Проект:

Определение намерений пользователя по сессии на сайте: покупка, просмотр товаров, изучение категорий и т.п.

Студент: Антон Саттаров
Ментор: Турал Гурбанов

Проблема:

В современных рекомендательных системах зачастую используется совокупность рекомендательных алгоритмов - гибридные рекомендательные системы, для улучшения качества рекомендаций. Такой подход обеспечивает большую вариативность рекомендаций, так как разные алгоритмы подходят к разному пользовательскому поведению.

Обратной стороной данного подхода, является увеличение накладных расходов на вычисления, а также большое количество получаемых рекомендаций. Увеличение накладных расходов на вычисления приводит к увеличению затрат на компьютерные мощности или к увеличению длительности вычислений, это в свою очередь приводит к тому, что пользователь может получать полезные ему рекомендации слишком поздно, а большое количество получаемых рекомендаций перегружаю интерфейс системы и может отвлекать пользователя от его первоначальных целей.

Оба этих фактора снижают доверие пользователя к рекомендательной системе и портят пользовательский опыт от взаимодействия с сервисом.

Пути решения:

Для устранения данных недостатков предлагается построить модель определяющую намерения пользователей по их поведению и исходя из результатов её работы оптимизировать гибридную рекомендательную систему. Это может помочь улучшить качество рекомендаций, снизив вес рекомендациям неэффективных алгоритмов для конкретного пользователя, а также снизить накладные расходы на вычисления, не производя расчётов по неэффективным алгоритмам(при этом конечно нужно учитывать, что система по определению намерения тоже имеет свои накладные расходы).

Цель проекта:

На примере дата сета содержащего *log*-и пользовательских сессий исследовать возможность определения пользовательских намерений по последовательности действий. Под намерением пользователя понимается цель с которой он воспользовался сервисом.

Например, совершить покупку товара, найти и узнать информацию о товаре(наличие, цена, технические характеристики), сравнить несколько товаров и так далее. Под намерения понимается некоторая обобщённая последовательность событий в *log*-ах характеризующая определённые действия пользователя сервиса.

Предполагается, что пользователи имеют ограниченно количество намерений и их можно определить исходя из содержимого *log*-ов и что последовательности событий совершенные пользователями с похожими целями будут так же похожи, то есть их могут быть выделены из данных и обобщены.

Для достижения данной цели предлагается:

1. Провести анализ данных, выделив последовательности действий пользователей.
2. Отобрать признаки и построить модель способную в online режиме по последовательно поступающим данным о взаимодействии пользователя с сервисом предсказывать намерение пользователя.
3. Подобрать метрики и оценить качество по отложенной выборке.

Похожие проекты:

Статей по данной теме не много, но они есть. Это говорит о том, что проблема существует и актуальна. Например, есть успешное исследование в e-commerce области: *Improving Recommender Systems with an Intention-based Algorithm Switching Strategy* [8]

В статье описывается решение похожей задачи - но в качестве данных для определения намерения пользователя используются данные о перемещениях мышки и пользователя по сайту.

Дата сет:

https://www.kaggle.com/retailrocket/ecommerce-dataset

Итак нам доступны следующие данные:

- *events.csv* - 4.5 месяца клик-стрима пользовательских сессий содержит информацию о времени события в ms, элементе взаимодействия и его типе { "*view*", "*addtocart*" и "*transaction*" }

index	timestamp	visitorid	event	itemid	transactionid
0	1430622004384	693516	addtocart	297662	-1
1	1430622011289	829044	view	60987	-1

- *item_properties_part1.csv* , *item_properties_part2.csv* описание объектов, содержит информацию о свойствах объекта(например описание, категория и так далее) и их изменения во времени с шагом в 1 неделю.

index	timestamp	itemid	property	value
0	1431226800000	317951	790	n32880.000
1	1431226800000	179597	available	0

  - свойство *categoryid* - категория объекта
  - свойство *available* - информация о наличии
  - остальные свойства и их значения хешированы

- *category_tree.csv* - дерево категорий объектов, содержит информацию о родительской категории объекта

categoryid	parentid
1016	213
809	169

Анализ данных:

Для начала разобьём наши дата сеты на 2 выборки - тренировочную и отложенную в пропорции **80/20** и так чтобы события в них не пересекались по времени. Таким образом в тренировочную выборке у нас будут все записи старше записей в отложенной.

```
https://github.com/dirtymew/netology-project/blob/master/Project-data-prepare.ipynb
```

Тренировочная выборки *events* и *item_properties* будут использоваться для всех последующих анализов и построения всех моделей, а на отложенной выборке events будет производиться финальная оценка качества, при этом выборка item_properties будет использоваться полная.

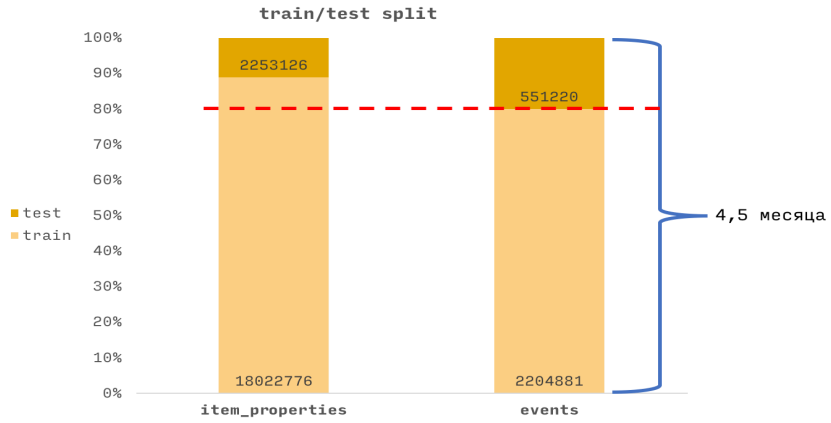

Рис.1 Train/test split

Задача на следующем этапе - понять каким образом по событиям различать поведение пользователей, и какое оно бывает. Для этого нужно внимательно посмотреть на данные.

Событий дата сете	2204881
Уникальных visitorid в дата сете	1123767

Количество событий по типу:

view	2132032
addtocart	54985
transaction	17864

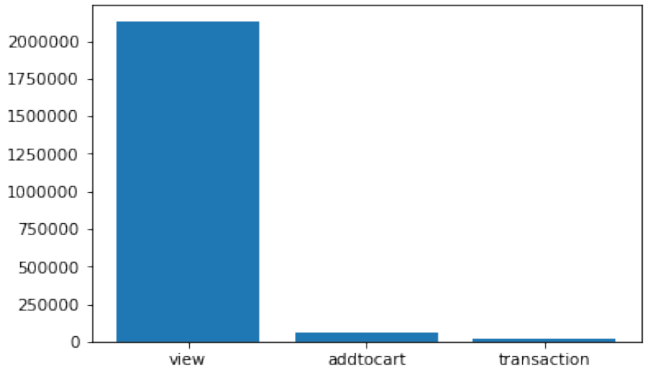

Рис.2 Гистограмма распределения событий по типу

Далее сгруппируем данные по *visitorid* и вычислим общую длительность взаимодействия:

Кол-во пользователей:	1123767
Кол-во пользователей у которых по 1-му событию:	801886
Кол-во пользователей у которых более 1-го события:	321881

Длительность взаимодействия (разница между первым и последним событием):

Минимальная длительность	мин: 0
Максимальная длительность	мин: 153809

Выводы:

- Большинство пользователей в данном дата сете имеют 1-но взаимодействие с серви сом - эти данные для обучения модели не имеют смысла, по ним не получиться опр еделить намерения пользователя.

- Существуют пользователи у которых длительная история взаимодействия - это нав одит на мысль, что такие пользователи использовали сервис несколько раз и возмож но с разными целями.

1. Анализ сессий:

https://github.com/dirtymew/netology-project/blob/master/Project-analys-sessions.ipynb

Определим интервалы между событиями пользователей и построим гистограмму количество интервалов между событиями пользователя относительно их длительности (в минутах).

Квантили распределения интервалы относительно длительности (в минутах):

0.25	0.601854
0.50	2.127133
0.75	31.553021

Постоим гистограмму интерквантильного размаха(дисперсии) данного распределения (0.75 квантиль)

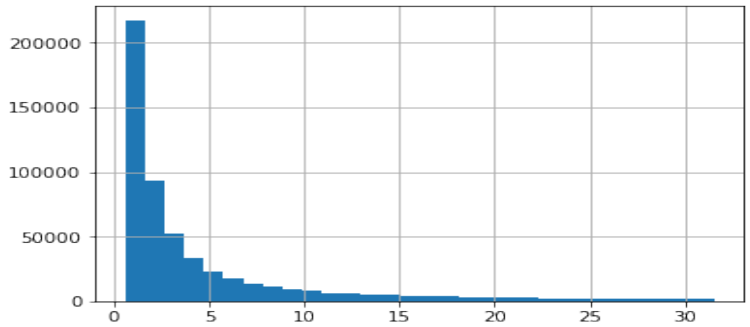

Рис.3 Гистограмма интерквантильного размаха 0.75 квантиль

Гипотеза №1 о сессий:
Пользовательская сессия - это некоторая конечная временная последовательность вз аимодействий. При этом конец сессии обусловлен завершением взаимодействия со сто роны пользователя на определённый период времени. Таким образом для нахождения п ользовательских сессий предполагается определить граничный интервал, при превыш ении которого с момента последнего взаимодействия будем считать, что сессия заве ршилась. И следующее взаимодействия пользователя будут началом новой сессии.

Верхнюю границу сессии определим как 3-й квантиль (**31.553021** минут)

Далее разобьём выборку на сессии:

Кол-во пользователей	321881
Кол-во сессий	592160

Теперь сгруппируем данные по сессия и вычислим общую длительность взаимодействия:

Сессий	92160
с 0-ой длительностью	285387
с не 0-ой длительностью	306773

Длительность сессии (сумма всех интервалов):

Минимальная длительность, мин	0.0
Максимальная длительность, мин	728.4

Оставим только сессий с вероятностью 99% в данном распределении, тогда гистограмма будет выглядеть следующим образом:

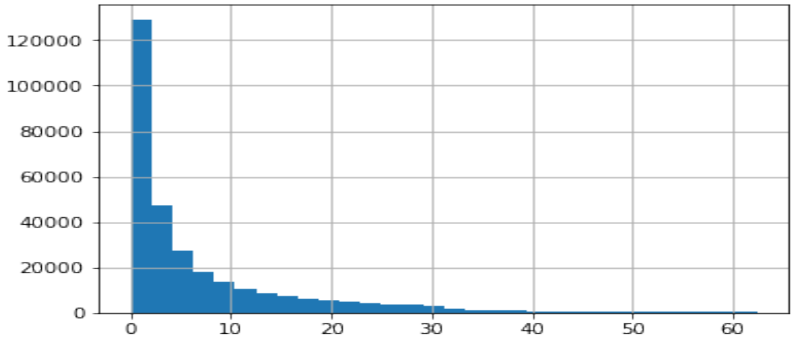

Рис.4 Гистограмма количество сессий пользователя (99%)

Длительность сессий (сумма всех интервалов):

Минимальная длительность мин:	0.0
Максимальная длительность мин:	62.45

Квантили данного распределения:

0.25	0.917700
0.50	2.906800
0.75	9.170033

Таким образом можем разделить сессии по длительности на:

Короткие	75928
Средние	151851
Длинные	75926

Как видно коротких сессий довольно много, при этом их максимальная длительность меньше 1 минуты. Возможно это куски других сессий и граница разбиения была выбрана неверно.

Событий в коротких сессиях	166861
Пользователей с короткими сессиями	71326
Общее количество пользователей	253761

Сгруппируем данные события по пользователям у которых есть короткие сессии, их количество - **51405**

Таким образом мы потенциально неверно определили сессий для **19921** пользователя.

Сгруппируем события по данным пользователям и по сессиям – **31696** сессий

Можно заменить, что короткие сессии бываю 2 видов, одни являются первой сессий пользователя и их максимальный интервал не будет больше **31.553021** - граничного интервала сессии, другие напротив являются потенциальным концом предыдущий сессий и их максимальный интервал будет больше граничного.

Количество начальных сессий	15462
Количество оконечных сессий	16234

Для анализа начальных сессий составим выборку из сессий идущих следующими. И сравним их максимальный интервал с максимальной длительностью в **62.45** минут.

Количество таких начальных сессий – **1743**

Оконечные сессии также сравним с максимальной длительностью **1631**

Таким образом ошибочных разбиений **~ 3 374**, что составляет **~ 1%** от общего количества сессий.

2. Анализ событий:

Исходя из этих данных можно определить намерения, то есть провести кластеризацию по признакам, но отсутствие ground-truth данных не позволит проверить качество этих решений.

По этой причине, было решено переключиться на задача классификации намерения пользователя, предполагая, что намерения заранее известны.

В нашем случае, в дата сете присутствует как минимум одно чёткое намерение - покупка. Далее будет рассматриваться задача классификации намерения по 2-м классам "Купить" и "Не купить"

Для классификации сессий нужно подобрать набор признаков.

Пользователь может взаимодействовать с :

- одним товаром
- с несколькими товарами:

- одной категории
- разных категорий

Категорию товара можно найти в *item_properties* ,а «близость» категорий можно определить как количество величину кратчайшего пути, если представить *category_tree* как граф.

Так мы можем представить 3 элементарных действия:

- Посмотреть - *view*
- Отложить - *addtocard*
- Купить – *transaction*

Намерение в общем случае не тоже самое что действие. Но в нашем случае ("Купить" и "Не купить") оно безусловно выражается в действии *transaction.* Таким образом если в сессии есть событие *transaction,* то по нашей гипотезе все предыдущие события должны выражать намерение “Купить". На данном допущении будет строится дальнейшие исследование.

Построение модели:

1. Классификация намерений по признакам:

```
https://github.com/dirtymew/DS1-netology/blob/master/HomeWork/Project/Project-transaction-classifier.ipynb
```

Для проверки гипотезы о намерении - предлагается построить классификатор отличающий намерение "Купить" от других по признаковому описанию сессии.

Составим набор признаков для классификации, предварительно очистив сессии с покупками от событий следующими за *transaction*.

Далее определим категорию объектов исходя из данных в *category_tree* и *item_properties.*

Очистим сессии от последнего события – сохранив его как целевое значение, а сгруппировав их по *id* сессии, получим следующие признаки для классификации:

```
count - количество событий
items_unique - количество уникальных обьектов
sessions_len - длительность сессий
ssid_type - тип сессии(короткая, средняя, длинная)
delta_mean - средние значение интервалов между событиями
categories - количество уникальных категорий обьектов
event_type_nu - категориальный признак 0-если в сессии один тип события, 1-если несколько
dist1_mean - среднее расстояний категорий обьектов в сессии
dist1_max - максимальное расстояний категорий обьектов
dist1_sum - сумма расстояний категорий обьектов
target - была ли покупка в сессии
```

Заметим что классы несбалансированные:

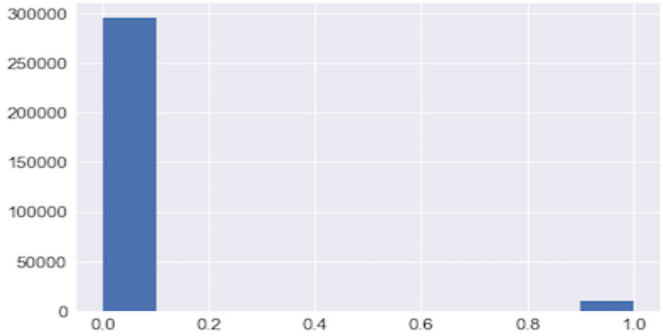

Рис.5 Гистограмма распределения классов

Необходимо сбалансировать выборку, иначе модель хорошо обучиться только на объектах одного класса.

Разделим выборку на *train/test* с сохранением баланса между классами.

Проведём *oversampling train* сета, а test оставим без изменений [2].

Стратегия балансировки - *Naive random over-sampling* - суть которой случайное копирование с перемещением строк в выборке.

Полученное распределение классов в обучающей выборке:

класс 0	236750
класс 1	236750

Моделью для классификации был выбран ансамбль алгоритмов *Random Forest* библиотеки *scikit-learn* с подбором параметров по сетке и оценкой качества по *'f1'*. [1]

Деревья решений просты в интерпретации и удобны в использовании. Это сильные алгоритмы, но склонные к переобучению.

Параметры подбираемые по сетке:

```
param_grid = {'n_estimators': [ 20, 50, 150, 500],'max_features':[ 3, 5, 7, 10]}
```

Кросс-валидация:

```
StratifiedKFold(n_splits=5, random_state=None, shuffle=True)

 (n_jobs=-1,oob_score=True,random_state = 42,verbose=0)
```

Параметры "лучшего классификатора":

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features=3, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=150, n_jobs=-1,
            oob_score=True, random_state=42, verbose=0, warm_start=False)
```

Метрики качества

Обучим модель и предскажем намерения для test сета и оценим качество по метрикам.

Рис.6 Матрица ошибок классификации

Доля правильных ответов алгоритма:

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

accuracy **- 0.967**

В нашем случае выборка несбалансированная и оценивать качество модели по accuracy не правильно.

Например если в предсказанной выборке все значения будут 0-класса то:

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN} = \frac{59188 + 0}{59188 + 0 + 0 + 2107} = 0{,}966$$

Поэтому ориентироваться нужно на точность

$$precision = \frac{TP}{TP + FP}$$

и полноту

$$recall = \frac{TP}{TP + FN}$$

Precision - это долю объектов определённых классификатором действительно являющимися объектами этого класса

Recall - долю объектов определённого классификатором класса из всех объектов класса определённого алгоритмом.

Ф-мера *f1*- это среднее гармоническое precision и recall

$$F_1 = (1 + b^2) * \frac{precision * recall}{b^2 * precision + recall}$$

где b=1

Precision, *Recall*, *F1* не зависят от баланса классов.

Для оценки качества модели была выбрана *f1*, так как неясно что важнее для данной задачи точность или полнота.

Ниже приведены результаты классификации:[2]

	Precision	recall	f1-score	support
class 0	0.98	0.98	0.98	59188
class 1	0.52	0.54	0.53	2107
avg / total	0.97	0.97	0.97	61295

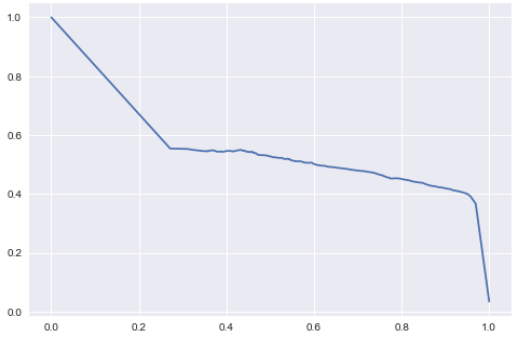

Рис.7 precision – recall кривая

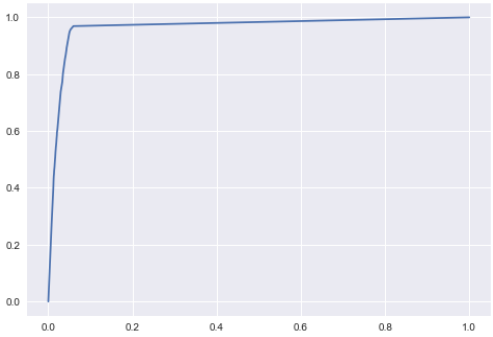

Рис.8 ROC кривая

Видно что намерение "Купить" можно выделить.

Определяющими факторами являются:

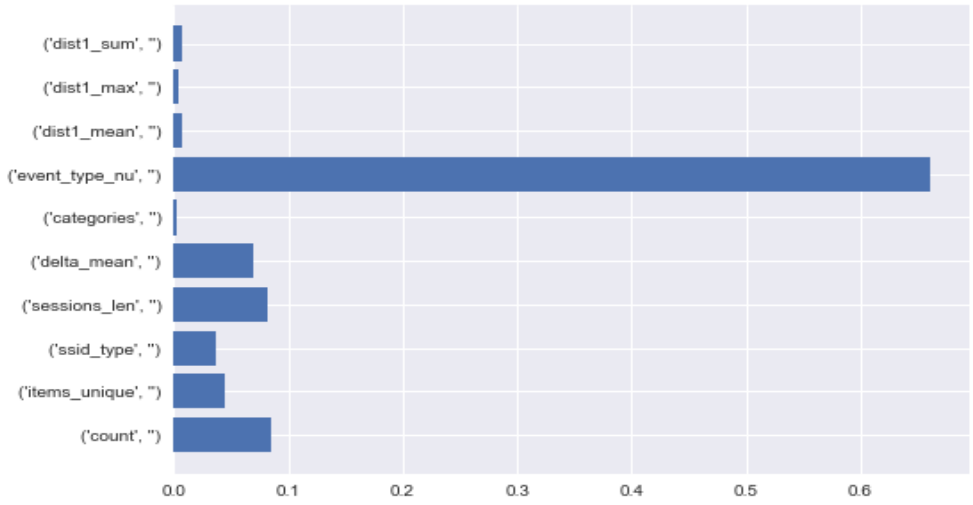

Рис.9 Важность признаков

Данный факт наводит на мысль, что анализ последовательности действий event над товарами *itemid* должен дать хорошие результаты.

2. Классификация намерений по последовательности действий :

https://github.com/dirtymew/DS1-netology/blob/master/HomeWork/Project/Project-transaction-classifier.ipynb

Гипотеза №2 о намерении:
Предположим что намерение пользователя отражаются в его действиях,то в нашем случае они должны выражаться в последовательности элементарных действий(actions) с товарами(items) совершенных пользователем(visitorid).

Для анализа последовательности существует несколько алгоритмов - основанные на Марковских цепях или на рекуррентных нейронный сетях (*RNN*). Особенность этих методов в том, что они учитываю зависимость следующего состояния от предыдущего.[9]

Для данной задачи был выбран вариант использования нейронной сети, так как считается что, они способны находить более глубокие зависимости в последовательностях чем Марковские цепи. [9]Но у него есть и свои недостатки - результаты работы сложно объяснить в отличии от Марковских цепей, а также высокие накладные расходы на обучение модели.[9]

В качестве рекуррентных слоёв сети стоит использовать слои *LSTM* или *GRU*, так как они обладают свойством долгосрочной памяти, а проблемы долговременных зависимостей и *vanishing/exploiding gradient* проявляются в меньшей степени, чем в классической *RNN*.[3][9]

Для предсказания последующих действий будем использовать следующее представление сессии:

$$(event_i, item_i) -> (event_{i+1}, item_{i+1}) -> \cdots -> (event_{n-1}, item_{n-1}) -> (event_n, item_n)$$

где event_i - event , item_i - это itemid

Последовательность:

$$(event_i, item_i) -> (event_{i+1}, item_{i+1}) -> \cdots -> (event_{n-1}, item_{n-1})$$

будет использоваться как данные - X

$(event_n)$ - как целевое значение.

Сгруппируем данные в сессии, таким образом чтобы получить 2 массива actions и items, и заменим строковые значение action {*view*, *addtocard*, *transaction*} на {0,1,2} , *itemid* оставим без изменений.

Также как и при классификации моделью *Random Forest*, перед обучение необходимо сбалансировать выборку, *Oversampling* будем проводить по той же стратегии[2]

Реализации *RNN* в *Keras* не поддерживают динамическую длину последовательности, соответственно необходимо выбрать максимальную длину и привести все сессии к ней. [7]

В дата сете есть очень длинных последовательностей - для ускорения процесса обучения стоит или их удалить из выборки или удалить часть событий с начала последовательности.

Максимальную длину последовательности определим следующим образом – определим 0.99 квантиль длин сессий. Для обучающей выборки – это 99% сессий с покупкой имею меньше **36** событий.

У сессий с длинной больше 36, удалим часть событий с начала сессии.

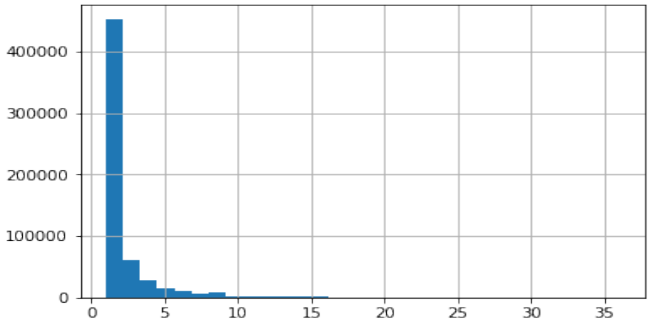

Рис.10 Гистограмма распределения количества событий в сессии

Модель будет обучаться на $(n-1)$ элементах массива и предсказывать следующие действие(либо класс намерения).

Тут стоит отметить, что если использовать action_n как целевую переменную, то по факту алгоритм будет обучаться на предсказание следующего действия, в нашем случае можно сразу предсказывать намерение и отнести сессия окончившиеся на 0, 1 к одному классу.

Архитектура сети:

Входными параметрами сети являются 2 массива вида:

actions = [0,1…1,0…]

items=[422417,422417,…,220109..]

Далее следует преобразование элементов обоих массивов - *Embedding* . На этом слое происходит преобразование последовательности *actions* и *items* в последовательность вектор латентных признаков.

Размерность векторов подбирается вручную. [4,5,6]

В результате по сути получаются 2 матрицы размерностью:

количество типов событий/объектов X максимальная длинны последовательности

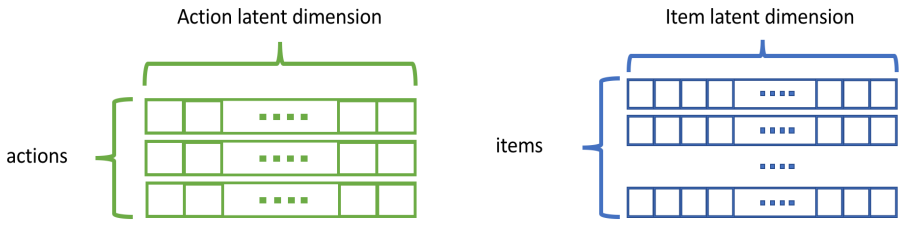

Рис.11 Embedding действия и объектов.

Далее производится с полученных векторов: [4,5,6]

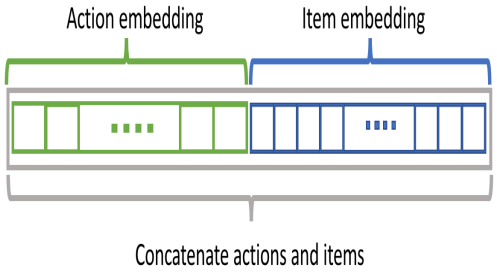

Рис.12 Конкатенация векторов действия и объектов.

и результат отправляется в рекуррентные слои.

Будем использовать 2 слоя *GRU* с регуляризацией рекуррентных слоёв *l2* и *dropout*.[4,5]

Параметры регуляризации и dropout подбираются вручную.

Затем идут полносвязные слои, их может быть несколько, но выход последнего слоя имеет размерность соответствующую количеству предсказываемых классов, с функцией активации *softmax*.[3]

Функция потерь - *'categorial_crossentropy'* [5,3]

Функция оптимизации - *'adam'* [5,7]

Метрика качества - *'categorical_accuracy'* [5]

Полная схема сети:

```
Layer (type)                 Output Shape              Param #      Connected to
==================================================================================
actions (InputLayer)         (None, 60)                0
_____________________________________________________________________________
items (InputLayer)           (None, 60)                0
_____________________________________________________________________________
embedding_1 (Embedding)      (None, 60, 100)           300        actions[0][0]
_____________________________________________________________________________
embedding_2 (Embedding)      (None, 60, 100)           14718400    items[0][0]
_____________________________________________________________________________
concatenate_1 (Concatenate)  (None, 60, 200)           0         embedding_1[0][0]
                                                                 embedding_2[0][0]
_____________________________________________________________________________
gru_1 (GRU)                  (None, 60, 200)           240600 concatenate_1[0][0]
_____________________________________________________________________________
gru_2 (GRU)                  (None, 200)               240600      gru_1[0][0]
_____________________________________________________________________________
dense_1 (Dense)              (None, 4)                 804         gru_2[0][0]
_____________________________________________________________________________
dense_2 (Dense)              (None, 2)                 10          dense_1[0][0]
_____________________________________________________________________________
activation_1 (Activation)    (None, 2)                 0           dense_2[0][0]
==================================================================================
Total params: 15,200,714
Trainable params: 15,200,714
Non-trainable params: 0
```

Обучение модели:

Нужно определить количество эпох обучения и *batch_size* и можно обучать модель. Каждая эпоха занимает ~ **400** секунд - с вычислениями на *GPU GeForce GTX 1060 6GB*

Так как на выходе получаются вероятности, то для определения классов из вероятностей необходимо использовать функцию *argmax*.

Оценим качество предсказаний:

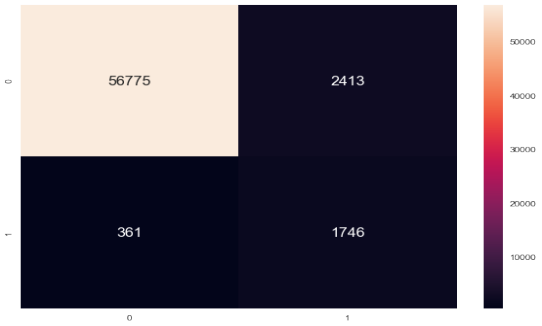

Рис.13 Матрица ошибок классификации

Отчет по классификации

	precision	recall	f1-score	support
0	0.99	0.96	0.98	59188
1	0.42	0.83	0.56	2107
avg / total	0.97	0.95	0.96	61295

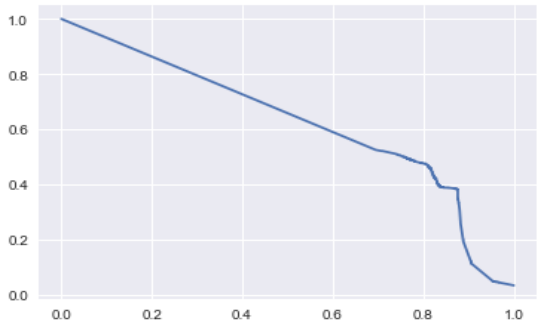

Рис.14 precision – recall *кривая*

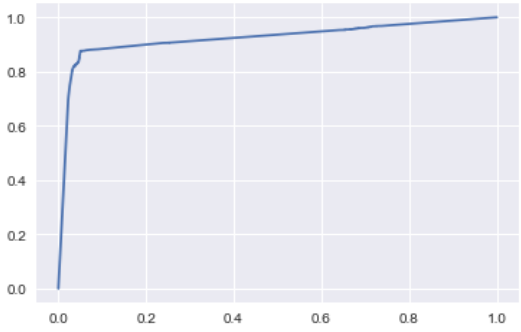

Рис.15 ROC кривая

Качество сравнимо с *Random Forest: - F1* примерно равно, точность ниже, но выше полнота.

Результаты

`https://github.com/dirtymew/DS1-netology/blob/master/HomeWork/Project/Project-transaction-classifier.ipynb`

Для предсказания намерения на отложенной выборке сначала необходимо подготовить данные : разделить на сессии, выделить последовательности [(*action*,*items*)], удалить события после покупки и привести последовательности у одной длине. Предварительно стоит доучить сеть на данных использовавшихся для тестирования.

Ниже приведены результаты:

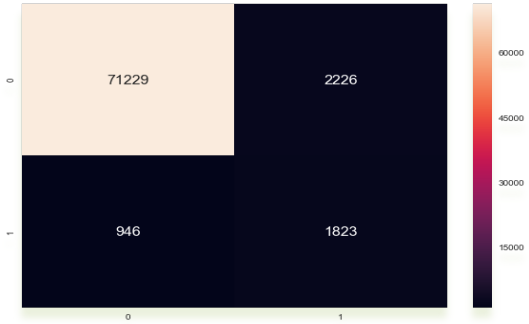

Рис.16 Матрица ошибок классификации

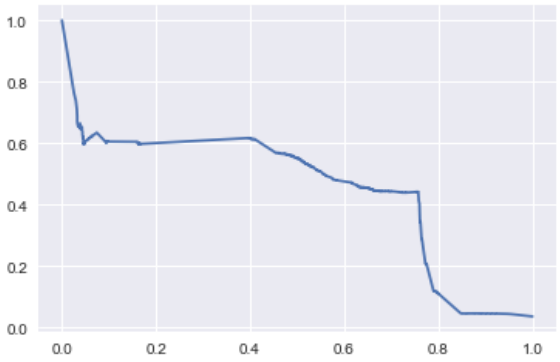

Рис.17 precision – recall *кривая*

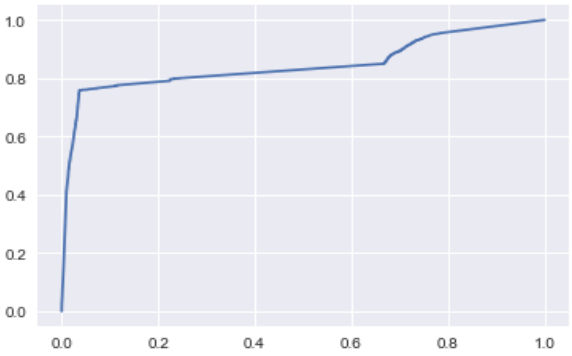

Рис.18 ROC кривая

	precision	recall	f1-score	support
0	0.99	0.97	0.98	73455
1	0.45	0.66	0.53	2769
avg / total	0.97	0.96	0.96	76224

Качество на отложенной выборке немного ниже чем при обучении.

Выводы

Обе выбранных модели показываются сравнимое качество работы. Но не высокое качество предсказаний возможно обусловлено слабыми зависимостями между выделенными признаками и предсказываемым намерением.

Если сравнивать модели то, к достоинствам *Random Forest* можно отнести:

- Накладные расходы на вычисления ниже чем у *RNN*
- Простота в реализации и подборе параметров
- Объяснимость результатов

Недостатки следующие:

- Сложность в подготовке данных и больше признаков чем у *RNN*
- Не учитывает последовательности
- Сложнее доучивать модель на новых данных

В защиту *RNN* также можно отнести относительно малое количество данных для обучения в данном дата сете.

На мой взгляд решение задачи методом анализа последовательности более перспективно. Модели на основе рекуррентных сетей хорошо для этого подходят.

Дальнейшие пути улучшения:

Попробовать различные архитектуры рекуррентных сетей в комбинации с различными параметрами и модели на основе скрытых Марковский цепей(*HMM*).

Попробовать добавить больше признаков для обучения модели - например добавить *Embedding* для пользователя - это даст сети историю пользователя.

Используемые материалы

[1] - *Documentation of scikit-learn* (http://scikit-learn.org/stable/documentation.html)

[2] - *Data Mining for Imbalanced Datasets: An Overview.* (2010) *Chawla, Nitesh V.* (https://www3.nd.edu/~dial/publications/chawla2005data.pdf)

[3] - *Deep Learning.* (2016) *Ian Goodfellow, Yoshua Bengio, Aaron Courville* (http://www.deeplearningbook.org/contents/rnn.html)

[4] - *Tensorflow Documentation* (https://www.tensorflow.org/api_docs/python/)

[5] - *Keras Documentation* (https://keras.io)

[6] - *Deep matrix factorization using Apache MXNet. (2017) Jacob Schreiber* (https://www.oreilly.com/ideas/deep-matrix-factorization-using-apache-mxnet)

[7] - *An overview of gradient descent optimization algorithms. Sebastian Ruder(2017) Sebastian Ruder* (https://arxiv.org/pdf/1609.04747.pdf)

[8] - I*mproving Recommender Systems with an Intention-based Algorithm Switching Strategy. (2017) Fanjuan Shi , Chirine Ghedira* (https://www.researchgate.net/publication/316035689_Improving_Recommender_Systems_with_an_Intention-based_Algorithm_Switching_Strategy)

[9] - *A Critical Review of Recurrent Neural Networks for Sequence Learning (2015) Zachary C. Lipton , John Berkowitz* (https://arxiv.org/pdf/1506.00019.pdf)