Санкт-Петербургский государственный университет
Кафедра математической теории игр и статистических решений

Казаков Данил Игоревич

Выпускная квалификационная работа бакалавра

Рекомендация тегов задач в одном онлайн сервисе управления проектами

Направление 010400
Прикладная математика и информатика

Научный руководитель,
доктор технических наук
профессор
Буре В.М.

Санкт-Петербург
2017

Содержание

Введение

На сегодняшний день великое множество сервисов (например, предоставляющие некоторые продукты: фильмы, музыку и т.п.) заинтересованы в том, чтобы помочь пользователям в выборе тех или иных действий, имея некоторую информацию о них. Данная проблематика довольно распространена, потому что самостоятельно пользователь может потратить очень много времени на поиск интересуещей его информации.

Например, интернет-сервис продажи товаров Amazon использует информацию о совершившихся покупках, чтобы в дальнейшем предлагать каждому конкретному пользователю товары, в которых они нуждаются и о которых даже не подозревал. Таким образом данный сервис сокращает время покупателей, проведённое за поиском товаров, и увеличивает прибыль.

В данной работе будет рассматриваться онлайн сервис управления проектами Wrike. Данный сервис обеспечивает онлайн-среду для рабочего взаимодействия в команде (позволяет планировать проекты и отслеживать график их выполнения).

При создании некоторой задачи (проекта) автор приписывает ей совокупность ключевых слов (метка, далее мы будем использовать термин "тег"; с англ. tag — ярлык), чтобы обозначить вкратце суть задачи и отнести её к конкретной категории для дальнейшей успешной навигации.

Не редко случается ситуация, что некоторый работник может регулярно создавать некоторую задачу и приписывать ей одни и те же теги. Зачастую такая процедура несет в себе рутинный характер.

В работе рассматривается задача рекомендации тегов (т.е. предсказание действий пользователей, основываясь на истории их поведения в онлайн сервисе управления проектами). В работе изучаются известные методы и алгоритмы, которые рассматриваются в контексте задачи рекомендации тегов. Все рассматриваемые модели были программно реализованы (в некоторых случаях использовались готовые пакеты) и применены к действительным данным. Для каждого метода получена оценка качества.

Работа состоит их трех глав. В первой главе изучается вероятнотная модель на основе цепей Маркова, где отдельно изучается конечная цепь Маркова и предлагается идея экспоненциального сглаживания для

адаптивного обновления переходных вероятностей. Также в первой главе рассматривается алгоритм контекстного прогноза, структуры скрытой марковской модели (СММ) и алгоритм Баума — Велша для нахождения наиболее вероятных параметров по последовательности наблюдаемых величин. Во второй главе изучается перцептронная сеть и метод обратного распространения ошибки. В третьей главе рассматривается задача поиска ассоциативных правил с применением алгоритма Apriori.

Постановка задачи

Сформулируем задачу данной работы: по данным поведения пользователей составить модель пронозирования тегов для каждого индивидуального пользователя и применить её к действительным данным.

Для анализа поведения пользователей имеется в распоряжении данные о действиях сотрудников одной компании в течение некоторого времени. Данные представляют собой историю поведения пользователей, т.е. какие теги приписывали пользователи к конкретным задачам. Для каждого факта использования тега фиксируется время (мы можем проранжировать историю поведения по времени). Отметим, что в онлайн сервисе [14] пользователь может быть либо автором задачи, либо её испольнителем (т.е. автор задачи может назначить других пользователей в качестве исполнителей). Тип пользователя для каждой задачи также отражен в данных.

Реальные данные играют ключевую роль: только основываясь на анализе структуры данной информации, мы можем построить математическую модель и оценить качество этой модели.

В ходе данной работы были поставлены следующие задачи:

1. Изучить структуру тегов для дальнейшего подбора математических моделей.
2. Составить вероятностную модель задачи рекомендации тегов на основе конечных цепей Маркова с применением адаптивного обновления матриц переходных вероятностей.
3. Изучть алгоритм контекстного прогноза в терминах задачи рекомендации тегов.
4. Рассмотреть структуру скрытой марковской модели и изучить алгоритм Баума — Велша.
5. Исследовать структуру нейронной сети и метод обратного распространения ошибки.
6. Изучить алгоритм поиска ассоциативных правил Apriori.
7. Для каждого метода написать программную реализацию или использовать готовые пакеты (в языке программирования R).

8. Применить программы к реальным данным и получить оценку моделей.

Обзор литературы

В учебниках [1] и [2] расматриваются и строго формализуются конечные цепи Маркова. Основные определения взяты именно оттуда для дальнейших предположений относительно наших данный и построения вероятностной модели .

В работе [3] исследуется задача предсказания положения человека в офисе по истории его перемещения (в качестве положения понимается комната в офисе). Эта же задача рассматривается в работах [6] и [9]. В качестве исследуемых методов авторы предложили алгоритм, который опирается на марковское свойство, но при этом вместо вероятностей подсчитывает специфичные характеристики.

В учебном пособии [4] рассматривается метод экспоненциального сглаживания для временных рядов. Данный подход был предложен для обновления переходной матрицы конечной цепи Маркова. Также в этом пособии вводится понятие среднего возраста информации, которое позволяет легко подобрать параметр сглаживания.

В работах [5] и [6] изучается скрытая марковская модель и приводится алгоритм Баума-Велша для поиска наилучшей модели по наблюдениям. Скрытая марковская модель позволяет расширить структуру нашей модели последовательностью скрытых состояний, которые влияют на появление тегов.

В книге [7] рассматривается структура модели нейронной сети. В работах [7] и [8] описан метод обратного распространения ошибки для обучений нейронной сети. В работе [9] авторы предложили представлять натуральное число в виде двоичного кода, и далее использовать нейронную сеть для обучения. Данная идея даёт возможность рассматривать нашу задачу в терминах модели нейронной сети.

В книге [10] рассматривается понятие ассоциативного правила и приводятся хорошие примеры для понимания. В работе [11] предлагается алгоритм поиска ассоциативных правил Apriori, основанный на априорных предположениях.

Глава 1. Вероятностная модель

1.1 Анализ структуры данных

Как и оговаривалось ранее, каждый тег представляет собой некое ключевое слово или фразу. В целом полный набор уникальных тегов может быть бесконечен. Но в рамках нашей задачи мы можем говорить, не умаляя общности, что набор тегов конечен (потому что компания использует ограниченное число тегов в рамках своей области и сотрудникам необходимо иметь общую для всех систему для организованной деятельности).

При изучении наших данных выяснилось, что количество уникальных тегов составляет менее 10% от общего числа тегов. Данное замечание говорит о том, что появление новых тегов не несёт в себе хаотичный характер.

Предположим, что при длительной работе команды отношение уникальных тегов к общему числу тегов будет стремиться к 0:

$$\lim_{n\to\infty} \frac{k}{n} = 0,$$

где k — количество уникальных тегов, n — общее количество тегов, зафиксированных при работе.

Ввиду данного предположения мы в будем рассматривать тег как дискретную величину (т.е. мы можем пронумеровать теги), принимающую значения от 1 до k, где k — количество уникальных тегов.

Для каждого факта появление нового тега фиксируется время. Отсюда мы можем рассматривать историю поведения пользователей как последовательность натуральных чисел (про распределение который мы, к сожалению, ничего не можем сказать).

В данной главе мы будем рассматривать только те случаи, когда при задании тега к данной задачи пользователь являлся автором этой задачи. Такой шаг сделан из следующих соображений: если некоторого работника назначили испольнителем задачи, то он становится зависимым от автора задачи (а именно от времени, когда была создана задача) и тогда теги, приписанные к этой задаче, могут внести некоторый "шум"в историю поведения пользователя.

1.2 Конечные цепи Маркова

Рассмотрим последовательность испытаний a_t, $t = 1, 2, ..., T$, в каждом из которых возможно k исходов из множества $\{1, ..., k\}$, где a_t — функция, которая принимает значение $r_i \in \{1, ..., k\}$, если r_i — исход t-го испытания.

В рассматриваемой задаче мы можем рассматривать генерацию тега уже как некоторое "испытание". Конечно, когда пользователь создаёт очередную задачу и задаёт определённые теги, он не воспринимает генерацию тега как эксперимент в своей голове: он точно знает какие теги необходимы для каждой задачи. Но для модели это выглядит как последовательность экспериментов. Такой подход к нашим данным позволяет воспользоваться основополагающими идеями теории конечных цепей Маркова.

В ходе рассмотрения определений теории конечных цепей Маркова мы будем выдвигать предположения относительно нашей структуры данных.

Определение. [1] *Конечный марковский процесс* — конечный стохастичесий процесс, в котором для любого момента времени $t = 1, ..., T$, любого исхода $r_j \in \{1, ..., k\}$ и исходов предшевствующих испытаний $r_i \in \{1, ..., k\}, i = 1, ..., t-1$ выполняется условие:

$$P\{a_t = r_j | a_{t-1} = r_i, ..., a_1 = r_m\} = P\{a_t = r_j | a_{t-1} = r_i\} \tag{1}$$

Под записью $P\{A|B\}$ мы понимаем условную вероятность события A при условии, что событие B произошло.

Определение. [1] Если вероятность события B отлична от нуля, то условная вероятность события A при условии, чир событие B произошло, определяется формулой

$$P\{A|B\} = \frac{P\{A \cap B\}}{P\{B\}} \tag{2}$$

Первое предположение: вероятность значения нового тега зависит только от значения предыдущего тега. Такое предположение довольно жёсткое и не отражает всех зависимостей, но в то же время оно необходимо для нашей модели.

Определение. [2]Переходные вероятности на t-ом шаге, которые будем

обозначать $p_{ij}^{(t)}$, это

$$p_{ij}^{(t)} = P\{a_t = r_j | a_{t-1} = r_i\} \tag{3}$$

Определение. [2]*Конечной цепью Маркова* называется конечный марковский процесс, для которого переходные вероятности $p_{ij}^{(t)}$ не зависят от t.

Определение. [2]Квадратная матрица с неотрицательными элементами, у которой сумма элементов в любой строке равна единице, называется *стохастической.*

Определение. [2]Переходной матрицей *цепи Маркова называется стохастическая матрица P с элементами p_{ij}.* Вектором начальных вероятностей (*или* начальным распределением) *называется вектор* $\pi_0 = \{p_j^0\}$.

В этом случае, переходные вероятности мы будем обозначать p_{ij}. Индексация ij означает i-ую строку и j-ый столбец в переходной матрице P.

Второе предположение: переходные вероятности не зависят от t. Данное предположение позволяет для каждого пользователя оперировать лишь одной матрицей.

Для решения данной задачи можно обобщить модель Маркова, а именно: мы можем предполагать, что значение текущего тега зависит от p предыдущих тегов. Тогда вероятность значения текущего тега будет равна:

$$P\{r_t | r_{t-1}, r_{t-2}, ..., r_{t-p}, ..., r_1\} = P\{r_t | r_{t-1}, r_{t-2}, ..., r_{t-p}\} \tag{4}$$

Необходимо отметить, что при увеличении параметра p количество переходных вероятностей будет иметь степенную зависимость, и для хорошей модели необходима будет обширная история пользователей.

Предложим наивный алгоритм прогнозирования для модели с параметром $p = 1$ (Данный алгоритм легко обобщается в случае $p \geqslant 2$). Для конечной марковской цепи $\omega = (a_1, a_2, ..., a_T)$ введём индикаторную функцию $I_{ij}(t, \omega)$, $t = 2, 3, ..., T$:

$$I_{ij}(t) = \begin{cases} 1, & \text{если } a_t = r_j \text{ и } a_{t-1} = r_i \\ 0, & \text{если } a_t \neq r_j \text{ или } a_{t-1} \neq r_i \end{cases} \tag{5}$$

Алгоритм прогозирования:

1. Для каждого пользователя u по конечной марковской цепи $\omega^{(u)} = (a_1^{(u)}, ..., a_T^{(u)})$ мы построим переходную матрицу $P_{k\times k}^{(u)}$, где

$$p_{ij}^{(u)} = \frac{\sum_{t=2}^{T} I_{ij}(t, \omega^{(u)})}{\sum_{m=1}^{k} \sum_{t=2}^{T} I_{im}(t, \omega^{(u)})} \tag{6}$$

   Заметим, что по построению матрица $P_{k\times k}^{(u)}$ является стохастической.

2. Для каждого нового действия пользователя (т.е. введения тега $r_i \in \{1, 2, ..., k\}$) наиболее вероятный тег считается по формуле:

$$r_{predict} = \max_{1\le j\le k} p_{ij}^{(u)} \tag{7}$$

   Также можно предлагать пользователю не один, а несколько наиболее вероятных тегов на выбор.

Далее в целях удобства мы опустим индекс (u), обозначающий конкретного пользователя, и будем считать, что любое вычисление проходит в рамках одного пользователя.

1.3 Экспоненциальное сглаживание

В ходе работы того или иного пользователя мы будем получать новую информацию о его поведении. Необходимо придумать подход учёта новой информации.

Сразу оговрим следующую поправку: постоянно растущую конечную марковскую цепь мы будем разбивать на временные блоки для обновления информации. И поэтому в данном параграфе индекс $t, t = 1, 2, ...$ будет обозначать t-ый промежуток времени.

Можно привести простой метод: через определённые промежутки времени (реального или измеренного через количество новой информации) обновлять переходную матрицу P_t по всей истории пользователя. Тогда с течением времени мы будем накапливать всю информацию о пользователе.

Но при анализе поведения некоторых пользователей было замечено, что некоторые ключевые слова в определённый момент выходили из употребления. Данный факт можно объяснить тем, что сотрудник мог сменить должность (в таком случае его набор тегов может измениться) или начать работу над новым проектом.

Для быстрой адаптации переходной матрицы P к новой информации можно использовать другой подход: пересчитывать матрицу P не по всей известной истории, а только взятой за последний период (т.е. за последний промежуток времени). Такой подход более применим к нашей задачи, но он является слишком жестким, потому что полностью игнорирует старую информацию.

В ходе поиска метода, который бы быстро адаптировался и оставался консервативен к информации, была изучина работа Лукашина Ю.П. [4], в которой был рассмотрен метод экспоненциального сглаживания для временного ряда. Данную идею можно применить к нашей переходной матрицы.

Обновление переходной матрицы осуществляется по рекуррентной формуле [3]:

$$\hat{P}_t = \alpha P_t + \beta \hat{P}_{t-1}, \tag{8}$$

где $\hat{P}_t$ – адаптивная переходная матрица в момент времени t (или во временной промежуток t); α – параметр адаптивная, $\alpha = const$, $0 \leq \alpha \leq 1$; $\beta = 1 - \alpha$; $\hat{P}_1 = P_1$

Выражение (7) мы можем переписать следующим образом:

$$\hat{P}_t = \alpha P_t + (1 - \alpha)\hat{P}_{t-1} = \hat{P}_{t-1} + \alpha(P_t - \hat{P}_{t-1}) \tag{9}$$

Параметр α задаёт скорость адаптации нашей переходной матрицы $\hat{P}_t$ к новой информации.

Выбору величины постоянной сглаживания следует уделять особое внимание. Введём как вспомогательный инструмент понятие среднего возраста данных, который был определён в [3]. Возраст наблюдений в текущий промежуток времени равен 0, возраст наблюдений в предыдущий период времени равен 1 и т.д. Средний возраст — это взвешенная сумма возрастов данных, причём веса равны весам соответствующей информации. Вес переходной матрицы с возрастом q равен $\alpha\beta^q$, где $\beta = 1 - \alpha$.

Тогда средний возраст информации k равен:

$$k = 0 \cdot \alpha + 1 \cdot \alpha\beta + 2\alpha\beta^2 + ... = \alpha \sum_{i=1}^{\infty} i\beta^i = \frac{\beta}{\alpha}$$

Понятие среднего возраста информации помогает с выбором параметра сглаживания α. Если задать средний возраст информации k, то параметр α будет вычисляться по следующей формуле:

$$\alpha = \frac{1}{k+1}, k \geqslant 0 \tag{10}$$

Ещё одним преимуществом экспоненциального сглаживания является следующий факт: в нашей задаче переходные матрицы являются сильно разреженными и в памяти вычислительной машины для хранения матриц можно использовать другие структуры данных (например, хеш-таблицы). И если с течением времени некоторая p_{ij} стремится к 0, то её можно удалить из ячейки памяти.

1.4 Контекстный прогноз

В работе [3] был предложен алгоритм контекстного прогноза, который во многом схож с методом из (1.2). В данном алгоритме для каждого состояния $r = (t_i, ..., t_j)$ мы храним только 2 параметра: наиболее вероятный тег tag_k и "степень уверенности"в текущем прогнозе (т.е. при наблюдении события r мы будем прогнозировать tag_k с некоторый степенью уверенности).

Авторы работы [3] разобрали данный метод на примере задачи прогноза положения работника в офисе. Мы же сразу приведём данный [3] алгоритм, т.к. он прост для понимания. Данный метод задаётся двумя параметрами: p – длина "истории"(т.е. тег tag_t зависит от p предыдущих) и max_onf – максимальный уровень уверенности ($1 \leqslant p \leqslant T - 1$)

Пусть для каждого r_i в $A_{r_i}[1]$ содержится прогнозируемое следующее состояние (в начале процедуры $A_{r_i}[1] = \emptyset$), а в ячейке $A_{r_i}[2]$ – степень увренности (в начале $A_{r_i}[2] = 0$).

Алгоритм контекстного прогноза (для поседовательности тегов $tag_1, tag_2, ..., tag_T$):

Последовательно для каждого $t = p+1, p+2, ..., T-1$ выполняем:

1. Состояние r_t будет равно $(tag_{t-p}, tag_{t-p+1}, ..., tag_t)$.

2. Если $A_{r_t}[1] = \emptyset$, то $A_{r_t}[1] = tag_{t+1}$ (т.е. мы ничего не можем рекомендовать и лишь запоминаем ответ для последующего прогноза). Иначе переходим к п.2.

3. Прогнозируем тот тег, который содержится в $A_{r_t}[1]$. Далее мы обновляем модель.

4. Сравниваем значения $A_{r_t}[1]$ и tag_{t+1}. Возможны 2 случая:

   - $A_{r_t}[1] = tag_{t+1}$ (наш прогноз совпал с тегом в момент времени $t+1$). В этом случае $A_{r_t}[2] = max(A_{r_t}[2] + 1, maxConf)$.
   - $A_{r_t}[1] \neq tag_{t+1}$.
     Тогда, если $A_{r_t}[2] = 0$, то $A_{r_t}[1] = tag_{t+1}$ (т.е. мы не были уверены в нашем прогнозе, и поэтому обновляем ответ). Иначе $A_{r_t}[2] = A_{r_t}[2] - 1$ (т.е. уменьшаем нашу степень уверенности).

Параметр модели max_onf можно расценивать как параметр адаптации модели: чем он меньше, тем чаще будет обновляться $A_{r_i}[1]$ при неверном прогнозе.

Главные отличия этого метода от предложенного в 1.2 состоят в следующем:

1. Алгоритм контекстного прогноза быстрее реагирует на запрос (т.е. он уже хранит ответ, в то время как алгоритму из 1.2 необходимо найти наиболее вероятное состояние). Но с другой стороны, алгоритм на основе подсчёта вероятностей может спрогнозировать несколько наиболее вероятных тегов.

2. Алгоритм контекстного прогноза обновляет модель после каждой итерации.

1.5 Скрытые Марсковские цепи

В этом параграфе рассматривается другая модель на основе цепей Маркова — скрытая марковская модель (СММ). В данной модели рассматриваются 2 последовательности: последовательность скрытых состояний и последовательность наблюдаемых состояний. Формализация СММ взята из [5] и [6].

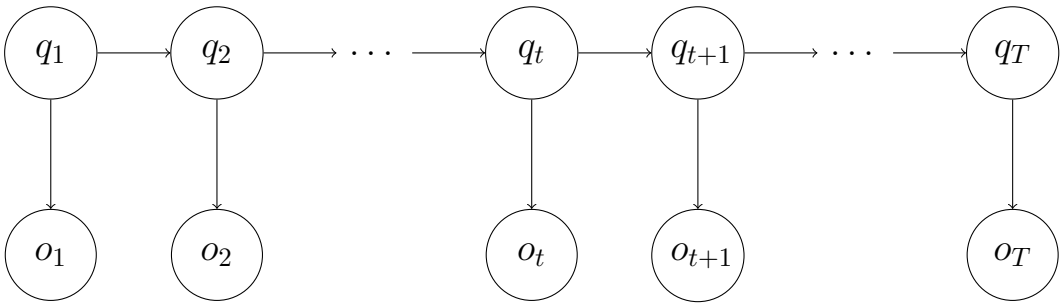

Рис. 1: Структра скрытой марковской модели

Пусть $S = \{S_1, S_2, ..., S_n\}$ — значения скрытых состояний, а $V = \{V_1, V_2, ..., V_m\}$ — значения наблюдаемых состояний. Последовательность $q_t, t = 1, 2, ...$ скрытых состояний образуют конечную цепь Маркова, а это значит, что

$$P\{q_t = S_j \ |q_{t-1} = S_i, q_{t-2} = S_k, ...\} = P\{q_t = S_j|q_{t-1=S_i}\} \tag{11}$$

Для последовательности скрытых состояний мы также вводим основные предположения из параграфа 1.2. В силу данных предположений введём стохастическую матрицу переходов A, для которой:

$$a_{ij} = P\{q_t = S_j|q_{t-1} = S_i\}, \qquad 1 \leqslant i, j \leqslant n \tag{12}$$

Также введём вектор начальных вероятностей π:

$$\pi_i = P\{q_1 = S_i\}, \qquad i = 1, 2, ..., n \tag{13}$$

Определим, наконец, последовательность наблюдаемых состояний $o_1, o_2, ..., o_T$. Каждое наблюдаемое состояние зависит только от соответствующего скрытого состояния, т.е. в момент времени t мы наблюдаем "эмиссию"наблюдаемого состояния.

Введём матрицу B размерности $n \times m$, в которой:

$$b_{jk} = P\{o_t = V_k|q_t = S_j\}, \qquad j = 1, 2, .., n; k = 1, 2, ..., m, \tag{14}$$

где b_{jk} — вероятности "эмиссии"наблюдаемого состояния. Далее, удобства мы будет обозначать вероятности b_{jk} в виде $b_j(k)$.

СММ задаётся с помощью значений A, B, π. Для компактной записи будем использовать обозначение $\mu = (A, B, \pi)$.

Пример. Предположим, что у крупье есть две монеты: одна симметричная, другая — нет. Нам известны вероятности выпадения орла и решки для обоих монет и вероятности смены монеты (пусть нам ещё известны начальные вероятности). Тогда скрытыми состояниями будут та или иная монета (мы не знаем какую монету использует крупье в момент времени t), а наблюдаемыми — выпадение орла или решки.

Рассматривают 3 основные задачи для скрытых цепей Маркова:

1. Вычисление вероятности данной последовательности для модели μ.
2. Подсчёт наиболее вероятных последовательностей скрытых состояний по последовательности наблюдаемых состояний и модели μ
3. По последовательности наблюдаемых состояний определить наилучшею модель μ.

Для задачи рекомендации тегов мы рассмотрим только 3 задачу. Для решения данной задачи был предложен алгоритм Баума — Велша [5]. Перед рассмотрением самого алгоритма введём следующие величины (все определения и алгоритмы взяты из работ [5, 6]):

- $\alpha_t(i)$ — вероятность данной последовательности наблюдений $o_1, o_2, ..., o_t$ и скрытого наблюдения S_i в момент времени t при данной модели μ:

$$\alpha_t(i) = P\{o_1, o_2, ..., o_t, q_t = S_i|\mu\}, \tag{15}$$

Алгоритм нахождения $\alpha_t(i)$ выглядит следующим образом:

1. *Инициализация*:

$$\alpha_1(i) = \pi_1 b_j(o_1), \qquad i = 1, 2, ..., n \tag{16}$$

2. *Индукция*:

$$\alpha_{t+1}(j) = \left(\sum_{i=1}^{n} \alpha_t(i) a_{ij}\right) b_j(o_{t+1}), \qquad i = 1, 2, ..., n \tag{17}$$

- $\beta_t(i)$ — вероятность данной последовательности наблюдений

$o_{t+1}, o_{t+2}, ..., o_T$ и скрытого наблюдения S_i в момент времени t при данной модели μ.

$$\beta_t(i) = P\{o_{t+1}, o_{t+2}, ..., o_T, q_t = S_i|\mu\}, \tag{18}$$

Алгоритм вычисления $\beta_t(i)$:

1. *Инициализация*:

$$\beta_T(i) = 1, \qquad i = 1, 2, ..., n \tag{19}$$

2. *Индукция*:

$$\beta_t(i) = \sum_{i=1}^{n} a_{ij} b_j(o_{t+1}) \beta_{t+1}(j), \qquad i = 1, 2, ..., n; t = T-1, T-2, ... \tag{20}$$

- Вероятность $\xi_t(i, j)$ прохождения ребра $i \to j$ при последовательности наблюдении O и модели μ:

$$\xi_t(i, j) = P\{q_t = S_i, q_{t+1} = S_j|O, mu\} = \frac{\alpha_t(i) a_{ij} b_j(o_{t+1}) \beta_{t+1}(j)}{\sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_t(i) a_{ij} b_j(o_{t+1}) \beta_{t+1}(j)} \tag{21}$$

- Вероятность $\gamma_t(i)$ нахождения во время t в состоянии i:

$$\gamma_t(i) = \sum_{j=1}^{n} \xi_t(i, j) \tag{22}$$

Величину $\sum_{t=1}^{T-1} \gamma_t(i)$ как ожидаемое число переходов из состояния S_i, а $\sum_{t=1}^{T-1} \xi_t(i, j)$ как ожидаемое число переходов их состояния S_i в S_j.

Теперь мы можем сформулировать алгоритм Баума — Велша [5] нахождения параметров модели μ по наблюдениям $O = (o_1, o_2, ..., o_T)$

1. Инициализируем случайным образом π, A, B (т.е. задаём случайные вероятности модели).

2. На каждом шаге считаем вероятности (21) и (22).

3. Обновляем параметры модели:

$$\hat{\pi}_i = \gamma_1(i) \tag{23}$$

$$\hat{a}_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)} \tag{24}$$

$$\hat{b}_j(k) = \frac{\sum_{t=1;o_t=V_k}^{T} \gamma_t(j)}{\sum_{t=1}^{T} \gamma_t(j)} \tag{25}$$

4. Повторяем п.2 и п.3 до схождения вероятностей (в качестве критерия остановки можно выбрать норму разностей матриц или критерий, предложенный в [6]).

В работе [5] показана сходимость $P\{O|\hat{\mu}\}$ к локальному минимуму.

В работе [6] алгоритм Баума — Велша [5] был применён к задаче прогноза положения работника в офисе (которая была упомянута при обзоре литературы). Данный подход дал очень хорошие результаты.

Основываясь на работе [6], формализуем задачу рекомендации тегов в терминах скрытой Марковской модели: в качестве последовательности наблюдаемых состояний мы будем рассматривать последовательность тегов, а в качестве скрытых состояний мы можем использовать тип пользователя (см. 1.1) или же просто задать количество скрытых состояний (т.е. мы только предлагаем зависимость тегов от некоторых скрытых состояний). В первом случае мы сразу можем определить параметры модели μ, во втором случае нам необходимо воспользоваться алгоритмом Баума — Велша [5].

После определения параметров модели мы можем предложить алгоритм [6] рекомендации следующего наблюдения o_{t+1} по истории наблюдений $o_1, o_2, ..., o_t$:

1. Вычислить наиболее вероятное скрытое состояние S_t:

$$S_t^* = \arg\max_i \alpha_t(i), \qquad i = 1, 2, ..., n.$$

2. Вычислить наиболее вероятное скрытое состояние S_t:

$$S_{t+1}^* = \arg\max_j a_{S_t^* j}, \qquad j = 1, 2, ..., n.$$

3. Предсказать значение следующего состояния V_k (при $t+1$) по формуле:

$$V_{predict} = \arg\max_k b_{S_{t+1}^*}(k), \qquad k = 1, 2, ..., m.$$

1.6 Программная реализация

Для изучения данных и реализации вероятностых моделей использовался язык программирования R и среда разработки $RStudio$. Код программы содержится в приложении.

В качестве меры оценки прогноза используется функция точности:

$$accuracy = \frac{k}{n}, \tag{26}$$

где k — количество совпадений рекомендуемого тега и действительного тега, n — общее число тегов в тестовой последовательности.

Опишем алгоритм, основанный на подсчете переходных вероятностей (далее будем его называть алгоритмом вероятностного прогноза):

1. На вход процедуре подаётся следующие параметры: последовательность тегов пользователя, количество тегов в одном временном промежутке, средний возраст информации и количество предлагаемых пользователю тегов.

2. В первый промежуток времени вычисляется переходная матрица $\hat{P}_1 = P_1$ (в случае $p = 2$ – трехмерная таблица).

3. В течение каждого последующего промежутка времени t, $t \geqslant 2$ вычисляется переходная матрица P_t и проводится рекомендация тегов t-го промежутка времени по переходной матрице $\hat{P}_{t-1}$.

4. После каждого промежутка времени t вычисляется переходная матрица $\hat{P}_{t+1}$ по формуле (8).

Алгоритм контекстного прогноза описан в 1.4.

Для сравнения было реализовано 4 модели:

1. Алгоритм вероятностного прогноза с параметрами $p = 1$ и $p = 2$

2. Контекстный алгоритм с параметрами $p = 1$ и $p = 2$.

Также реализована модель, основанная на скрытой марковской модели с помощью встроенного в $RStudio$ пакета HMM. Алгоритм Баума — Велша [5] и алгоритм прогноза [6] следующего наблюдения описаны в 1.5. Было решено оценить модель только для пользователй с наибольшей историей поведения.

1.5 Результаты вычислений

В ходе анализа структуры данных было выяснено, что в среднем пользователь использует 10 тегов. Можно сделать вывод о том, что рекомендательная система, прогнозирующая случайный тег, обладала бы мерой точности 10%. Далее точность выражена в процентах. Отметим, что каждый тег в среднем упонимается в данных три раза. Это очень маленький показатель для определения каких-либо закономерностей. Но тем не менее мы оцениваем нашу модель на абсолютно всех данных.

Сперва посмотрим зависимость контекстного алгоритма от параметра $maxConf$:

	Max. conf = 0	Max. conf = 2	Max. conf = 5
p = 1	50.94	50.76	49.83
p = 2	54.77	54.57	54.11

Заметим, что в целом параметр $ax.onf$ не оказывает влияния, а параметр p влияет на точность модели 4%.

Далее приступим к рассмотрению алгоритма вероятностного прогноза. Сначала посмотрим на зависимость алгоритма от количества временных блоков Num blocks (при этом параметр $\alpha = 0.33$):

	Num. of blocks = 5	Num. of blocks = 10	Num. of blocks = 20
p = 1	35.41	38.2	40.0
p = 2	38.92	40.63	40.44

При увеличении числа блоков точность модели при $p = 1$ растёт стремительней, но для всех значений мараметра модель с $p = 2$ показывает наибольшую точность.

Рассмотрим зависимость алгоритма от параметра среднего возраста данных Avg. age (при этом Num. of blocks = 20):

	Avg. age = 0	Avg. age = 1	Avg. age = 2
p = 1	40.48	38.59	39.4
p = 2	44.28	40.35	41.17

На наших данных хороший результат модели демонстрируют при Avg. age = 0 (т.е. при мнгновенной адаптации).

Наконец, рассмотрим влияние числа предлагаемых тегов $Num.oftags$ на точность модели ($Avg.age = 0, Num.ofblocks = 20$):

	Num. of tags = 1	Num. of tags = 2	Num. of tags = 3
p = 1	40.48	50.50	55.84
p = 2	44.28	50.68	57.99

Как видно из таблиц, для наших данных алгоритм контекстного прогноза работает лучше вероятностного алгоритма.

Скрытая марковская модель (с тремя скрытыми состояниями) дала прирост точности 10% (по сравнению с вероятностным и контекстным алгоритмами) для пользователей с наибольшей историей (с 30% до 40%). Нельзя утверждать, что модель хорошо описывает наши данные. Но необходимо отметить, что данная модель была испытана на пользователях, для которых количество уникальных тегов равно 40, а вся история состоит всего из 300 наблюдений. Делаем вывод, что для хорошего обучения модели необходимо иметь очень большую историю поведения пользователей, которой у нас нет в данных.

В целом отметим, что все три модели дают неплохие результаты (при том, что существуют пользователи, для которые доля уникальных тегов составляет более 50%).

Глава 2. Нейронные сети

Нейронные сети представляют широкий класс моделей, которые различаются в зависимости от:

- решаемой задачи (регрессия, классификация и т.п.).
- архитектуры модели.
- алгоритма обучения сети.

Для решения задачи распознавания тегов мы рассмотрим персептроннную сеть.

2.1 Персептронная нейронная сеть

Персептронная нейронная сеть состоит из персептронов. Персептроны являются математической моделью нейрона головного мозга.

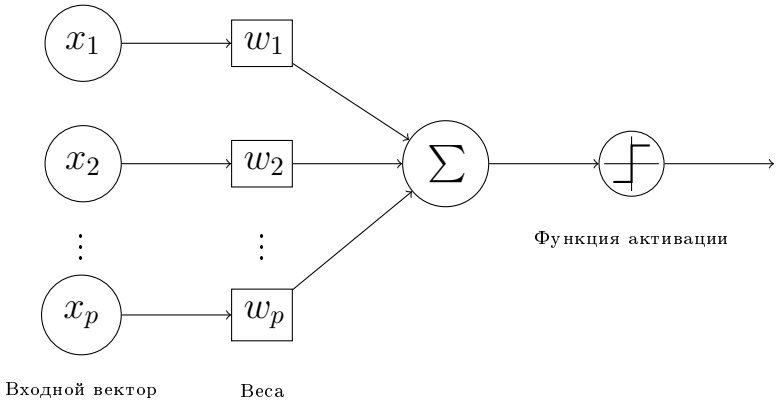

Рис. 2: Модель одного персептрона

Опишем принцип действия одного персептрона [7]:

1. На вход перцептрону подаётся несколько значений $x_1, x_2, ..., x_p$ (от других прецептронов или из входного слоя) с весами $w_i, i = 1, 2, ..., p$.
2. Вычисляется линейная комбинация I всех входов:

$$I = \sum_{i=1}^{p} x_i w_i$$

3. Активационная функция s преобразует линейную комбинацю I в выходной сигнал y.
4. Выходной сигнал y поступает к другим нейронам или является выходом нейронной сети.

В данной работе мы будем использовать логистическую пороговую функцию s:

$$s(x) = \frac{e^x}{1 + e^x} \tag{27}$$

Необходимо подчеркнуть, что значения логистической функции $s(x) \in (0, 1)$.

В структуре нейронной сети персептроны (или нейроны) располагаются послойно. Выделяют следующие типы слоёв [7]:

- *Входной слой.* Нейроны входного слоя имеют только один вход от внешней среды.
- *Скрытый слой.* Каждый нейрон скрытого слоя связан с каждым нейроном предыдущего слоя.
- *Выходной слой.* Выход каждого нейрона является откликом сети.

Число скрытых слоёв и количество нейронов в каждом слое может быть каким угодно.

Отдельно подчеркнём ограничения, накладываемые на структуру нейронной сети:

- В пределах слоя нейроны не связаны.
- Нейроны передают информацию только нейронам последующего слоя.

Под *обучением нейронной сети* мы будем понимать определение значений весов.

2.2 Метод обратного распространения ошибки

Для того, чтобы обучить нейронную сеть по данным $(x_{1i}, x_{2i}, ..., x_{pi}, y_i)$, $i = 1, 2, ...n$ необходимо ввести функцию ошибки (и далее её минимизировать). В данном параграфе рассматривается метод обратного распространения ошибки (backpropagation), изложенный в работах [7] и [8].

Пусть $X = (x_1, x_2, ..., x_p)$ – некоторые входные значения, для которых мы знаем ответ Y. Для того, чтобы подсчитать ошибку нейронной сети, необходимо:

1. Задать какие-то веса и подать сигнал $X_i = (x_{1i}, x_{2i}, ..., x_{pi})$ на входной слой для каждого i.

2. Последовательно для каждого слоя вычислить линейную комбинацию I и применить функцию активации s.

3. На выходном слое получим значение v_i.

Тогда функция ошибки будет выглядеть следующим образом:

$$E = \frac{1}{2}\sum_{i=1}^{n}(v_i - y_i)^2 \tag{28}$$

Отметим, что функция ошибки E может задаваться разными способами, но для нашей задачи мы используем функцию (28).

Для минимизации функции ошибки используется метод градиентоного спуска:

$$w_{new} = w_{old} - \alpha\frac{\partial E}{\partial w} \tag{29}$$

,где α – параметр обучения.

Распишем метод обратного распространения ошибки для фунции активации (27) и функции ошибки (28) в случае одного скрытого слоя. Обозначим $y^l = s(W^l y^{l-1})$, $y_0 = X$, где W^l – матрица весов l-го слоя.

Производная логистической функции (27) равна

$$\frac{ds}{dx} = \frac{-1}{1+e^{-x}}(-e^{-x}) = s(1-s) \tag{30}$$

Для выходного слоя производная для каждого веса w^2_{jk} с учетом (30) равна

$$\frac{\partial E}{\partial w^2_{jk}} = \frac{\partial E}{\partial y^2_j}\frac{\partial s}{\partial w^2_{jk}} = (v_k - y^2_k)y^2_k(1-y^2_k)y^1_j \tag{31}$$

Для внтреннего слоя для w^1_{ij} с учетом (30) производная равна

$$\frac{\partial E}{\partial w^1_{ij}} = \left(\sum_k^p \frac{\partial E}{\partial y^2_k}\frac{\partial s}{\partial y^1_j}\right)\frac{\partial s}{\partial w^1_{ij}} = \left(\sum_k^p (v_k - y^2_k)y^2_k(1-y^2_k)w^2_{jk}\right)y^1_j(1-y^1_j)x_i \tag{32}$$

2.3 Решение задачи рекомендации тега с помощью нейронной сети

В работе [9] был предложен следующий подход для задачи прогноза с помощью нейронной сети: представить каждое натуральное число из нашей последовательности в виде некоторого вектора характеристик, а именно в виде двоичного кода (каждый элемент полученного вектора будет иметь значение либо 0, либо 1.), и после решать задачу с помощью нейронной сети.

Приемуществом данного подхода является следующий факт: при увеличении количества нейронов на входном слое сети можно увеличить число связей (а значит и число весов) на каждом слое. Такой подход усложняет архитектуру нейронной сети.

Недостатком подхода является зависимость модели от способа нумерации состояний, а именно: мы сопоставляем каждому тегу некоторое натуральное число, и проблема заключается в том, что способ нумерации не является единственной.

Каждую последовательность тегов $(x_1, x_2, ..., x_n)$ мы представляем в виде пар входных значений и отклика:

$$(x_t, x_{t+1}, ..., x_{t+p}) : x_{t+p+1}, t = 1, ..., n-p-1; 0 \leqslant p \leqslant n-1 \qquad (33)$$

Параметр p отражает следующее *предположение*: значение каждого тега зависит от значений p предыдущих тегов.

Для программной реализации использовался язык программирования *Python*. Код программы приложен к работе. Для данной задачи использовалась нейронаая сеть с одним внутренним слоем, активационной функцией (27) и функцией ошибки (28). Для обновления весов использовались формулы (30) – (32). Для оценки модели мы разделили выборку на 2 подвыборки: тренировочную и тестовую в соотношении 4 к 1. По тренировочным данным мы обучили модель, а по тестовым оценивали модель при помощи функции точности.

Необходимо заметить важную особенность. Как уже отмечалось ранее, логистическая функция принимает значения от 0 до 1, и поэтому на выходном слое в качестве ответа мы получаем вектор значений от 0 до 1. Но мы оговаривали ранее, что вектора наших данных состоят исклю-

чительно из 0 и 1 (по построению). Поэтому был реализован следующий подход: те значения, которые меньше 0.5 мы относим 0, а которые больше – к 1.

Для оценки модели было принято решение отобрать несколько пользователей с наибольшей историей, чтобы нейронная сеть хорошо обучилась. Также отметим, что была взята полная история пользователя (т.е. были выбраны и те случаи, когда пользователь был исполнителем).

Были опробованы 2 модели: при $p = 1$ и $p = 2$. Результаты представлены в следующей таблице:

	user 1	**user 2**	**user 3**	**user 4**	**user 5**
p = 1	0.51	0.46	0.53	0.77	0.55
p = 2	0.50	0.79	0.50	0.85	0.56

Как видно из таблицы, нейронная сеть вполне может справиться с задачей рекомендации тегов. Но проблема заключается в том, что для каждого случая необходимо подбирать отдельно структуру нейронной сети (смена модели для $user2$ дала прирост точности в 30%). Для обучения глобальной нейронной сети в работе [9] было предложено следующее решение: помимо входных значений можно подавать на вход нейронной сети вектор, характеризующий пользователя. Данный подход не рассматривается в нашей работе.

Глава 3. Ассоциативные правила

3.1 Формализация задачи

Пусть $I = \{i_1, ..., i_m\}$: конечный набор объектов, $T = \{t_1, ..., t_n\}$ — набор транзакци. Каждая транзакция t_i содержит набор $X \subseteq I$, если $X \subseteq t_i$.

Для набора $X \subset I$ определим величину $\sigma(X) = |\{t_i : X \subseteq t_i, t_i \in T\}|$.

Определение. [10] **Ассоциативное правило** – это импликация вида $X \Rightarrow Y$, где $X, Y \subset I, X, Y \neq \emptyset$

Для того, чтобы измерить эффективвность ассоциативного правила необходимо ввести некоторые характеристики.

Определение. [10] *Поддержкой (support)* или частотностью ассоциативного правила $X \Rightarrow Y$ называется величина $support(X \Rightarrow Y)$, равная

$$support(X) = P\{X \cup Y\} = \frac{|\{t_i : X \cup Y \subseteq t_i, t_i \in T\}|}{|T|} \tag{34}$$

Определение. [10]*Значимостью (confidence)* ассоциативного правила $X \Rightarrow Y$ называется величина $conf(X)$, равная

$$conf(X \Rightarrow Y) = P\{Y|X\} = \frac{support(X \cup Y)}{support(X)} \tag{35}$$

Задача поиска ассоциативных правил является состоит в следующем: задав набор транзакций T, необходимо найти все ассоциативные правила, для которых:

- Поддержка не меньше некоторого порога min_sup.
- Значимость не меньше порога min_conf.

Для этого задачу разбивают на 2 подзадачи:

- Поиск частных наборов (frequent itemsets), т.е. наборов с уровнем поддержки не меньше min_sup.
- Используя частные наборы, построить ассоциативные правила с уровнем значимости не меньше min_conf.

3.2 Алгоритм Apriori

Алгоритм [11] Apriori был предложен Р.Аграви и Р.Шрикантов в 1994 году. Данный алгоритм основывается на некторых "априорных"знаниях для поиска ассоциативных правил.

Первое свойство Apriori: *Все непустные подмножества частого множества также являются частыми.*

Данное свойство основано на следующем наблюдении. По определению, если набор данных X не удовлетворяет минимальному порогу поддержки, то данный набор не является частным: $supp(X) < min\_sup$. Если добавить некоторый элемент A к набору X, то получившийся набор (т.е. $I \cup A$) не может встречаться чаще, чем X. Т.е. набор $X \cup A$ не является частным: $supp(X \cup A) < min\_sup$.

Пусть C_k — множество кандидатов k-элементных наборов.

F_k: множество частных k-элементных наборов.

Алгоритм [11] Apriori (генерация частных наборов):

1. $C_1 = I$. Исключаем из C_1 наборы, для которых поддержка меньше порога min_sup, тем самым образуем множество L_1.

2. Определив L_1, генерируем множество кандидатов 2-элементных наборов C_2.

3. Определяем поддержку для каждого кандидата из C_2. Исключаем наборы, для которых, поддержка меньше min_sup.

4. Для $k > 2$ определяем множество F_k, исключая наборы, для которых поддержка меньше min_sup. Далее генерируем множество кандидатов C_{k+1}.

5. Повторяем п.4 до тех пор, пока $F_k \neq \emptyset$, т.е. останавливаемся, когда нет возможности найти новые частные наборы.

Существует несколько подходов генерации множества кандидатов C_k. Мы рассмотрим подход, описанный в [10]: метод $L_{k-1} \times L_{k-1}$. При данном подходе объединяются только те наборы из L_{k-1} $A = (a_1, ..., a_{k-1}) \in L_{k-1}$ и $B = (b_1, ..., b_{k-1}) \in L_{k-1}$, для которых $a_i = b_i, i = 1, ..., k-2$ и $a_{k-1} \neq b_{k-1}$.

Необходимо заметить, что следует сортировать множества в лексикографическом порядке во избежание дублей.

Ассоциативное правило может быть получено разбиением Y на два непустых подмножества X и $Y \backslash X$ так, что $X \Rightarrow Y \backslash X$ удовлетворяет ограничению по min_conf.

Второе свойство Apriori: Если ассоциативное правило $X \Rightarrow Y \backslash X$ не удовлетворяет ограничению по min_conf, т.е.

$$conf(X \Rightarrow Y \backslash X) < min\_conf,$$

тогда любое ассоциативное правило $X' \Rightarrow Y \backslash X'$, где $X' \in X$ также не удовлетворяет ограничению по min_conf, т.е.

$$conf(X' \Rightarrow Y \backslash X') < min\_conf.$$

Действительно, если $X' \in X$, то $\sigma(X') \geqslant \sigma(X)$. Тогда

$$min_conf > conf(X \Rightarrow Y) = \frac{\sigma(X \cup Y \backslash X)}{\sigma(X)} = \frac{\sigma(Y)}{\sigma(X)} \geqslant$$
$$\geqslant \frac{\sigma(Y)}{\sigma(X')} = \frac{\sigma(X' \cup Y \backslash X')}{\sigma(X')} = conf(X' \Rightarrow Y).$$

Второе свойство Apriori позволяет сократить число рассматриваемых ассоциативных правил.

3.3 Решение задачи рекомендаци тегов с помощью ассоциативных правил

К каждой задаче пользователи могут приписать несколько тегов. Поиск ассоциативных правил является абсолютно другим подходом к задаче рекомендации тегов. В первой главе вы рассматривали только те случаи создания тега, когда пользователь являлся автором. При поиске ассоциативных правил мы можем снять это ограничение, потому что в терминах ассоциативных правил нам не важен порядок приписывания тега к задачам.

Необходимо отметить, что очень много задач (как выяснилось при анализе структуры данных) содержат только один тег и поэтому не стоит отбрасывать методы и подходы, изученные ранее. Ассоциативные правила можно рассматривать как вспомогательный иструмент для рекомендации тегов.

Для поиска ассоциативных правил мы можем рассматривать данные как матрицу D размерности $n \times k$, где n – количество уникальных задач, k – количество уникальных тегов. Каждый элемент d_{ij} равен 1, если j-ый тег содержится в i-ой задаче.

Для программной реализации и получения результатов был использован язык программирования R и встроенный пакет $arules$ для поиска ассоциативных правил. В качестве результатов выведем количество правил при нескольких параметрах $minmin\_sup$ и min_conf.

supp / conf	0.7	0.8
0.03	7	7
0.01	206	171
0.005	705	634
0.001	5102	4440

Отметим что, всего задач было около 13000 и ассоциативные правила находились по всем данным. Конечно, при уменьшении min_sup количество ассоциативных правил растёт очень быстро. Так же хотелось бы отметить, что для поиска ассоциативных правил для каждого пользователя необходимо иметь широкую историю поведения. Но как пример, данный метод можно использовать для подразделений организаций.

Заключение

В ходе решения задачи рекомендации тегов задач в одном онлайн сервисе управления проектами были изучено и рассмотрено множество методов, которые оценивались по реальным данным поведения пользователей.

В качестве базовой рекомендательной системы была рассмотрена модель на основе конечных цепей Маркова. Для модели были изучены и программно реализованы 2 алгоритма. Был предложен подход адаптивного обновления матрицы переходных вероятностей. Данная рекомендательная система обладает неплохой точностью прогноза.

Также были рассмотрены следующие модели решения задачи прогноза: скрытая марковская модель (и алгоритм Баума — Велша [5] в качестве метода обучения) и нейронная сеть с методом обратного распространения ошибки. Был предложен подход сведения задачи прогноза к данным моделям и программно реализована нейронная сеть. Данные модели лучше описывают поведение пользователей, чем базовая рекомендательная система, но для построения моделей необходим большой объём данных.

Также предложен подход поиска ассоциативных правил как дополнительный инструмент для рекомендательной системы.

Список литературы

[1] Буре В. М., Парилина Е. М. Теория вероятностей и математическая статистика: Учебник. - СПб.: Издательство "Лань" 2013. – 416 с.

[2] Кемени Дж., Снелл Дж. Конечные цепи Маркова. М.: Издательство "Наука 1970. – 272 с.

[3] J. Petzold, F. Bagci, W. Trumler, T. Ungerer. Context Prediction Based on Branch Prediction Methods // Technical Report, University of Augsburg, Germany, 2003.

[4] Лукашин Ю.П. Адаптивные методы краткосрочного прогнозирования временных рядов. Учеб. пособие. – М.: Финансы и статистика, 2003. – 416 с.

[5] Rabiner L.R. A tutorial on hidden Markov models and selected applications in speech recognition // Proceedings of the IEEE. 1989 Volume. 77, p. 257-286.

[6] A. Gellert, and L. Vintan. Person Movement Prediction Using Hidden Markov Models // Studies in Informatics and Control 15 (1), 2006, p. 17-30.

[7] T. Hastie, R. Tibshirani, J. Friedman. The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition. Springer, 2016. – 745 p.

[8] Rumelhart E., David E.; Hinton, Geoffrey E.; Williams, Ronald J. Learning representations by back-propagating errors // Nature. 323 (6088), 1986, p. 533–536.

[9] L. Vintan, A. Gellert, J. Petzold, T. Ungerer. Person Movement Prediction Using Neural Networks // Proceedings of the KI2004 International Workshop on Modeling and Retrieval of Context, At Ulm, Volume: 114, 2004.

[10] Jiawei Han, Micheline Kamber, Jian Pei. Data Mining: Concepts and Techniques, Third Edition. Morgan Kaufmann Publishers, 2011. – 740 p.

[11] Rakesh Agrawal and Ramakrishnan Srikant Fast algorithms for mining association rules in large databases // Proceedings of the 20th International Conference on Very Large Data Bases, VLDB, Santiago, Chile, 1994, p. 487-499.

Приложение

Алгоритм, основанный на конечных марковских цепях с методом адаптивного обновления матрицы переходных вероятностей:

```
prop.model = function(user.tags, time, avg.age, n.max){
uniq = (unique(user.tags))
n.uniq = length(uniq)
prop = matrix(rep(0, n.uniq ^ 2), nrow = n.uniq)
n = length(user.tags)
alpha = 1 / (1 + avg.age)
n.blocks = round(n / time)
n.true = 0
n.all = 0

if (time >= n){
return (0)
}

#Построение матрицы переходных вероятностей
for (i in 2:time){
prev = which(uniq == user.tags[i - 1])
nextt = which(uniq == user.tags[i])
prop[prev, nextt] =  prop[prev, nextt] + 1
}

for (b in 1:n.blocks){
        new.prop = matrix(rep(0, n.uniq ^ 2), nrow = n.uniq)
        for (i in (b * time + 1):min(n, (b + 1) * time)){
        #Построение матрцы переходных вероятностей
        prev = which(uniq == user.tags[i - 1])
        nextt = which(uniq == user.tags[i ])
        new.prop[prev, nextt] =  new.prop[prev, nextt] + 1

        #Оценка качества прогноза
        if (max(prop[prev, ]) > 0){
```

```
                if (user.tags[i] %in% uniq[order(prop[prev, ],
                        decreasing = T)[1:n.max]]){
                        n.true = n.true + 1
                }
                n.all = n.all + 1
                }
        }
        prop = alpha * new.prop + (1 - alpha) * prop
}
return (n.true / n.all)
}
```

Контекстный алгоритм прогноза:

```
context.prediction = function(user.tags, max_conf){
uniq = (unique(user.tags))
n.uniq = length(uniq)
A = matrix(rep(0, n.uniq * 2), nrow = n.uniq)
n = length(user.tags)

n.true = 0
n.all = 0

for (i in 2:n){
        tmp.tag = user.tags[i]
        prev.tag = user.tags[i - 1]
        pos.prev = which(uniq == prev.tag)
        pos.tmp = which(uniq == tmp.tag)

        #Мы не можем ничего предложить
        if (A[pos.prev, 1] == 0){
        A[pos.prev, 1] = pos.tmp
        }else {
        n.all = n.all + 1
        if (A[pos.prev, 1] == pos.tmp){
```

```
                #Увеличиваем степень уверенности
                A[pos.prev, 2] = max(max_conf,
                A[pos.prev, 2] + 1)
                n.true = n.true + 1
        }else if (A[pos.prev, 2] == 0){
        #Изменяем прогноз
                A[pos.prev, 1] = pos.tmp
        }else{
                #Уменьшаем степень уверенности
                A[pos.prev, 2] = A[pos.prev, 2] - 1
        }
        }
}
return (n.true / n.all)
}
```

Нейронная сеть с методом обратного распространения ошибки:

```
def one_level_one_step_NN(tmp, alpha, add_neurons, cv_rate):
        n = len(tmp)

        max_pow = math.ceil(math.log(max(tmp), 2))
        #Преобразование в двоичный код
        binary_code = [[int(x) for x in
        list('{0:0b}'.format(y).zfill(max_pow))] for y in tmp]

        X = np.array(binary_code[1:(n - 1)])
        y = np.array(binary_code[2:n])

        #Разбиваем множество на тренировочное и тестовое
        n_train = round(n * cv_rate)

        X_train = X[1:n_train]
        y_train = y[1:n_train]

        X_test = X[(n_train + 1):]
```

```
y_test = y[(n_train + 1):]

n_input = max_pow
n_output = max_pow
#Инициализация весов
weight_1 = 2 * np.random.random((n_input,
        n_input + add_neurons)) - 1
weight_2 = 2 * np.random.random((n_input +
        add_neurons, n_output)) - 1

#Обучение модели
for i in range(10000):
        y1 = 1/(1 + np.exp(-(np.dot(X_train, weight_1))))
        y2 = 1/(1 + np.exp(-(np.dot(y1, weight_2))))

        y2_delta = (y_train - y2) * (y2 * (1 - y2))
        y1_delta = y2_delta.dot(weight_2.T)
        * (y1 * (1 - y1))

        weight_2 += alpha * y1.T.dot(y2_delta)
        weight_1 += alpha * X_train.T.dot(y1_delta)

n_test = len(X_test)

true_ans = 0

#Оцениваем модель
for i in range(n_test):
        val = 1 / ( 1 + np.exp(-np.dot(X_test[i],
                 weight_1)))
        tmp = 1 / ( 1 + np.exp(-np.dot(val, w2)))
        for j in range(max_pow):
                if (tmp[j] < 0.5):
                        tmp[j] = 0
                else:
```

```
                        tmp[j] = 1


if (all(tmp == y_test[i])):
        true_ans += 1

return (true_ans / n_test)
```