора w и должно пересчитываться каждый раз после изменения данного вектора весов.

KNN для объектов, близость по характеристикам

Помимо явной близости пользователей, можно также рассчитать близость между объектами, которые пользователи выбирают, по их собственным характеристика. В примере с автомобилями данными характеристиками могут являться: тип автомобиля, трансмиссия, размер двигателя, потребление топлива. При рассмотрении очередного тестового запроса на выявление предпочтения можно брать данную информацию про близость объектов также во внимание. Метрика близости для данного метода аналогична метрике близости между пользователей по их характеристикам – метрика Минковского($p = 2$). Для данной метрики также остаётся и идея с вектором весов w для признаков объектов.

Шаг 3 (Работа классификатора на конкретном запросе)

Теперь о работе самого алгоритма на конкретном запросе. Пусть у нас есть запрос на выявление предпочтения $(user, o_a, o_b)$, то есть нам нужно определить для человека $user$, какой объект он, скорее всего, выберет – o_a или o_b. Для $user$ мы находим K_{user} ближайших соседей по метрике близости между пользователями [П. 2.1.2]. Пусть расстояние до K_{user} ближайших пользователей от $user$, по предложенной метрике, – это вектор (d_1, d_2 ... $d_{K_{user}}$). Заведём два значения $voteforfirst$ и $voteforsecond$, где $voteforfirst$ величина для накопления приоритета объекта o_1 над o_2, а $voteforsecond$ для накопления приоритета объекта o_2 над o_1 соответственно. Рассматривая очередного для $user$ ближайшего соседа (пусть его номер = x), мы можем узнать на основе обучающей выборки, правда ли, что для x верно, что o_a лучше, чем o_b (напомню, что лучше, по предположению, оговоренному выше, равносильно существованию пути от вершины a до вершины b и не существованию пути от b до a). Если ответ положительный, то нужно к приоритету первого над вторым ($voteforfirst$) прибавить величину равную Normal(d_x), где d_x расстояние между $user$ и x, Normal(d_x) — плотность нормального распределения с нулевым средним и некоторой дисперсией в точке d_x. Если же ответ отрицательный, то мы меняем в запросе местами o_a и o_b и проделываем точно такие же действия, но теперь, при положительном ответе, прибавляем величину Normal(d_x) уже к $voteforsecond$.
После этого используется идея KNN для самих объектов. Пусть в запросе на выявление предпочтения присутствуют два объекта o_a и o_b. Найдём у каждого из данных объектов K_o ближайших по характеристикам объектов. Пусть для объекта o_a – это $(o_{a_1}, o_{a_2}, \ldots, o_{a_{K_c}})$, для объекта o_b – это $(o_{b_1}, o_{b_2} \ldots o_{b_{K_c}})$. Построим всевозможные пары

вида (o_a, o_{b_j}) и (o_{a_j}, o_b). Затем попытаемся для данного пользователя узнать ответ на запрос предпочтения $(user, o_a, o_{b_j})$ и $(user, o_{a_j} o_b)$, но, в отличие от предыдущего метода для выявления предпочтения на основе ближайших пользователей, данный метод проверяет предпочтения уже на графе самого пользователя $user$, о предпочтениях которого у нас и спрашивают. В зависимости от ответов на данные запросы нужно увеличивать либо $votefor first$(в случае положительных ответов на запросы, в которых присутствует o_a) либо $voteforsecond$(в случае положительных ответов на запросы, в которых присутствует o_b) на Normal(d_j), где d_j – расстояние между o_a и o_{b_j}, или же между o_{a_j} и o_b.

После всех проделанных расчётов дать ответ на вопрос о том, какой же объект - o_1 или o_2 имеет больший приоритет для u становится совсем просто – нужно просто сравнить два накопленных приоритета $votefor first$ и $voteforsecond$. Если в результате получилось, что $votefor first > voteforsecond$, то o_1 имеет больший приоритет для u нежели o_2, иначе o_2 более приоритетный для u, чем o_1.
Осталось только использовать метод кросс-валидации для того, чтобы найти оптимальные K_{user}, K_o, наилучшее значения вектора весов w_u, w_o и оптимальные коэффициенты вклада двух методов ближайших соседей – KNN для пользователей и KNN для объектов(здесь и ниже оптимальными назовём те параметры, при которых алгоритм ошибается наименьшее количество раз на кросс-валидации). K_{user}, K_o ищутся перебором, а вектора весов и вклады двух методов ближайших соседей ищутся при помощи метода отжига [3].

Асимптотика работы алгоритма

Оценим два основных слагаемых, входящих в асимптотику данного алгоритма. Самым затратным временным этапом в алгоритме является этап подбора оптимальных параметров. Рассмотрим его сложность. Пусть мы сейчас находимся в конкретной итераций метода отжига, тогда:

- Первое времязатратное место – это пересчёт расстояний между всеми пользователями, а также между объектами по их характеристикам, ведь на новой итерации отжига веса атрибутов поменялись, следовательно поменялись и все расстояния, построенные по характеристикам.
  Сложность данного шага:

$$O(max(users^2 \cdot \log_2 users, items^2 \cdot \log_2 items)) \tag{1}$$

  где $users$ и $items$ количество пользователей и объектов в выборке.

- Второе времязатратное место – это поиск оптимальных K_{user} и K_o для фиксированных весов. Так как K_{user} и K_o ищутся независимо друг от друга, то рассмотрим асимптотику для поиска K_{user}, поиск величины K_o будет иметь аналогичную

асимптотику. Первым делом алгоритм перебирает всевозможные значения параметра K_{user}. При каждом фиксированном значении K_{user} нужно узнать ответ для каждого предпочтения из контрольной выборки кросс-валидации(пусть её размер = checksize). Каждое предпочтение из контрольной выборки кросс-валидации требует линейного от K_{user} объёма времени. М

Каждый K_{user} оценивается сверху общим количеством пользователей = $users$. Каждый K_o оценивается сверху общим количеством объектов = $items$. Значит имеем такую асимптотику на данный этап алгоритма:

$$O(max(users^2 \cdot checksize, items^2 \cdot checksize) \tag{2}$$

где $checksize$ – размер контрольной выборки кросс-валидации

Обозначив сложность метода отжига за $anneal$, отметив, что мы метод отжига запускается несколько раз из различных начальных состояний(пусть это количество = $ANNEAL\_STARTS$ раз), получаем итоговую асимптотику:

$$O(ANNEAL\_STARTS \cdot anneal \cdot (max([1], [2])) \tag{3}$$

Если $checksize$ не слишком мал, то 1 перевешивает 2. Получаем итоговую сложность:

$$O(ANNEAL\_STARTS \cdot anneal \cdot max(users, items)^2 \cdot checksize) \tag{4}$$

Второй алгоритм

Основная идея лексикографических моделей предпочтений

Как альтернативу первому методу выявления предпочтений рассмотрим алгоритм, построенный на идее о том, что предпочтения каждого пользователя имеют лексикографический порядок. Предположим, что каждый рассматриваемый объект опять однозначно задаётся вектором собственных признаков.

Лексикографический порядок предпочтения – отношение предпочтения, задающееся последовательностью признаков, где приоритетный объект выбирается путём последовательного сравнения признаков в некотором порядке по некоторому правилу. Основной задачей обучения лексикографических алгоритмов является нахождения оптимального, ошибающегося реже всего на тестовых предпочтениях, порядка признаков для последовательного сравнения, а также способа сравнения номиналов конкретного признака: убывающий приоритет, например, цена для автомобилей, чем цена больше, тем приоритет меньше, возрастающий приоритет, например, качество автомобиля, чем номинал данного атрибута выше, тем объект лучше, или же какие-то другие способы ранжирования.

Формальное определение лексикографического предпочтения. Пусть для пользователя u нам нужно выяснить, какой из предложенных объектов – o_1 или o_2 он выберет. Объект o_1 задаётся признаками $(x_1, x_2, ..., x_n)$, o_2 признаками $(y_1, y_2, ..., y_n)$. Для фиксированной последовательности сравнения признаков $i_1, i_2, ..., i_n$ объект o_1 будет предпочтительнее чем o_2, если выполнено следующее условие:

$$(x_{i_1} > y_{i_1}) \vee (x_{i_1} = y_{i_1}, x_{i_2} > y_{i_2}) \vee ... \vee (x_{i_1} = y_{i_1}, ..., x_{i_{n-1}} = y_{i_{n-1}}, x_{i_n} > y_{i_n})$$

Также стоит отметить, что для разных пользователей оптимальные последовательности сравнения признаков, вообще говоря, не обязаны совпадать. Как же для каждого пользователя найти оптимальную последовательность $i_1, i_2, ..., i_n$? Существует множество алгоритмов, построенных на идее лексикографических предпочтений, для решения данной задачи. В данной работе рассмотрим один из них.

Метод голосования переменных.

Рассмотрим идею построения лексикографического порядка признаков путём метода голосования переменных(variable voting)[5][6]. Пока будем считать, что номиналы признаков – это либо 0 либо 1, то есть каждый объект задаётся последовательностью 0 и 1, причём будем считать, что номинальное значение признака равное 1 всегда лучше, чем 0. В П. 3.3 рассмотрим идею обхода данного ограничения.

Перейдём к идее обучения алгоритма. Пусть U – множество пользователей, O – множество объектов, L – предпочтения обучающей выборки в виде тройки (u, o_1, o_2). Обозначим X как множество признаков элементов множества O, где мощность X равна n, а n –

количество признаков каждого объекта. Для каждого $u \in U$ будем независимо строить лексикографический порядок признаков. Зафиксируем пользователя $u \in U$. Обозначим R_i – ранг i-го признака.

- Начальное заполнение рангов: $\forall i \leq n : R_i = 1$.
- Рассматривая очередное предпочтение $l \in L = (u, o_1, o_2)$, построим множество признаков $D = \{x_i \in X\} : o_1(x_i) \neq o_2(x_i)$, то есть множество всех таких признаков, что их значение на паре рассматриваемых объектов различны.
- Обозначим за D^* множество признаков $\{x^* \in D : \forall x \in X : R(x^*) \leq R(x)\}$.
- Обозначим за D_1 множество $x \in D^*$: $o_1(x_i) \geq o_2(x_i)$, множество всех таких признаков, на которых первый объект не уступает второму.
- Обозначим за D_2 множество $x \in D^*$: $o_2(x_i) \geq o_1(x_i)$, множество всех таких признаков, на которых второй объект не уступает первому.
- Если оказалось, что $|D_1| \leq |D_2|$, то текущие ранги выбранных признаков не удовлетворяют предпочтению, а именно тому факту, что o_1 лучше, чем o_2, значит признакам, на которых второй объект оказался лучше(приоритетнее) чем o_1, а именно элементам из D_2, нужно увеличить ранг на единицу.
- Пересчёт рангов для конкретного пользователя завершается либо в тот момент, когда после очередного просмотра всей обучающей выборки никаких численных изменений в рангах не произошло, либо кода все ранги достаточно велики, например, выполнено условие: $\forall i : R(x_i) \geq n$.

В итоге для пользователя u получаем последовательность признаков:

$$\left\{x_{p_1}, x_{p_2}, ..., x_{p_n} : \forall i < j : R(x_{p_i}) \leq R(x_{p_j})\right\}.$$

Обобщение вида признаков. Небольшое изменение метода голосования.

До этого мы рассматривали случай обучения с признаками, имеющими значения только 0 или 1. Теперь представим, что у нас также есть монотонные признаки, имеющий характер предпочтения либо строго убывающий, либо строго возрастающий, или же категориальные признаки. Первое, что мы сделаем – это для каждого признака, имеющего монотонный характер, добавим его абсолютное значение и абсолютное значение со знаком минус. Затем для каждого категориального признака добавим к общим признакам данного объекта производный вектор рассматриваемого признака размера множества

значений данного признака, в котором у объекта o на i-ом месте будет стоять 1, если значение данного признака у объекта o равно i. Также добавим вектор отрицаний данного признака, в котором у объекта o на i-ом месте будет стоять 1, если значение данного признака у объекта o не равно i. Объединим все полученные признаки в один итоговый вектор атрибутов для объекта

Теперь посмотрим как данное изменение может повлиять на получившиеся последовательности предпочтений. Единственное изменение, которое необходимо внести в метод голосования – это после полного расчёта рангов признаков для пользователя u в итоговой последовательности опустить некоторые признаки: в получившимся лексикографическом порядке признаков мы оставляем лишь те части, где значения одного признака идут подряд: для категориальных признаков – это всевозможные номинальные значения с их отрицаниями, а для монотонных – это абсолютное значение признака со знаком плюс и минус. Для лучшего понимания приведём пример:

$\langle$синий, зелёный, качественный, не зелёный, дорогой, не синий, не качественный, дешёвый$\rangle$ $\Rightarrow$ $\langle$синий, зелёный, качественный, дорогой$\rangle$.

Выбор предпочтения для тестовых запросов

Рассмотрим запрос вида: $<u, o_1, o_2>$, где o_1 и o_2 два различных объекта. То есть нам нужно ответить, какой объект o_1 или o_2 является приоритетным для u. Пусть на обучении был получен оптимальный лексикографический порядок атрибутов для классификации предпочтения пользователя u: $(x_1, x_2, \ldots, x_n)$, где x_i – очередной атрибут для сравнения. Найдём наименьший значение i такое, что $o_1(x_i) \neq o_2(x_i)$. Теперь, если $o_1(x_i) > o_2(x_i)$, то алгоритм ответит, что u предпочитает объект o_1 объекту o_2, иначе u предпочитает объект o_2 объекту o_1.

Асимптотика работы алгоритма

Рассмотрим временную сложность полученного алгоритма, а именно этапа обучения. Каждый этап пересчёта рангов для каждого предпочтения из тестовой выборки требует линейного времени от числа признаков. Каждый из признаков меняет ранг не более, чем общее количество признаков раз. Значит расчёт рангов для одного пользователя занимает квадратичное время от количества признаков, умноженное на мощность обучающей выборки. Сортировка занимает асимптотически меньшее время, поэтому не будем её учитывать. Обозначив за n количество признаков каждого объекта, за m общее количество пользователей, за l размер обучающей выборки, получаем итоговую асимптотику:

$$O((n^2 \cdot l) \cdot m) \tag{5}$$

Эксперименты

В заключении работы исследуем эффективность описанных алгоритмах на экспериментальных данных, а также попробуем объяснить полученные результаты.

Описание экспериментов

Первый эксперимент (данные об автомобилях)

Для первого эксперимента были выбраны данные о предпочтениях пользователей между 10-ю машинами. Общее количество пользователей = 60. Пользователям предлагалось сделать выбор между двумя автомобилями (для каждой пары автомобилей). Алгоритм должен был предсказывать выбор пользователя между двумя предложенными автомобилями.

Более подробное описание характеристик автомобилей и пользователей

Признаки каждого автомобиля:

- Тип: Седан(1), Спортивный(2)
- Трансмиссия: Ручное управление (1), Автомат (2)
- Размер двигателя: 2.5л, 3.5л, 4.5л, 5.5л, 6.2л
- Потребление топлива: Гибрид (1), Не-гибрид (2)

Признаки каждого пользователя:

- Образование: Не указано (0), Средняя школа (1), Бакалавриат (2), Докторская степень (3)
- Возраст: Не указано (0), Меньше 25 (1), Между 25-30 (2), Между 30-35 (3), Больше 40 (4)
- Пол: Не указан (0), Мужчина (1), Женщина (2)
- Регион: Не указан (0), Юг (1), Восток (2), Северо-Восток (3), Запад (4)

Описание ответов пользователей
Каждый пользователь сравнивал все пары машин. Ответы пользователей представляют собой пару индексов ($<car_1>$, $<car_2>$), где $<car_1>$, в данной паре, для данного пользователя, лучше, чем $<car_2>$.

Второй эксперимент (данные о суши)

Во втором эксперименте каждому пользователю предлагалось 10 различных видов суши. Общее количество пользователей = 5000. Каждый из них расставил в порядке убывания, начиная от самого любимого и заканчивая менее любимым, предложенные суши. Алгоритм предсказывал по предложенной паре видов суши и номере пользователя какой из видов суши более предпочтителен для данного пользователя.

Более подробное описание признаков каждого пользователя и вида суши

Признаки каждого вида суши

- Идентификатор (от 0 до 9)
- Название
- Разновидность: маки(0), иначе(1)
- Тип: морепродукты(0), иначе(1)
- Основной компонент
  0:Синекожая рыба
  1:Красная рыба
  2:Белая рыба
  3:Угорь
  4:Моллюск
  5:Кальмар или осьминог
  6:Креветки или крабы
  7:Икра
  8:Другие морепродукты
  9:Яйца
  10:Мясо
  11:Овощи
- Жирность (от 0 до 4, 0: самый жирный)
- Как часто пользователь ест суши, от 0 до 3, 3: очень часто
- Цена
- Как часто суши продаются в магазине, от 0 до 1, 1: чаще всего

Признаки каждого пользователя

- Идентификатор пользователя
- Пол
  0:Мужчина
  1:Женщина
- возраст
  0:15-19
  1:20-29
  2:30-39
  3:40-49
  4:50-59
  5:Больше 60
- Сколько времени потребовалось для заполнения формы
- Идентификатор префектуры, в которым проживал до 15 лет
- Идентификатор региона, в которым проживал до 15 лет
- Восточный или Западный регион проживания до 15 лет
- Идентификатор префектуры, в котором проживает на данный момент
- Идентификатор региона, в котором проживает на данный момент
- Восточный или Западный регион проживания на данный момент
- 0, если пункты 5-8 одинаковы, иначе 1

Описание ответов пользователей
Для каждого пользователя порядок из 10 чисел – виды суши в порядке убывания приоритетности.

Разделение данных для экспериментов

Исходные данные (предпочтения) были поделены на две части: обучающую выборку и тестовую выборку. Размер тестовой выборки 25% от всех данных. Размер обучающей выборки 75% от всех данных, включая данные для кроссс-валидации.

Полученные результаты

Ниже представлены численные результаты трёх алгоритмов для двух экспериментов: средний процент верных ответов алгоритма, а также наихудший результат каждого из методов классификации среди всех запусков алгоритма. Третий алгоритм является простым явным вычислением транзитивных предпочтений на графе, давая случайный ответ на запрос предпочтения в спорных ситуациях. Для более детальных результатов во втором эксперименте было введено дополнительное разбиение данных: в некоторых частях эксперимента рассматривались предпочтения не всех пользователи, а только части из них. Так были выбраны предпочтения первых ста, двухсот, пятисот, а также первой тысячи пользователей. Для удобства обозначим метрический алгоритм, описанный в П. 2 как "Main solution".

Эксперимент 1. Данные об автомобилях.

название метода	средний процент правильных ответов алгоритма	худший результат
Main solution	82.16	79.82
Явное вычисление на графе	73.83	71.35
LPM(vote method)	69.82	63.46

Эксперимент 2. Данные о суши.

Main solution [2]

количество пользователей	средний процент правильных ответов алгоритма	худший результат
100	80.00	74.05
200	79.47	75.02
500	79.79	74.57
1000	81.01	77.75

Явное вычисление на графе [2.1.1]

количество пользователей	средний процент правильных ответов алгоритма	худший результат
100	74.15	71.32
200	74.60	72.47
500	74.63	73.60
1000	74.39	73.35

LPM Ranking method [3]

количество пользователей	средний процент правильных ответов алгоритма	худший результат
100	67.15	62.69
200	67.59	65.74
500	67.76	66.20
1000	67.44	66.31

Выводы. Объяснение получившихся результатов экспериментов.

Из результатов, приведённых в П. 4.2, наглядно видно, что метод голосования в обоих экспериментах достаточно сильно уступает метрическому алгоритму, а также явному вычислению предпочтений на графе. Для эксперимента с автомобилями это можно объяснить тем фактом, что метод голосования, представленный в П. 3 болезненно реагирует на любые противоречивые предпочтения в обучающей выборке, а в первом эксперименте таких предпочтений было не малое количество. Что же касается эксперимента с данными о суши, не содержащих шума, здесь можно выдвинуть гипотезу о том, что предложение о предпочтениях, выдвинутое в ходе описания метрического алгоритма П. 2.1.1 оказалось более подходящим для приведённых экспериментов, нежели идея о том, что предпочтения пользователей имеют лексикографический порядок 3.1. Данная гипотеза тесно связана с видом предпочтений пользователей в данном эксперименте – они имеют линейный вид. С другой стороны, лексикографическая модель хоть и даёт худшие результаты, но зато на данных экспериментах процесс обучения данной модели менее времязатратный, нежели процесс обучения метрического алгоритма, а также в отличие

от алгоритма явного вычисления предпочтений на графе, ответ модели метода голосования всегда является детерминированным, то есть не возникает спорных ситуаций со случайным результатом в итоге. Поэтому какую из приведённых моделей выявления предпочтений использовать зависит только от конкретной постановки задачи, а также целей, которые нужно достигнуть при внедрении алгоритма.

Список литературы

[1] Johannes Fürnkranz and Eyke Hüllermeier. Preference Learning: An Introduction. In Preference Learning (Johannes Fürnkranz and Eyke Hüllermeier, editors), pages 1-17, Springer-Verlag, 2010

[2] Francesco Ricci and Lior Rokach and Bracha Shapira, Introduction to Recommender Systems Handbook, Recommender Systems Handbook, Springer, 2011

[3] Kirkpatrick, S.; Gelatt Jr, C. D.; Vecchi, M. P. (1983). "Optimization by Simulated Annealing". Science 220 (4598): 671–680

[4] Segaran T. Programming Collective Intelligence, O'Reilly Media; 1st edition (August 26, 2007).

[5] Fusun Yaman, Thomas J. Walsh, Michael L. Littman, Marie desJardins: Democratic approximation of lexicographic preference models. Artif. Intell. 175(7-8): 1290-1307 (2011)

[6] Fusun Yaman, Thomas J Walsh, Michael Littman, and Marie desJardins. Democratic approximation of lexicographic preference models, 2009. Submitted to Artificial Intelligence.

Приложение

Описание и функциональное назначение методов и полей классов

Math

Краткое описание

Некоторый математический функционал, использующийся по ходу работы алгоритма.

Статические поля

$SIGMA$ – среднеквадратическое отклонение (нужно для нормального распределения Gaus(d))

Статические методы:

$pow2(x)$ – возвращает x^2

$Gaus(d)$ – возвращает значение нормального распределения в точке d

$rand01$ – возвращает значение случайной величины из равномерного($[0; 1]$) распределения

Quest

Краткое описание

Предпочтения пользователей, участвующих в опросе.

Публичные поля

id – идентификатор пользователя

$first$ – выбранная(ый) машина(вид суши) из двух предложенных пользователю в данном вопросе

$second$ – не выбранная(ый) машина(вид суши) из двух предложенных пользователю в данном вопросе

Item

Краткое описание

Характеристики автомобиля (суш).

Публичные поля

id – идентификатор автомобиля(суш)

$attr$ – вектор атрибутов(признаков)

Публичные методы:

$Item(init\_attr)$ – конструктор

User

Краткое описание

Характеристики пользователя.

Публичные поля

id – идентификатор пользователя

$attr$ – вектор атрибутов(признаков)

Публичные методы:

$Item(init\_attr)$ – конструктор

User_Graph

Краткое описание

Граф предпочтений для конкретного пользователя

Публичные поля

g_matrix – граф предпочтений в виде матрицы смежности $g[i][j] = 1$, если для данного пользователя i-ый автомобиль(вид суш) предпочтительнее, чем автомобиль j, $g[i][j] = 0$ иначе.

g – граф предпочтений в виде списка смежности

is_better – двумерный вектор, $is\_better[i][j] = 1$, если существует путь по уже известным предпочтениям в графе g от $Item_i$ до $Item_j$, 0 иначе

Статические методы:

$make\_user\_graph(learn)$ – строит g и g_matrix для каждого пользователя по обучающей выборке вопросов($learn$) и возвращает вектор классов $User\_Graph$.

$calc\_usersGr\_relations(vector < User\_Graph >)$ – по вектору, возращённому из $make\_user\_graph$, заполняет is_better для каждого пользователя.

Публичные методы:

$add\_edge(a, b)$ – вспомогательный метод для $make\_user\_graph(learn)$, добавляет ориентированное ребро (a, b) в g и g_matrix для конкретного пользователя.

Приватные методы:

$IsBetter(a, b)$ – вспомогательный метод для $calc\_usersGr\_relations$. При помощи поиска в глубину(dfs), проверяет существование пути от вершины a до вершины b конкретного пользователя.

$dfs(v, used, need)$ – вспомогательный метод для $IsBetter(a, b)$ – поиск в глубину, проверяет существование пути от вершины v до вершины $need$.

$get\_answers\_dist(User\_Graph,\ second)$ – возвращает расстояние между предпочтениями данного пользователя(у кого был вызван метод) и пользователя $second$ по метрике метода *Answer intersection Method*, описанного в П. 2.3.2.

Knn

Краткое описание

Родительский класс для Users_Knn и Items_Knn (описание данных классов приведено ниже)

Защищённые поля

$weight$ – текущий вектор весов атрибутов признаков User или Item

k – текущий параметр метода ближайших соседей

$answ\_kof$ – текущий вклад метода k-ближайших соседей для пользователей или же метода k-ближайших соседей для машин(суш) в общий классификатор

Публичные методы:

get_k – возвращает значение защищённого параметра k

$change\_k(new\_k)$ – изменяет значение k на new_k

$change\_weights(new\_w)$ – изменяет значение $weight$ на new_w

$show\_w$ – вспомогательные метод для отладки, выводит значение весов атрибутов с точностью до 2-х знаков

Users_Knn

Краткое описание

Вспомогательный класс для методы k-ближайших соседей для пользователей.

Публичные методы:

$Users\_Knn(number\_of\_users\_features)$ – конструктор, создаёт вектор весов размера $number\_of\_users\_features$ (константа = размерности признаков для пользователей)

$get\_users\_dist(User\ first, User\ second)$ – возвращает расстояние между пользователями $first$ и $second$ по метрике, описанной в разделе П. 2.3.1

Items_Knn

Краткое описание

Вспомогательный класс для методы k-ближайших соседей для автомобилей(суш)

Публичные методы:

$Items\_Knn(number\_of\_items\_features)$ – конструктор, создаёт вектор весов размера $number\_of\_users\_features$ (константа = размерности признаков автомобилей(суш))

$get\_items\_dist(Item\ first, Item\ second)$ – возвращает расстояние между автомобилями(видами суш) $first$ и $second$ по метрике, описанной в разделе П. 2.3.3

Distance

Краткое описание

Вспомогательный класс для расчёта расстояний между характеристиками пользователей, между характеристиками автомобилей(суш), между предыдущими ответами.

Статические методы:

$count\_items\_dist(items,\ Items\_par,\ items\_dist)$ – метод заполняет двумерный вектор пар $items\_dist$ на основе поступивших параметров: $items$ – вектор автомобилей(суш), $Items\_par$ – текущее параметры $Items\_Knn$(нужно только поле с весами атрибутов). В $items\_dist[i]$ хранятся пары вида (r_j, j), где j – номер автомобиля(суш), а r_j – расстояние от $item_i$ до $item_j$ по характеристикам. После расчётов $\forall i$ сортирует $items\_dist[i]$ по возрастанию поля r_j.

$count\_users\_answers\_dist(Gr, answ\_dist)$ – метод заполняет двумерный вектор $answ\_dist$, вызывая для каждой пары пользователей $get\_answers\_dist$, тем самым рассчитывает расстояние между каждой парой пользователей по метрике *Answer intersection method*.

$count\_users\_dist(users,\ Users\_par,\ answ\_dist,\ users\_dist)$ – заполняет двумерный вектор пар $users\_dist$ на основе поступивших параметров: $users$ – вектор пользователей, $Users\_par$ – текущее параметры $Users\_Knn$, а также $answ\_dist$ – вектор расстояний, полученный после вызова метода $count\_users\_answers\_dist$. В $users\_dist[i]$ хранятся пары вида (r_j, j), где j – номер пользователя, а r_j – итоговое расстояние от $user_i$ до $user_j$. Первым шагом рассчитывается расстояние между каждой парой пользователей на основе характеристик пользователей данной пары, затем к полученным расстояниям между $user_i$ и $user_j$ прибавляется величина $answ\_dist[i][j]$, так получается итоговое расстояние между $user_i$ и $user_j$. После расчётов $\forall i$ сортирует $users\_dist[i]$ по возрастанию поля r_j.

ReaderCarsData

Краткое описание

Считывает данные для задачи с автомобилями.

Публичные методы:

$get\_users(file, users)$ – считывает данные о пользователях с $file$. На выходе $users$ – вектор с данными о пользователях

$get\_cars(file, items)$ – считывает данные об автомобилях с $file$. На выходе $items$ – вектор с данными об автомобилях

ReaderSushiData

Краткое описание

Считывает данные для задачи с сушами

Публичные методы:

$get\_users(file, users)$ – считывает данные о пользователях с $file$. На выходе $users$ – вектор с данными о пользователях

$get\_sushi(file, items)$ – считывает данные о суши с $file$. На выходе $items$ – вектор с данными об автомобилях

PreparerData

Краткое описание

Родительский класс для $PreparerDataSushi$ и $PreparerDataCara$. Нормализует данные.

Статические методы:

$get\_mean(v)$ – возвращает выборочное среднее значение вектора v

$get\_var(v, mean)$ – возвращает выборочное среднеквадратичное отклонение значений вектора v

$normalize\_weights(X)$ – нормализует атрибуты вектора X

.

PreparerDataSushi

Краткое описание

Наследник класса $PreparerData$. Нормализация параметров, разделение выборки на $learn$, $check$, $quest$.

Публичные методы:

$divide\_pref\_data(file, learn, check, test)$ – делит все предпочтения на три выборки: $learn$, $check$, $test$.

PreparerDataCars

Краткое описание

Наследник класса PreparerData. Нормализация параметров, разделение выборки на *learn*, *check*, *quest*, поиск явно противоречивых ответов(циклов длинны 2)

Публичные методы:

$divide\_pref\_data(file, learn, check, test)$ – делит все предпочтения на три выборки: *learn*, *check*, *test*.

$clean\_quest(learn)$ – удаляет противоречивый предпочтения из *learn* выборки(циклы длинны 2).

KnnSolver

Краткое описание

Классификатор, решающий данную задачу методом k ближайших соседей.

Приватные поля:

MAX_K_USER – верхняя граница *knn* для пользователей

MIN_K_USER – нижняя граница *knn* для пользователей

MAX_K_ITEM – верхняя граница *knn* для объектов

MIN_K_ITEM – нижняя граница *knn* для объектов

$ANNEAL\_START$ – количество запусков метода отжига из случайных начальных точек

$NEED\_TEST\_QUEST$ – размер выборки для одного разбиения кросс валидации

Публичные методы:

$find\_best\_parametrs(users, items, check\_quest, Users\_par, Items\_par, users\_dist, users\_dist, answ\_dist, Gr)$ – на *check_quest* находит наилучшие параметры для классификатора: веса атрибутов, количество ближайших соседей для пользователей и объектов, запоминая их в $Users\_par$ и $Items\_par$.

$choose\_answ(Users\_par, Items\_par, users\_dist, items\_dist, Gr, user\_id, first, second)$ – для пользователя *user_id* возвращает более предпочтительный, по мнению классификатора, индекс объекта из пары ($first$, *second*).

$test\_algo(test\_quest, Users\_par, Items\_par, users\_dist, users\_dist, Gr)$

– возвращает количество неправильно классифицированных предпочтений на множестве $test\_quest$.

Приватные методы:

$check\_par(users, items, check\_quest, Users\_par, Items\_par, users\_dist, users\_dist, answ\_dist, Gr)$ – возвращает количество неправильно классифицированных предпочтений на множестве $check\_quest$ при текущих параметрах классификатора

$random\_fill(v, left, right)$ – заполняет вектор весов v случайными числами из отрезка $[left; right]$

$anneal(w\_user, w\_item, users\_kof, items\_kof, T)$ – имитация одного шага метода отжига. Сдвигает на случайную величину любой из весов векторов w_user и w_item или же любой из коэффициентов вклада двух классификаторов $users\_kof$ и $items\_kof$. Возвращает $true$, если T меньше, чем некоторая температура замерзания(температура остановки метода отжига), $false$ иначе

Дополнительные методы и глобальные параметры

Краткое описание

Так как алгоритм должен корректно работать на двух различных видах данных, в зависимости от данных, нужно менять некоторые параметры алгоритма, этим и занимаются данные методы.

Глобальные параметры

$number\_of\_users$ – количество участников опроса: для данных об автомобилях – это 60, для данных о сушах можно варьировать значение до 5000, для настройки алгоритма удобно ставить маленькое значение, чтобы проверять работу алгоритма сразу не на всём множестве данных, а брать во внимание только часть пользователей.

$number\_of\_items$ – количество объектов. Для обоих видов данных величина равна 10

$number\_of\_items\_features$ – количество атрибутов у объекта. Для автомобилей данная величина = 4, для суш = 7

$number\_of\_users\_features$ – количество атрибутов у пользователя. Для автомобилей = 4, для суш = 10

Глобальные методы

$global\_fill(fl)$ – инициализирует глобальные параметры в зависимости от fl. Если $fl = 1$, то заполняет параметры для данных об автомобилях, иначе для суш.

Preparer_LPM

Краткое описание

Вспомогательный класс, подготавливает данные для обучения классификатора Vote_LPM [П. 6.4.2].

Статические методы

$get\_parametrs\_type\_cars$ – возвращает вектор структур для данных об автомобилях. Поля данной структуры: $is\_category$ – категориальный ли это признак, ind – индекс атрибута, $left$ и $right$ – отрезок изменения категорий данного признака ([0;0], если признак номинальный).

$get\_parametrs\_type\_sushi$ – возвращает вектор структур для данных о сушах. Поля данной структуры: $is\_category$ – категориальный ли это признак, ind – индекс атрибута, $left$ и $right$ – отрезок изменения категорий данного признака ([0;0], если признак номинальный).

$make\_attr$ – Преобразует атрибуты каждого из объектов по алгоритму, описанному в П. 6.1, возвращает двумерный вектор преобразованных атрибутов.

Vote_LPM

Краткое описание

Классификатор, построенные по методу LPM voting

Приватные поля

$attr\_sz$ – количество атрибутов у одного объекта после подготовки данных в методах П. 6.4.1

num_user – количество пользователей в данных

Публичные поля

$attr\_order$ – двумерный вектор, хранящий лексикографический порядок атрибутов для каждого пользователя

Публичные методы

$learn\_user(user, q, attr)$ – строит лексикографический порядок атрибутов для $user$ по обучающей выборке q.

$learn$ – запускает для каждого пользователя $learn\_user$

$test(q)$ – возвращает количество неправильных ответов, полученных классификатором на выборке q.