КУРСОВ ПРОЕКТ

Препоръчваща система за състезанието Dressipi Recsys2022

|  |
| --- |
| **Факултет по Математика и Информатика** **Студент:** Борислав Стоянов Марков **Факултетен номер:** **0MI3400048** **Учебен план:** **Изкуствен Интелект (редовно, магистър)** Курс: **Курс 1**; Група: **Група 1** **Активен период**: 2021/2022 летен, магистри **Дисциплина**: Препоръчващи системи |

# 1. Съдържание

[1. Съдържание 2](#_Toc107221509)

[2. Увод 2](#_Toc107221510)

[2.1 За състезанието Dressipi 2022 3](#_Toc107221511)

[3 Корпус с данни 4](#_Toc107221512)

[4. Анализ на данните 8](#_Toc107221513)

[4.1 Отправна точка(baseline) 10](#_Toc107221514)

[5. Реализация с LightFM 10](#_Toc107221515)

[5.1 Алгоритъм 11](#_Toc107221516)

[5.1.1 Предаване на резултата към Дъска на Лидерите 15](#_Toc107221517)

[5.2 Използване на приложението 16](#_Toc107221518)

[5.3 Оценка на резултатите 17](#_Toc107221519)

[8. Недостатъци и подобрения 18](#_Toc107221520)

[9. Източници и използвана литература 18](#_Toc107221521)

[Приложения 18](#_Toc107221522)

[1. Сорс код (Source code) 18](#_Toc107221523)

# 2. Увод

Проектът има за цел да се реализират получените от курса по препоръчващи системи знания. Dressipi е състезание от експерти по изкуствен интелект с участието на ETH университета в Цюрих. Ние като студенти имаме възможността да сравним резултатите си със световни лидери по темата като тези резултати ще бъдат показани публично.

## 2.1 За състезанието Dressipi 2022

Дрессипи(Dressipi) е организация от експерти в областта на изкуствения интелект, която продава софтуер на онлайн магазини(ритейл мрежа). Те вече имат необходимия софтуер и препоръчваща система. Целта им е да подобрят софтуера, като организират подобно състезание всяка година.

Таз-годишното състезание е да се подобри препоръчващия механизъм за модни онлайн сайтове. Когато даден потребител разглежда няколко стоки, алгоритъмът на участниците трябва да предвиди акуратно какво е купил този човек накрая на потребителската си сесия. Потребителите са анонимизирани и са представени само като сесии от по максиум един ден. Съдържанието на данните са проверени от модни експерти и данните са от високо качество. На фигура 2.1.1 виждаме примерна сесия на потребител, който е разгледал няколко рокли и е купил подобна рокля.

|  |
| --- |
| Example Session and Purchase Data |

Фигура 2.1.1

Особености на задачата са:

* потребителите са 51% нови и няма исторически данни и предложените стоки трябва да се основават само на текущата сесия
* за тези потребители, за които има исторически данни, знаем, че си сменят вкуса и предпочитанията много рязко и това се обяснява със спецификата на домейна – модата.
* Сесиите са твърде кратки и трябват акуратни предложения за потребителя, преди да е напуснал сайта

Като част от задачата Дрессипи са реализирали набор от данни от 1.1 милиона ритейл сесии, които са приключили с поръчка. Всички стоки са етикирани по категории. Данните са стоките ще наричаме фийчъри на стоките. Всички данни са анонимизирани.

* Сесия: стоките, които са прегледани от потребител, максимално за един ден.
* Поръчка: Поръчка, която е станала накрая на сесията, максимално е дадена една стока за поръчка.
* Фийчър на стоката: Етикета на някой атрибут на стоката, например : „цвят:зелен“, „деколте: V-образно деколте“ и т.н.

Примерна атрибутирана стока е показана в фигура 2.1.2

|  |
| --- |
| Example Content Data |

Фигура 2.1.1

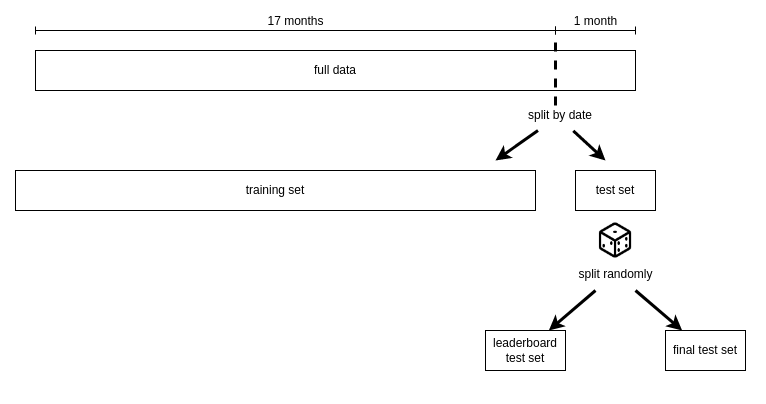
Повече информация за състезанието и правилата можете да намерите в [1]. Сега да се спрем на данните и тяхното подробно описание.

# 3 Корпус с данни

Описание на данните може е описано на сайта на Dressipi [2]. Данните са събрани от 1.1 милиона сесии от ритейл магазини за период от 18 месеца. Всички те са сесии в които има поръчка накрая. Артикулите, които се разглеждат, са облекло и обувки. Целта е да се предвиди какво е купено накрая на сесията. Най-общо можем да кажем какво означават сесия, поръчка и атрибут на стоката:

* Сесия: стоките, които са прегледани от потребител, максимално за един ден.
* Поръчка: Поръчка, която е станала накрая на сесията, максимално е дадена една стока за поръчка.
* Атрибут(фийчър) на стоката: етикета на някой атрибут на стоката, например : „цвят:зелен“, „деколте: V-образно деколте“ и т.н.

Разделянето на тестово и обучаемо множество става като тестовото множесто е от 17 месеца и данните от последният месец се ползва за тестово множество, като се разделя още на две: стена на лидерите, и финални данни. Задачата е да се направят 100 предположения за всяка сесия. Тренировъчните сесии са 1 милион а тестовите „стена на лидерите“ и „финално множество“ са по 50 хиляди сесии.



Фигура 3.1

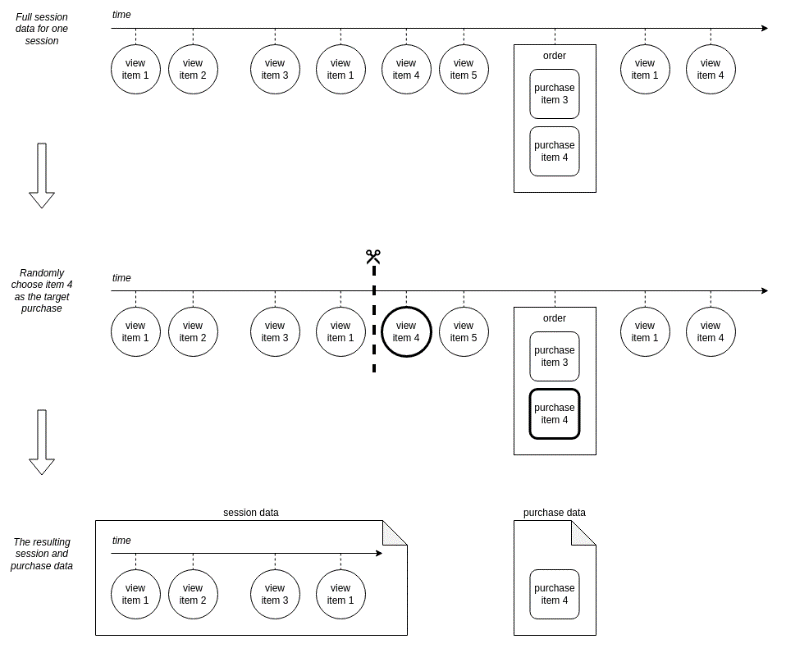
Подаването на решение се изисква във формат описан в таблица 3.2

|  |
| --- |
| session\_id,item\_id,rank  1,100,1  1,105,2  1,107,3  ...  1,101,100  2,108,1  2,107,2  ... |

Таблица 3.2

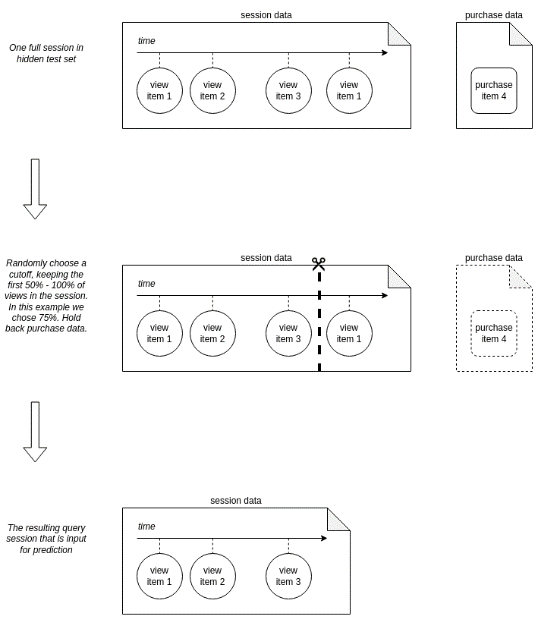
Метриката за оценяване е MRR (среден реципрочен ранк) Повече може да се прочете в [3].

Характеристиките на данните можем да кажем с няколко изречения. Данните са анонимизирани. При покупка на няколко стоки се взема предвид само първата. Много от сесиите имат само по едно две преглеждания преди покупка, но това е поради реалността и поради особеността как е конструирано множеството от данни. А именно при поръчка от няколко неща и продължаване на разглеждане, бъдещите интеракции не влизат в множеството за състезанието. А също и ако има разглеждане на артикула, който впоследствие е купен, той не влиза в данните за обучение. На следващата фигура е показана схема как се получават данните за сесиите и данните за покупките.



Фигура 3.3

Конструирането на данните за тестовите сесии се събира по следния начин. За всяка сесия се определя случаен процент х% между 50% и 100% и сесията се отрязва до там. Нарочно не се включва първият преглед на артикула, който е закупен накрая. Смисълът на това е да се предложи на потребителя стоки, които той би си купил и това предложение да е на по-ранен етап. Описаният алгоритъм за получаване на тестови сесии е показан на фигура 3.4.



Фигура 3.4

Сваляне на архива с данните за състезанието става само след регистрация. В архива, който се сваля има csv файлове и те изглеждат по този начин:

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |

Таблица 3.5

# 4. Анализ на данните

Анализът е направен с помощта на python и Jupyter Notebook средата. Можем един по един да покажем какво представляват CSV файловете, след като ги заредим в pandas.DataFrame, използвайки функцията “df.info()”. Ще покажем само само за по-важните „train\_purchases.csv” и “ train\_sessions.csv” в Таблица 4.1

|  |
| --- |
| <class 'pandas.core.frame.DataFrame'>  RangeIndex: 1000000 entries, 0 to 999999  Data columns (total 3 columns):  # Column Non-Null Count Dtype  --- ------ -------------- -----  0 session\_id 1000000 non-null int64  1 item\_id 1000000 non-null int64  2 date 1000000 non-null datetime64[ns]  dtypes: datetime64[ns](1), int64(2)  memory usage: 22.9 MB  train\_purchases\_df.info() : None |
| <class 'pandas.core.frame.DataFrame'>  RangeIndex: 4743820 entries, 0 to 4743819  Data columns (total 3 columns):  # Column Dtype  --- ------ -----  0 session\_id int64  1 item\_id int64  2 date datetime64[ns]  dtypes: datetime64[ns](1), int64(2)  memory usage: 108.6 MB  train\_sessions\_df.info() : None |

Таблица 4.1

Сега следва да видим как са разпределени прегледите по сесии и преглежданията по артикули – таблица 4.2.

|  |  |
| --- | --- |
| а) интеракции по сесии | б) брой прегледи по артикул |

Таблица 4.2

Средно имаме 4.74 интеракции във всяка сесия. От подточка а) виждаме, че най-много сесии има с по един преглед. От подточка б) Виждаме, че имаме някои много популярни артикули, докато други са преглеждани минимално и не са търсени. Можем да видим и как са прегледите по артикули:

|  |
| --- |
| train\_sessions\_df['item\_id']**.**value\_counts()**.**describe()  count 23496.000000  mean 201.899047  std 362.441815  min 1.000000  25% 14.000000  50% 95.000000  75% 263.000000  max 14714.000000  Name: item\_id, dtype: float64 |

Таблица 4.3

Прегледите по месеци и покупките по месеци са показани в таблица 4.4.

|  |  |
| --- | --- |
| *# session views per month*  train\_sessions\_df**.**groupby([train\_sessions\_df["date"]**.**dt**.**year, train\_sessions\_df["date"]**.**dt**.**month])**.**count()**.**plot(kind**=**"bar") | *# Purchases per month*  train\_purchases\_df**.**groupby([train\_purchases\_df["date"]**.**dt**.**year, train\_purchases\_df["date"]**.**dt**.**month])**.**count()**.**plot(kind**=**"bar") |
|  |  |

Таблица 4.4

Прегледите по година и час се виждат на следващата таблица. Силните часове за 2020г и 2021г са 19,20 и 21 часа. Часовете между 9 и 17 изглеждат еднакво натоварени. Останлите часове са слаби на преглеждания.

|  |
| --- |
| *# session views per year and hour*  train\_sessions\_df**.**groupby([train\_sessions\_df["date"]**.**dt**.**year, train\_sessions\_df["date"]**.**dt**.**hour])**.**count()**.**plot(kind**=**"bar") |
|  |

Таблица 4.5

Анализът на данни може да се види от „notebooks/EDA.ipynb“ в Приложения.

## 4.1 Отправна точка(baseline)

Като отправна точка за всеки алгоритъм е да се сравни с някой друг и това става като например даваме случайни предложения и измерим MRR (среден реципрочен ранк). За целта е разработена специална ноутбук папка „notebooks/Baseline.ipynb“ и там е реализиран случаен алгоритъм и замерване на 5000(поради ограничената RAM памет) елемента на случайно подбрана 5% извадка от поръчките. Вижда се, че MRR граничи около 0.01, което е и минимума, който е описан в състезанието.

Направен е и втори алгоритъм за базов, който ни препоръчва 100-те най-купувани стоки подредени в съответния ред. Това ни дава резултат **MRR=0.027**, което ще ни служи за отправна точка. Това е направено в „notebooks/Baseline\_top100.ipynb“

# 5. Реализация с LightFM

LightFM е библиотека на python за хибридни препоръчващи системи. Тя работи добре при студен старт, както имаме в текущите данни, тъй като нямаме информация за потребителите и техните вкусове. Ноутбук папката с разработката е „notebooks/lightfm.ipynb“ и там е кодът реализиращ препоръчваща система. Като цяло библиотеката работи с атрибути на артикулите(item\_features), атрибути на потребителите(user\_features) и потребителски взаимодействия(user\_interactions).

## 5.1 Алгоритъм

Тъй като нямаме потребители, то тогава идентификаторите на сесиите ще са потребителите. Разделяме множеството с поръчките на трениращо и тестово множество.

|  |
| --- |
| train\_p\_df, test\_p\_df = train\_test\_split(train\_purchases\_df, test\_size=0.02)  len(train\_p\_df)=980000 train purchases  len(test\_p\_df)=20000 test purchases |

Сесиите винаги са трениращо множество, защото там няма как да се тества. За атрибутиране на потребителите бихме могли да сложим месеца и часа на сесията, например “m6, h23”, или “m1, h11, h12”.

|  |
| --- |
| train\_sessions\_df['month']='m' + train\_sessions\_df['date'].dt.month.apply(str)  train\_sessions\_df['hour']='h' + train\_sessions\_df['date'].dt.hour.apply(str)  # Aggregate by sessionId and add hour and month as list of features  user\_features = train\_sessions\_df.groupby('session\_id').agg(  session\_id=pd.NamedAgg(column="session\_id", aggfunc="min"),  month=pd.NamedAgg(column="month", aggfunc=set),  hour=pd.NamedAgg(column="hour", aggfunc=set),  ).reset\_index(drop=True).apply( \  lambda row: (row['session\_id'], list(row['month']) + (list(row['hour']))), axis = 1)  user\_features.head(2) |

След направени експерименти се вижда, че слагане на фийчъри на потребителя **не носи добър резултат и те са премахнати**.

За стоките, имаме дадени категории и техните стойности, бихе могли да дадем атрибути на всяка стока нейните категории, конкатенирани със стойностите на категории. Например стока „3567“ ще има атрибути [“56-365”, “50-317”, “42-409”…] и т.н.

|  |
| --- |
| item\_features\_df['cat\_val'] = item\_features\_df['feature\_category\_id'].apply(str)+ '-' + item\_features\_df['feature\_value\_id'].apply(str)  # Aggregate by item\_id and add category and value as list of features  item\_features = item\_features\_df.groupby('item\_id').agg(  item\_id=pd.NamedAgg(column="item\_id", aggfunc="min"),  cat\_val=pd.NamedAgg(column="cat\_val", aggfunc=list),  ).reset\_index(drop=True).progress\_apply( \  lambda row: (row['item\_id'], most\_valuable(row['cat\_val'])), axis = 1)  item\_features.head(15) |

При направените експерименти се вижда, че **слагайки атрибути това не дава добър резултат и затова ги махаме от експеримента**. Направени са следните експерименти.

|  |  |  |  |
| --- | --- | --- | --- |
| Екс-пери-мент № | Фийчъри на стоката | Пример | Резултат MRR |
| 1 | (item\_id,[cat1-val1, cat2-val2..] | (2, [56-365, 62-801, 68-351, 33-802, 72-75, 29...  (3, [56-365, 69-592, 68-14, 17-378, 32-902, 11... | 0.03 |
| 2 | (item\_id,[cat1-val1, cat2-val2..]  Където cat-val са само измежду популярни категории | (2, [56-365, 62-801, 68-351, 33-802, 72-75, 29...  (3, [56-365, 69-592, 68-14, 17-378, 32-902, 11... | 0.03 |
| 3 | (item\_id,[cat1, cat2..]  Използвани са само ИД на категориите без стойностите  Където cat-val са само измежду популярни категории | (2, [56, 62, 68, 33, 72, 29...  (3, [56, 69, 68, 17, 32, 11... | 0.03 |
| 4 | (item\_id,[])  Използва се празен масив за фийчъри на стоката | (2, [])  (3, []) | 0.08 |

Таблица 5.1.1

Виждаме, че слагането на категории не носи резултат и затова всички фийчър вектори са премахнати (експеримент 4). Нямам обяснение за това, вероятно е замърсяване на модела с ненужна информация, защото при ръчно направен анализ се вижда, че измежду прегледаните стоки и закупената почти няма прилика, само една-две категории измежду 20 имат същата стойност.

LightFM изисква и масив на интеракциите. Ще получим този масив като използваме сесиите и поръчките и ги конкатенираме. Всяка интеракция трябва да получи тегло. Емпирично можем да кажем, че при преглед теглото ще е по-малко от 1, а при покупка ще е 1. Бихме могли да предположим, че потребителят се приближава все повече до желаната стока и следователно с всяко ново преглеждане можем да даваме повече точки за тази сесия и този артикул. Емпирично съм тествал няколко формули, които имат различен резултат:

|  |  |  |
| --- | --- | --- |
| Екс-пери-мент № | Формула | Резултат MRR |
| 1 | train\_sessions\_df['weight'] = train\_sessions\_df['score'].apply(lambda x: **np.tanh(x/4)**) | 0.04 |
| 2 | train\_sessions\_df['weight'] = train\_sessions\_df['score'].apply(lambda x: **np.tanh(x/7)**) | 0.0759 |
| 3 | train\_sessions\_df['weight'] **=** train\_sessions\_df**.**progress\_apply(**lambda** x: **x['view\_ord']/(x['views']+4)**, axis**=**1) | 0.0802 |

Таблица 5.1.2

При последният експеримент, вече не се ползва tanh а наслагването е линейно и пропорционално спрямо всички преглеждания за конкретната сесия.

Получените масиви се зареждат в LightFM модела и той се обучава.

|  |
| --- |
| model **=** LightFM(  no\_components**=**60,  learning\_rate**=**0.02,  loss**=**'warp',  random\_state**=**42)  model**.**fit(  trn\_interactions\_ds,  item\_features**=**item\_features\_ds,  user\_features**=**user\_features\_ds,  epochs**=**25, num\_threads**=**8, verbose**=True**)  (user\_map, feature\_map, item\_map, item\_feature\_map) **=** ds**.**mapping() |

Отново са направени различни експерименти с размера на компонентите „no\_components“.

|  |  |  |
| --- | --- | --- |
| Екс-пери-мент № | Компоненти | Резултат MRR |
| 1 | no\_components = 10 | 0.02 |
| 2 | no\_components = 30 | 0.06 |
| 3 | no\_components = 60 | 0.08 |
| 4 | no\_components = 70 | 0.07 |

Таблица 5.1.3

Виждаме, че моделът започва да деградира след 60, затова остава на тази стойност.

За изчисляване на резултатите избираме 2500 измежду най-продаваните стоки, тъй като ако сметнем всички възможни стоки в комбинация със всички възможни сесии, то не би ни стигнала RAM паметта, но това е само за тест, при предаване за Дъската на лидерите няма да има този проблем.

|  |
| --- |
| # Recommend from most sold items  items\_to\_predict\_array = \  list(train\_sessions\_df['item\_id'].value\_counts()[0:2500].keys()) |

Функцията predict\_ratings има за цел да получи готов DataFrame във вид за публикуване на резултатите. Реално се подготвят масиви с идентификатори на LightFM и после трябва да се обърнат във идентификатори, които ние ползваме.

|  |
| --- |
| **def predict\_ratings(model, sessions\_or\_purchase\_df, items\_to\_predict) :**  session\_ids = pd.Series(sessions\_or\_purchase\_df['session\_id'].unique())  all\_users = session\_ids \  .apply(lambda x: user\_map[x]).to\_numpy()  all\_available\_items = items\_to\_predict.apply(lambda x: item\_map[x]).to\_numpy()  users =[]  items =[]  for user\_item\_tuple in tqdm(product(all\_users, all\_available\_items)):  users.append(user\_item\_tuple[0])  items.append(user\_item\_tuple[1])  preds = model.predict(np.array(users), np.array(items))    session\_ids\_expanded = []  item\_ids\_expanded = []  for tup in tqdm(product(session\_ids, items\_to\_predict)):  session\_ids\_expanded.append(tup[0])  item\_ids\_expanded.append(tup[1])      df\_score = pd.DataFrame({'session\_id': np.array(session\_ids\_expanded), \  'item\_id':np.array(item\_ids\_expanded) , \  'score':np.array(preds)})  return df\_score |

Тук времеемките цикли са от декартовото произведение на **product(all\_users, all\_available\_items)** и подаването към LightFM модела.

Смятането на MRR е със следната функция, като по подразбиране ранка е 100 ако стоката не е в първите 100 предложения.

|  |
| --- |
| # Calculate Mrr (Mean reciprocal rank)  **def calc\_mrr(result\_df, test\_df):**  mrr = 0  # Iterate all sessions  for sess\_id in tqdm(test\_df['session\_id']):  # Make view for only this session with all ranked  ranked = result\_df[result\_df['session\_id']==sess\_id]['item\_id'].reset\_index(drop=True)  real\_item\_id = test\_df[test\_df['session\_id']==sess\_id]['item\_id'].reset\_index(drop=True)[0]  first\_rank = 100  found\_t = ranked[ranked == real\_item\_id]  if len(found\_t)!=0 :  first\_rank = found\_t.index[0]+1    mrr =mrr+ 1/first\_rank    mrr = mrr / test\_df['session\_id'].nunique()  return mrr |

След пресмятане на MRR над тестовото множество имаме резултат 0.0811.

Само за проба тестваме как се държи модела върху обучаваните данни и виждаме, че резултата над едно избрано множество от 10000 сесии от обучаваните данни ни дава резултат MRR=0.179, което е очаквано, тъй като това са виждани данни, но е едно потвърждение, че модела работи правилно, макар и не давайки добри резултати.

### 5.1.1 Предаване на резултата към Дъска на Лидерите

Предаването на резултат изисква да се обработи още един масив с интеракции „test\_leaderboard\_sessions.csv“ и да се премахне разделянето на обучаемо и тестово множество, защото всичко дадено е обучаемо множество и нямаме достъп до тестово множество. Всичко това е направено в отделен ноутбук „lightfm\_for\_leaderboard.ipynb“.

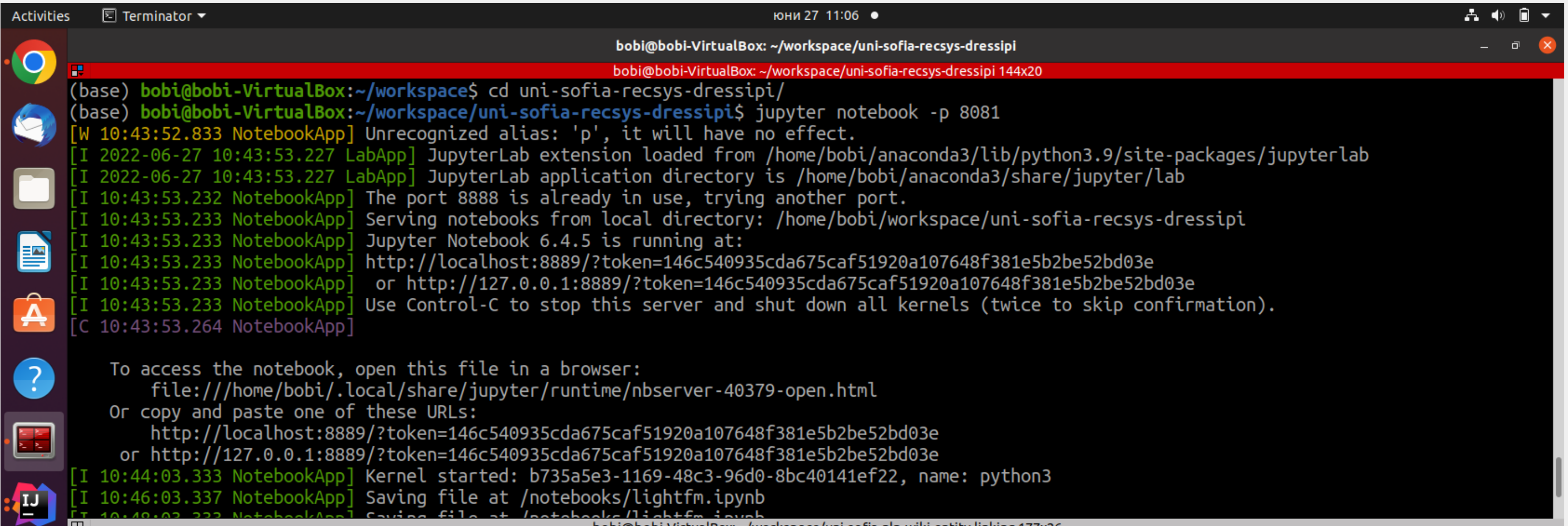
|  |
| --- |
|  |

Фигура 5.1.1.1

Виждаме, че след предаване на резултата има същият MRR , както и при тестовете, което е още една гаранция, че няма допуснати технически грешки, и че данните са наистина добре подбрани и истински.

## 5.2 Използване на приложението

Приложението се използва като локално на Линукс се инсталира jupyter notebook, който идва по подразбиране със средата Anaconda.



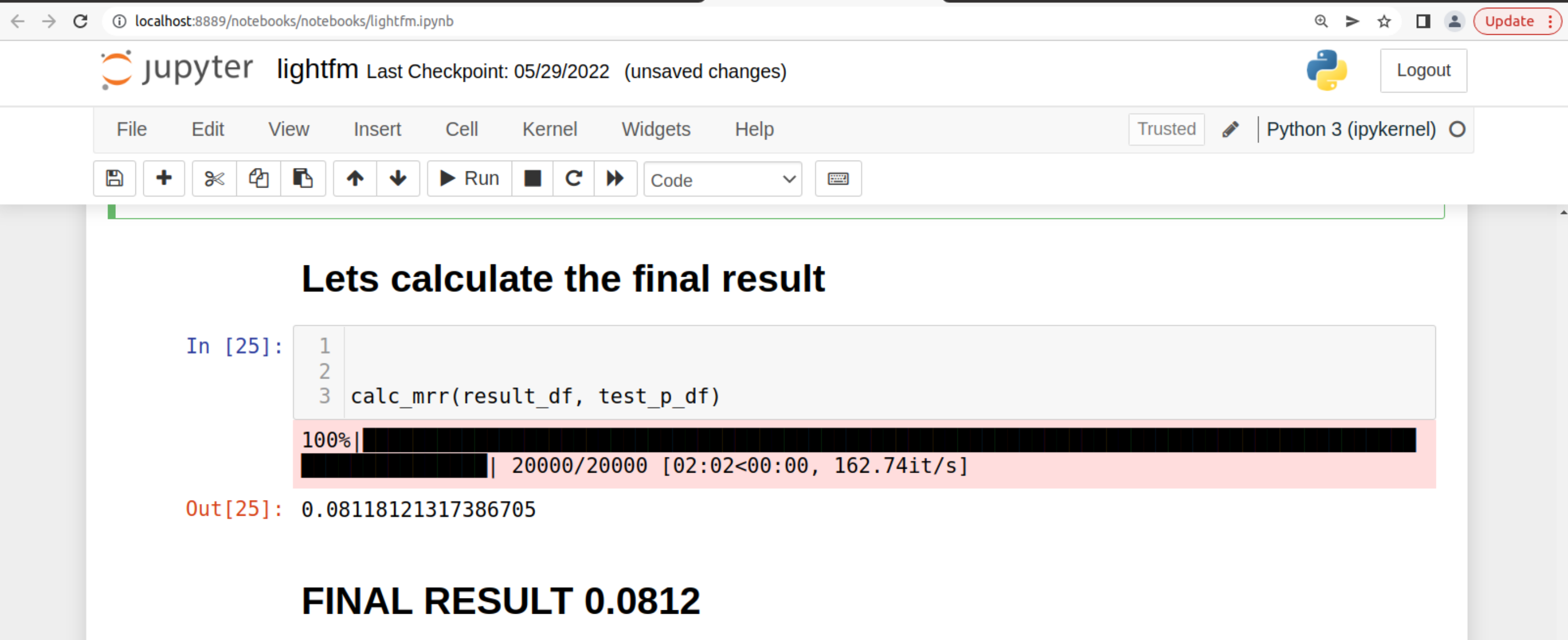
Фигура 5.2.1

След командата ‚jupyter notebook -p 8081’, което ни стартира сървър на порт 8081, отваряме браузър в текущата пака и можем да стартираме ноутбука.



Фигура 5.2.2

Отваряме например **lightfm.ipynb** и можем да стартираме всичко от началото, но след като свалим ZIP файла с данните, който не е публикуван в github поради лицензни съображения.



Фигура 5.2.3

## 5.3 Оценка на резултатите

Резултатите не са впечатляващи, защото **MRR=0.0812** не се класира след водещите резултати. Стигнахме до извода, че подаването към лидерборда също е с MRR=0.08, което ни дава увереност, че правилно сме предвидили какви точки ще имаме и данните са наистина добре подбрани от специалисти. Направихме оценка на тестовите данни и там имаме MRR=0.179, което означава, че правилно провеждаме експериментите, тъй като това са данните за трениране и е нормално да е повече MRR. Библиотеката LightFM не работи добре с добавени фийчър вектори на стоките и потребителите, което е озадачаващо, вероятно има някакъв проблем с имплементацията й.

# 8. Недостатъци и подобрения

С направените експерименти не можахме да постигнем добри резултати с LightFM. Може да се изпробва друга библиотека за подобен вид задачи. Като подобрения, можем да пробваме ансамблови алгоритми като Boosting или Stacking, което вероятно ще подобри резултата.

# 9. Източници и използвана литература

[1] The RecSys Challenge 2022, Dressipi, Bruce Ferwerda (Jönköping University, Sweden), Saikishore Kalloori (ETH Zürich, Switzerland), and Abhishek Srivastava (IIM Jammu, India).  
<http://www.recsyschallenge.com/2022/#participation>

[2] RecSys Challenge 2022 Dataset,   
<http://www.recsyschallenge.com/2022/dataset.html>

[3] Mean Reciprocal Rank, Wikipedia,  
<https://en.wikipedia.org/wiki/Mean_reciprocal_rank>

# Приложения

## 1. Сорс код (Source code)

Кодът е публичен и качен в платформата Гитхъб.

<https://github.com/borkox/uni-sofia-recsys-dressipi>