

Дипломный проект

по профессии Data Science

Разработка рекомендательной системы актуальных процедур закупок

Пояснительная записка

| Выполнил: | С. Федюшин |
| --- | --- |
| Руководитель: | А. Миронов |

2021

В данной работе выполнена разработка рекомендательной системы для рекомендации актуальных процедур закупок участникам на основе их истории участий в конкурсах. Исходными данными являются данные хакатона, проведенного компанией roseltorg в феврале 2021 года.

Ссылка на сайт хакатона <https://www.roseltorg.ru/zakupkihack>.

Описание исходных данных в ноутбуке 00\_Input\_data\_analyse. Разработка велась по следующим этапам:

1. Обработка текстовых описаний процедур (удаление не текстовой информации, удаление стоп-слов, лемматизация);
2. Векторизация текстовых описаний процедур (было реализовано 2 способа - первый с использованием TF-IDF с уменьшением размерности с использованием TruncatedSVD, второй с использованием модели Doc2vec из библиотеки gensim;
3. Обработка категориальных переменных методом OneHotEncoding, логарифмирование и нормализация значения цены;
4. Сборка датасетов, формирование матрицы взаимодействий, формирование матрицы признаков, обучение моделей LightFM, предсказание результатов из списка актуальных процедур, формирование выходных данных. (модели обучались на двух датасетах с разными способами векторизации текстовых описаний);
5. Подсчет метрик (точность и полнота) по методике организатора хакатона на основе датасета с истинными участиями;
6. Визуальная оценка результатов предсказаний моделей на основе сравнения текстовых описаний из истории участий с описаниями рекомендованных процедур.

Содержание

[1 Бизнес-анализ 4](#_gjdgxs)

[2 Анализ данных 5](#_30j0zll)

[3 Подготовка и генерация данных (feature engineering) 8](#_1fob9te)

[4 Моделирование 22](#_3znysh7)

[5 Оценка качества моделей 30](#_2et92p0)

[6 Оценка результатов 33](#_tyjcwt)

[Заключение 40](#_3dy6vkm)

# 1 Бизнес-анализ

1.1 Определение бизнес-целей

Создать рекомендательную систему для электронных торгов.

Проанализировать ретроспективные данные об участиях и победах поставщиков в торгах для разработки рекомендательной системы.

Создать работающий прототип рекомендательной системы, предлагающий поставщикам наиболее релевантные торги.

# 2 Анализ данных

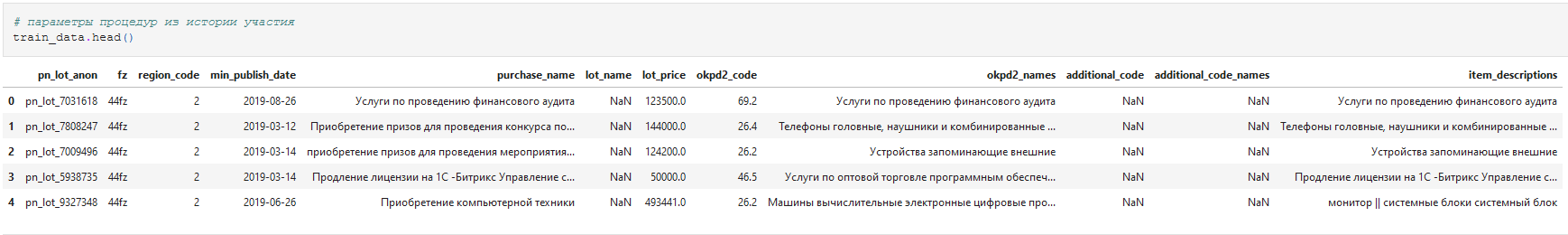
2.1 Входные данные в виде csv-файлов.

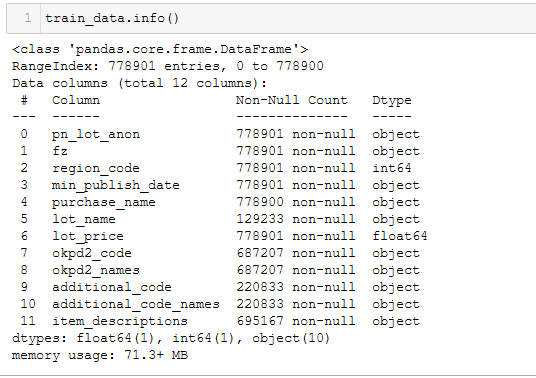
- train\_data.csv – уникальные процедуры за период обучения;

- train\_labels.csv – разметка;

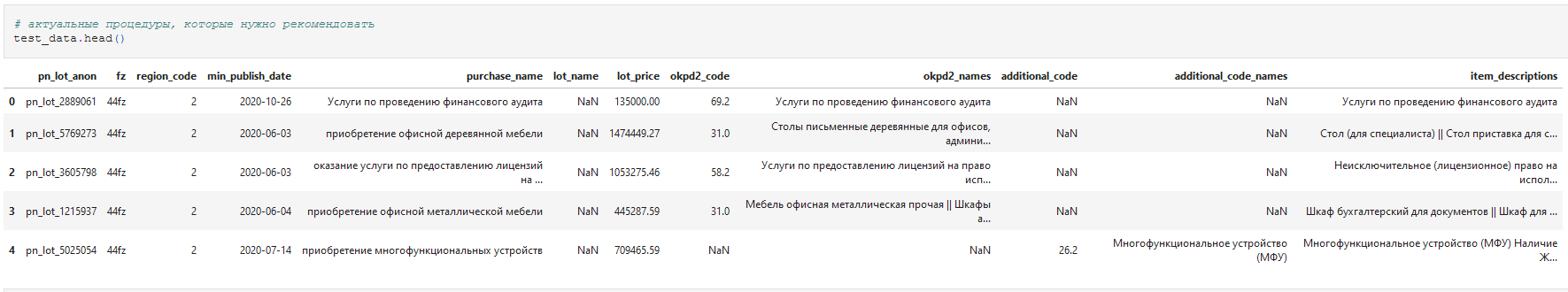
- test\_data.csv – уникальные актуальные процедуры.

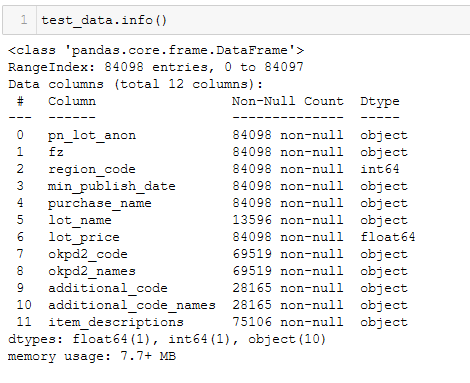
2.2 train data – уникальные процедуры за период обучения





2.3 test data – уникальные актуальные процедуры (которые мы рекомендуем);





pn\_lot\_anon – анонимизированный номер связки процедура-лот;

fz – федеральный закон, к которому относится процедура;

region\_code – код региона (справочник регионов приложен);

min\_publish\_date – дата первой публикации извещения;

purchase\_name – название закупки;

lot\_name – название лота;

lot\_price – цена лота (в руб.);

okpd2\_code – код ОКПД2;

okpd2\_names – название кода ОКПД2 (разделитель “||”);

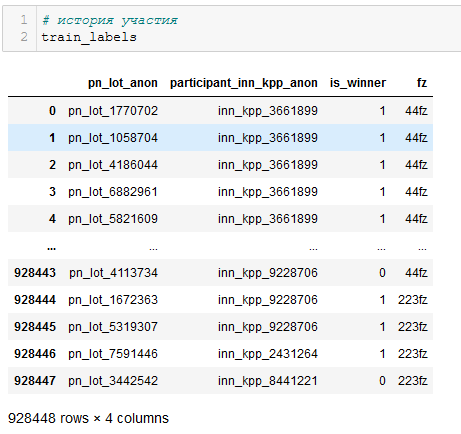
additional\_code – добавочный код (если нет кода ОКПД2, используется КТРУ для 44-ФЗ или ОКВЭД2 для 223-ФЗ);

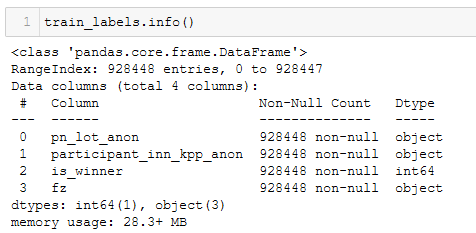
additional\_code\_names – название добавочного кода (разделитель “||”);

item\_descriptions – описание товаров (разделитель “||”).

Код ОКПД2 имеет несколько уровней вложенности. Например, «24.20.31.000 – трубы сварные для нефте- и газопроводов, наружным диаметром не более 406,4 мм, стальные». В поле okpd2\_names название кода ОКПД2 всегда соответствует реальному уровню вложенности, а в okpd2\_code код. ОКПД2 верхнеуровневый. В нашем примере okpd2\_names сохраняется, но okpd2\_code будет 24.2 (не больше одного знака после первой точки). Аналогично для additional\_code и additional\_code\_names.

2.4 train\_labels – разметка, связь поставщика с процедурой, в которой он участвовал;





pn\_lot\_anon – анонимизированный номер связки процедура-лот из train\_data;

participant\_inn\_kpp\_anon – анонимизированный ИНН\_КПП поставщика;

is\_winner – 1 – если поставщик победил в соответствующей процедуре, 0 – поставщик участвовал в соответствующей процедуре;

fz – федеральный закон, к которому относится процедура.

Ссылка на полный код: <https://github.com/FED32/Diplom_ds/blob/master/Diplom/notebooks/00_Input_data_analyse.ipynb>

# 3 Подготовка и генерация данных (feature engineering)

3.1 Текстовые данные

3.1.1 Формируем столбец с текстовым описанием тендера путем объединения содержимого из столбцов названия закупки, названия лота, названия кода ОКПД2 или дополнительного кода, описания товаров

train\_data[‘text\_description\_tender’] = train\_data[‘purchase\_name’] + “ “ + train\_data[‘lot\_name’] + “ “ + train\_data[‘okpd2\_names’] + “ “ + train\_data[‘additional\_code\_names’] + “ “ + train\_data[‘item\_descriptions’]

test\_data[‘text\_description\_tender’] = test\_data[‘purchase\_name’] + “ “ + test\_data[‘lot\_name’] + “ “ + test\_data[‘okpd2\_names’] + “ “ + test\_data[‘additional\_code\_names’] + “ “ + test\_data[‘item\_descriptions’]

3.1.2 Определяем функции для обработки текста

mystopwords = stopwords.words(‘russian’)

ru\_words = re.compile(“[А-Яа-я]+”)

# функция фильтрующая текст от остальных символов

def words\_only(text):

return “ “.join(ru\_words.findall(text))

# функция лемматизации текста

def lemmatize(text):

try:

return “ “.join(pymorphy2\_analyzer.parse(word)[0].normal\_form for word in text.split())

except:

return « «

# функция для удаления стоп-слов

def remove\_stopwords(text, mystopwords = mystopwords):

try:

return “ “.join([token for token in text.split() if not token in mystopwords])

except:

return «»

# функция для удаления дубликатов (для случая когда в text\_description\_tender попал одинаковый текст при объединении текста из ячеек)

def remove\_word\_duplicates(text):

text\_ = []

for word in text.split():

if word not in text\_:

text\_.append(word)

return ‘ ‘.join(text\_)

# итоговая функция обработки текста

def preprocess(text):

return remove\_word\_duplicates(remove\_stopwords(lemmatize(words\_only(text.lower()))))

3.1.3 Обработка текста путем вызова итоговой функции обработки текста

train\_data[‘text\_description\_tender\_stemmed’] = train\_data.text\_description\_tender.apply(preprocess)

test\_data[‘text\_description\_tender\_stemmed’] = test\_data.text\_description\_tender.apply(preprocess)

3.1.4 Также приведем pn\_lot\_anon – анонимизированный номер связки процедура-лот к числовому типу данных для удобства.

digits = re.compile(“[0-9]+”)

def digits\_only(text):

return int(‘’.join(digits.findall(text)))

train\_data[‘pn\_lot\_id’] = train\_data.pn\_lot\_anon.apply(digits\_only)

test\_data[‘pn\_lot\_id’] = test\_data.pn\_lot\_anon.apply(digits\_only)

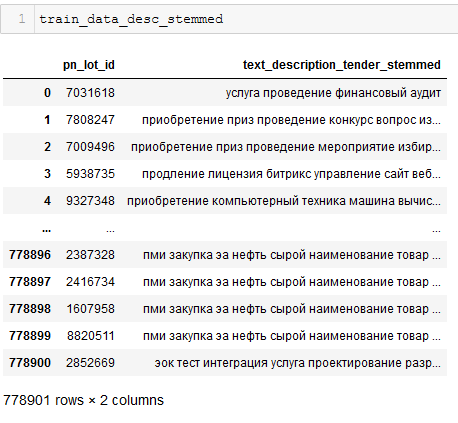
3.1.5 Создаем датасеты с описаниями процедур и сохраняем в csv-файлы для дальнейшего использования.

train\_data\_desc\_stemmed = train\_data[[‘pn\_lot\_id’, ‘text\_description\_tender\_stemmed’]]

test\_data\_desc\_stemmed = test\_data[[‘pn\_lot\_id’, ‘text\_description\_tender\_stemmed’]]

train\_data\_desc\_stemmed.to\_csv(‘../data/intermid/train\_data\_desc\_stemmed.csv’, index=False, encoding=’utf-8’)

test\_data\_desc\_stemmed.to\_csv(‘../data/intermid/test\_data\_desc\_stemmed.csv’, index=False, encoding=’utf-8’)





Ссылка на полный код: <https://github.com/FED32/Diplom_ds/blob/master/Diplom/notebooks/01_Data_description_preprocessing.ipynb>

3.1.6 Векторизация текста (1 вариант)

В качестве первого способа векторизации текста используем TF-IDF с последующим снижением пространства методом TruncatedSVD.

TF-IDF (от англ. TF — term frequency, IDF — inverse document frequency) — статистическая мера, используемая для оценки важности слова в контексте документа, являющегося частью коллекции документов или корпуса. Вес некоторого слова пропорционален частоте употребления этого слова в документе и обратно пропорционален частоте употребления слова во всех документах коллекции.

Мера TF-IDF часто используется для представления документов коллекции в виде числовых векторов, отражающих важность использования каждого слова из некоторого набора слов (количество слов набора определяет размерность вектора) в каждом документе. Подобная модель называется векторной моделью и даёт возможность сравнивать тексты, сравнивая представляющие их вектора в какой-либо метрике (евклидово расстояние, косинусная мера, манхэттенское расстояние, расстояние Чебышёва и др.), то есть производя кластерный анализ.

TruncatedSVD выполняет уменьшение линейной размерности с помощью усеченного сингулярного разложения (SVD). В отличие от PCA, этот оценщик не центрирует данные перед вычислением разложения по сингулярным значениям. Это означает, что он может работать с матрицами scipy.sparse. В частности, TruncatedSVD работает с матрицами count / tf-idf, возвращаемыми векторизаторами в sklearn.feature\_extraction.text. В этом контексте он известен как латентный семантический анализ (LSA).

import pandas as pd

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.decomposition import TruncatedSVD

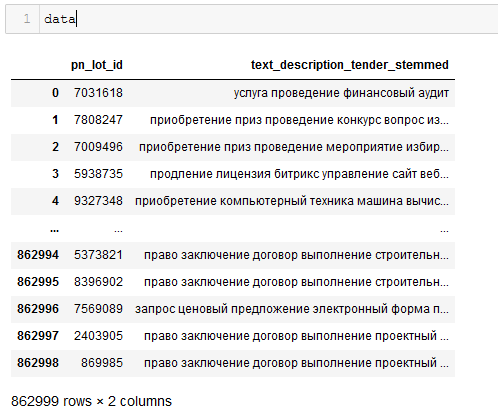
# считываем датасеты с нормированными описаниями тендеров

train\_data\_desc\_stemmed = pd.read\_csv(‘../data/intermid/train\_data\_desc\_stemmed.csv’)

test\_data\_desc\_stemmed = pd.read\_csv(‘../data/intermid/test\_data\_desc\_stemmed.csv’)

# соединяем датасеты

data = pd.concat([train\_data\_desc\_stemmed, test\_data\_desc\_stemmed], ignore\_index=True)



Строим TF-IDF матрицу документ-слово

vectors = TfidfVectorizer().fit\_transform(data.text\_description\_tender\_stemmed)

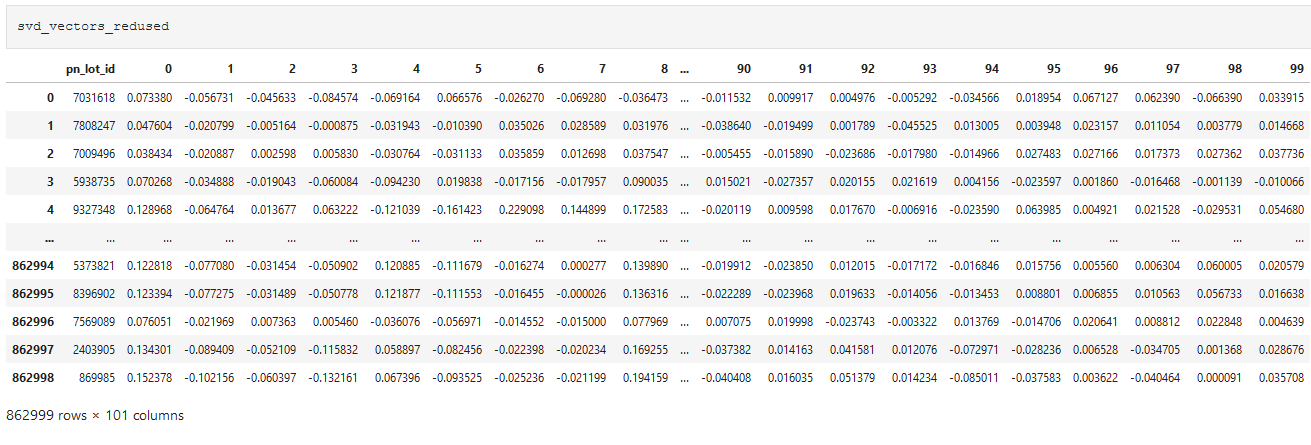
Оставляем 100 главных компонент

X\_reduced = TruncatedSVD(n\_components=100, random\_state=0).fit\_transform(vectors)

Создаем датасет из идентификаторов и векторов и сохраняем в csv-файл для дальнейшего использования

svd\_vectors\_redused = pd.DataFrame(data[‘pn\_lot\_id’]).join(pd.DataFrame(X\_reduced))

svd\_vectors\_redused.to\_csv(‘../data/intermid/svd\_vectors\_redused.csv’, index=False, encoding=’utf-8’)



Ссылка на полный код: <https://github.com/FED32/Diplom_ds/blob/master/Diplom/notebooks/02_svd_vectors.ipynb>

3.1.7 Векторизация текста (2 вариант)

В качестве альтернативного способа векторизации текстовых описаний используем модель Doc2Vec из библиотеки enism.

Алгоритм Doc2Vec (первоначальное название Paragraph Vector), алгоритм обучения без учителя, учится получать распределенные векторы для частей текстов. Тексты могут быть переменной длины: от предложения до большого документа. В данной модели векторные представления документов обучаются предсказывать слова в документе, точнее берется вектор документа и объединяется с несколькими векторами слов из него, и модель пытается предсказать следующее слово с учетом контекста. Векторы слов и документов обучаются с использованием метода стохастического градиентного спуска и метода обратного распространения ошибки. Векторы документов являются уникальными, а векторы одинаковых слов в разных документах совпадают.

Существует 2 архитектуры для построения векторных представлений документов:

1. Distributed Memory (DM, D2V-DM);

2. Distributed Bag-of-Words (DBOW, D2V-DBOW).

D2V-DM. В данной конструкции каждый документ представлен уникальным вектором в виде столбца в матрице D, и каждый термин представлен уникальным вектором в виде столбца в матрице W. Вектор .документ и векторы слов в нем объединяются или усредняются для предсказания следующего слова из контекста.

D2V-DBOW. Описанный выше метод рассматривает объединение вектора документа с векторами слов, входящих в него, для предсказания следующего слова в текстовом окне. Другой способ заключается в игнорировании слов из контекста на входе, но при этом модель должна предсказывать случайно отобранные слова для документа на выходе. Это означает, что на каждой итерации стохастического градиентного спуска просматривается текстовое окно, затем просматривается случайное слово в текстовом окне и формируется задача классификации с учетом вектора документа. Эта версия называется Distributed Bag-of-Words.

# считываем датасеты с нормированными описаниями тендеров

train\_data\_desc\_stemmed = pd.read\_csv(‘../data/intermid/train\_data\_desc\_stemmed.csv’)

test\_data\_desc\_stemmed = pd.read\_csv(‘../data/intermid/test\_data\_desc\_stemmed.csv’)

# соединяем датасеты

data = pd.concat([train\_data\_desc\_stemmed, test\_data\_desc\_stemmed], ignore\_index=True)

Чтобы обучить модель, нам нужно связать тег/номер с каждым документом корпуса обучения. В нашем случае тег – это просто отсчитываемый от нуля номер строки.

# функция считывания корпуса

def read\_corpus(corpus, tokens\_only=False):

for i, line in enumerate(corpus):

tokens = enism.utils.simple\_preprocess(line)

if tokens\_only:

yield tokens

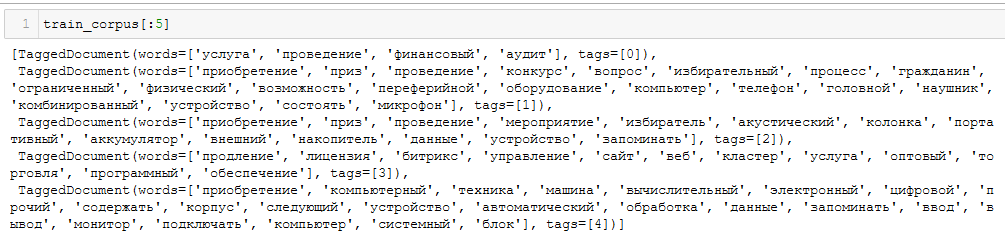
else:

# For training data, add tags

yield enism.models.doc2vec.TaggedDocument(tokens, [i])

# формируем корпус

train\_corpus = list(read\_corpus(data.text\_description\_tender\_stemmed))



Cоздадим экземпляр модели Doc2Vec с векторным размером с 100 измерениями и выполним итерацию по обучающей совокупности 100 раз. Устанавливаем минимальное количество слов равным 2, чтобы отбрасывать слова с очень небольшим количеством вхождений.

model = enism.models.doc2vec.Doc2Vec(vector\_size=100, min\_count=2, epochs=100)

# создаем словарь

model.build\_vocab(train\_corpus)

Тренируем модель на корпусе

model.train(train\_corpus, total\_examples=model.corpus\_count, epochs=model.epochs)

# сохраняем модель

model\_path = ‘../models/doc2vec\_c2\_e100.model’

print(“Saving model…”)

model.save(model\_path)

Получаем векторы описаний используя обученную модель, передав список слов в функцию model.infer\_vector.

vectors = []

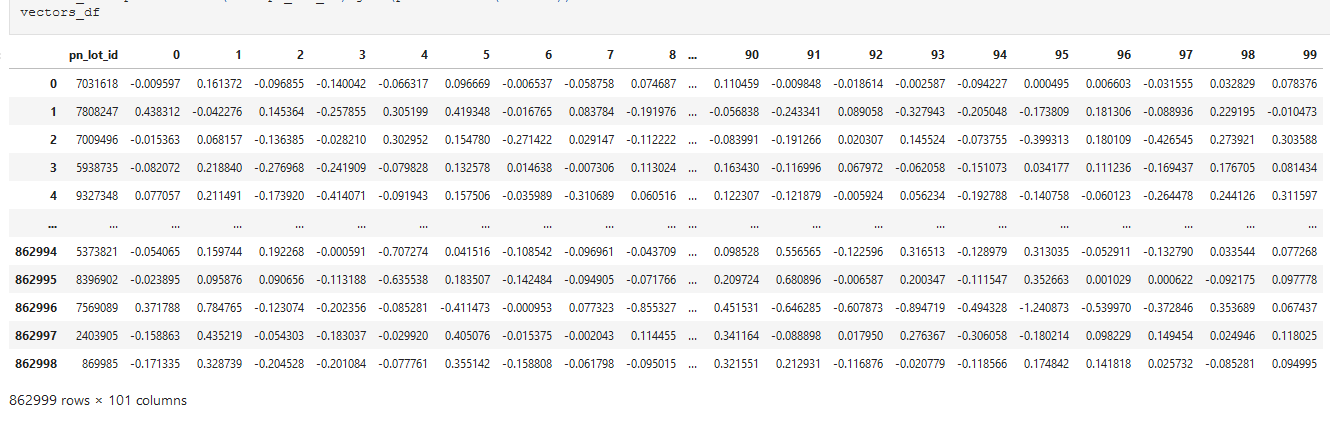
for doc\_id in tqdm(range(len(train\_corpus))):

vectors.append(model.infer\_vector(train\_corpus[doc\_id].words))

Создаем датасет из идентификаторов и векторов и сохраняем в csv файл для дальнейшего использования

vectors\_df = pd.DataFrame(data.pn\_lot\_id).join(pd.DataFrame(vectors))

vectors\_df.to\_csv(‘../data/intermid/doc2Vec\_vectors.csv’, index=False, encoding=’utf-8’)



Ссылка на полный код: <https://github.com/FED32/Diplom_ds/blob/master/Diplom/notebooks/03_Doc2vec.ipynb>

3.2 Числовые и категориальные переменные

3.2.1 Собираем датасет

# загружаем данные

train\_data = pd.read\_csv(“../data/external/train\_data.csv”, sep=’;’)

test\_data = pd.read\_csv(“../data/external/test\_data.csv”, sep=’;’)

#Заполнение пропусков строкой ‘None’

train\_data = train\_data.fillna(‘None’)

test\_data = test\_data.fillna(‘None’)

# Обработка okpd2 и additional\_code

train\_data[‘okpd2\_or\_additional\_code’] = train\_data[[‘okpd2\_code’, ‘additional\_code’]].apply(lambda x: x[0] if x[1] == ‘None’ else x[1], axis=1)

test\_data[‘okpd2\_or\_additional\_code’] = test\_data[[‘okpd2\_code’, ‘additional\_code’]].apply(lambda x: x[0] if x[1] == ‘None’ else x[1], axis=1)

# приведение идентификаторов к числовому типу

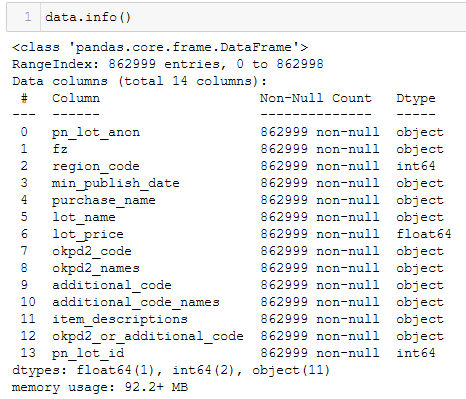
train\_data[‘pn\_lot\_id’] = train\_data.pn\_lot\_anon.apply(digits\_only)

test\_data[‘pn\_lot\_id’] = test\_data.pn\_lot\_anon.apply(digits\_only)

# объединяем тренировочные и тестовые данные

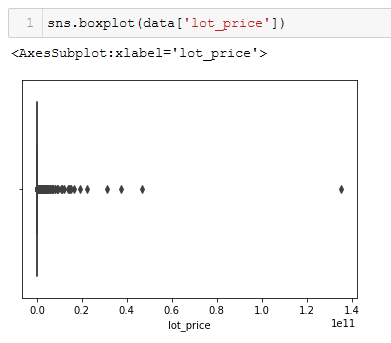
data = pd.concat([train\_data, test\_data], ignore\_index=True)





3.2.2 Обработаем числовой признак – цену лота.

Посмотрим на разброс значения цены



Логарифмируем и нормируем цену в диапазоне от 0 до 1

# прологарифмируем цену

data[‘log\_lot\_price’] = np.log(data.lot\_price)

# для нулевых цен заменим отрицательные значения логарифма нулем

data.loc[data[‘log\_lot\_price’] < 0, ‘log\_lot\_price’] = 0

# нормируем логарифмированную цену и помещаем результат в отдельный столбец

from sklearn.preprocessing import MinMaxScaler, StandardScaler

mm\_scaler = MinMaxScaler()

data[‘scale\_log\_lot\_price’] = mm\_scaler.fit\_transform(pd.DataFrame(data.log\_lot\_price))

3.2.3 Обработаем категориальный признак – дату публикации, извлечем месяц и закодируем его с помощью OneHotEncoder.

# cоздаем регулярное выражение для извлечения

re\_month = re.compile(«[0-9]{4}-([0-9]{2})»)

# определяем функцию, извлекающую месяц из даты

def extract\_month(text):

return ‘’.join(re\_month.findall(text))

# применяем функцию к столбцу min\_publish\_date и создаем столбец с результатом

data[‘month’] = data[‘min\_publish\_date’].apply(extract\_month)

Кодировщик OneHotEncoder берёт столбец с категориальными данными, который был предварительно закодирован в признак, и создаёт для него несколько новых столбцов. Числа заменяются на единицы и нули, в зависимости от того, какому столбцу какое значение присуще.

from sklearn.preprocessing import OneHotEncoder

ohe = OneHotEncoder()

# создаем матрицу категориальных признаков месяца

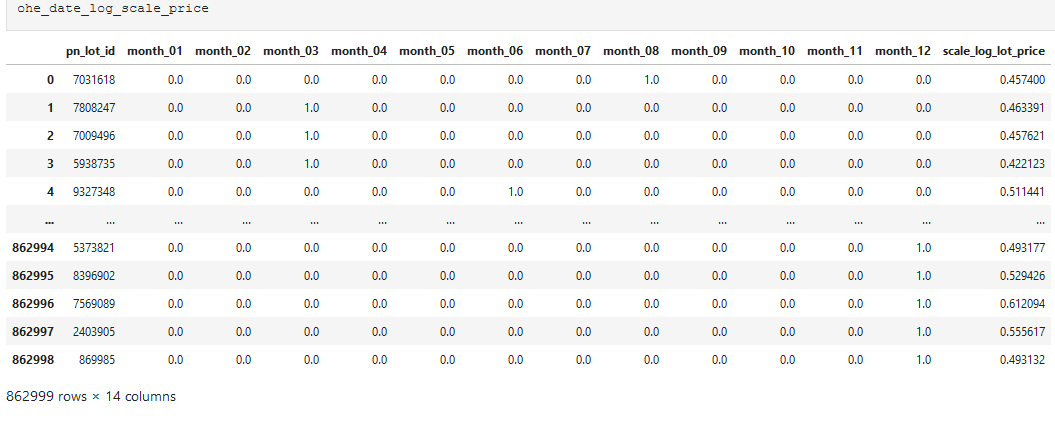
ohe\_month = ohe.fit\_transform(data[[‘month’]]).toarray()

Создаем датасет, соединяем индексы процедур с матрицей категориальных признаков, переименовываем ячейки, добавляя префикс month к именам категорий

ohe\_month = pd.DataFrame(data[‘pn\_lot\_id’]).join(pd.DataFrame(ohe\_month, columns=’month\_’+ohe.categories\_[0]))

Присоединяем к данному датасету датасет с логарифмированной нормированной ценой и сохраняем в csv-файл для дальнейшей работы.

ohe\_date\_log\_scale\_price = ohe\_month.join(data[‘scale\_log\_lot\_price’])



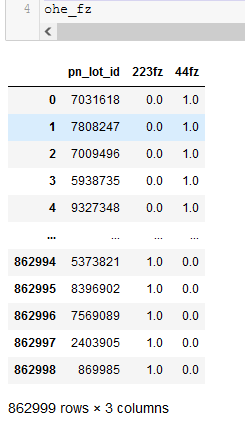
3.2.4 Обработаем категориальный признак – федеральный закон (fz).

Закодируем его с помощью OneHotEncoder, соединим индексы процедур с матрицей категориальных признаков, и сохраним в csv-файл для дальнейшего использования. Имена столбцов задаем по именам категорий.

ohe\_fz = ohe.fit\_transform(data[[‘fz’]]).toarray()

ohe\_fz = pd.DataFrame(data[‘pn\_lot\_id’]).join(pd.DataFrame(ohe\_fz, columns=ohe.categories\_[0]))

ohe\_fz.to\_csv(‘../data/intermid/ohe\_fz.csv’, index=False, encoding=’utf-8’)



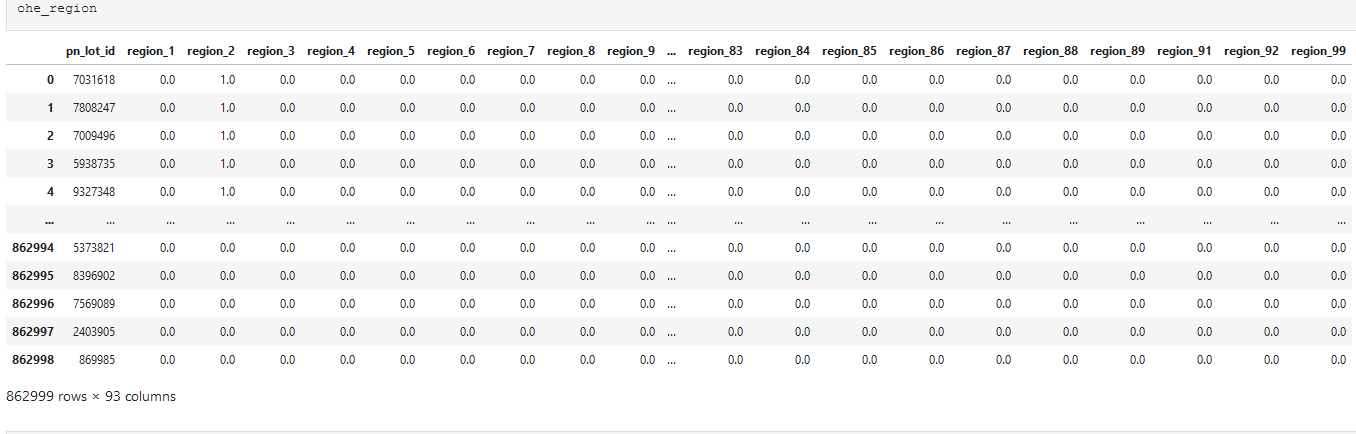
3.2.5 Обработаем категориальный признак – код региона (region\_code).

Закодируем его с помощью OneHotEncoder, соединим индексы процедур с матрицей категориальных признаков, и сохраним в csv-файл для дальнейшего использования. Имена столбцов задаем по именам категорий с добавлением префикса region.

ohe\_region = ohe.fit\_transform(data[[‘region\_code’]]).toarray()

ohe\_region = pd.DataFrame(data[‘pn\_lot\_id’]).join(pd.DataFrame(ohe\_region, columns=[‘region\_’+item for item in ohe.categories\_[0].astype(str)]))

ohe\_region.to\_csv(‘../data/intermid/ohe\_region.csv’, index=False, encoding=’utf-8’)



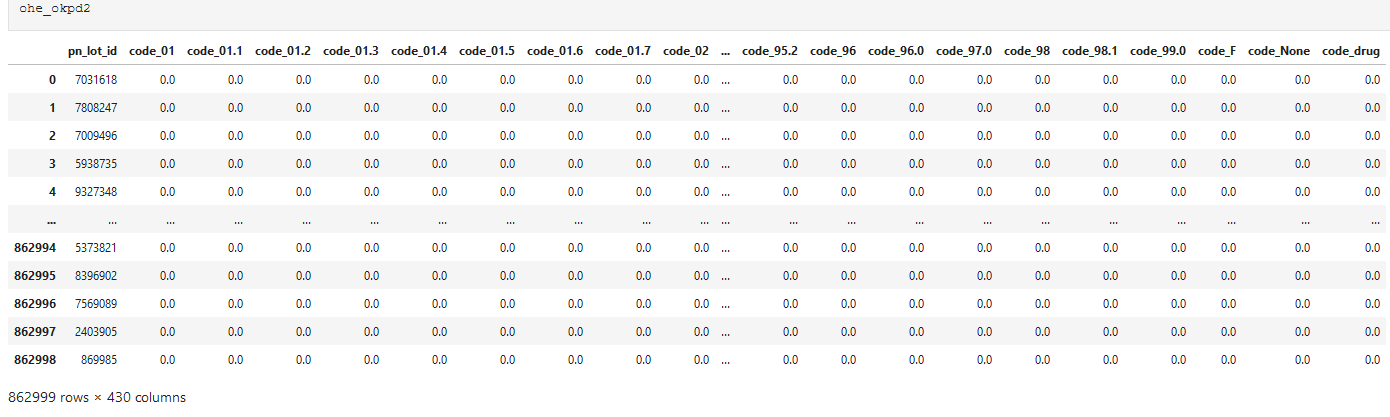
3.2.6 Обработаем категориальный признак – ОКПД2 или дополнительный код (okpd2\_or\_additional\_code).

Закодируем его с помощью OneHotEncoder, соединим индексы процедур с матрицей категориальных признаков, и сохраним в csv-файл для дальнейшего использования. Имена столбцов задаем по именам категорий с добавлением префикса code.

Ohe\_okpd2 = ohe.fit\_transform(data[[‘okpd2\_or\_additional\_code’]]).toarray()

ohe\_okpd2 = pd.DataFrame(data[‘pn\_lot\_id’]).join(pd.DataFrame(ohe\_okpd2, columns=’code\_’+ohe.categories\_[0]))

ohe\_okpd2.to\_csv(‘../data/intermid/ohe\_okpd2.csv’, index=False, encoding=’utf-8’)



Ссылки на полный код: <https://github.com/FED32/Diplom_ds/blob/master/Diplom/notebooks/04_feature_engineering_p1_(ohe_fz_region_okpd2).ipynb>,

<https://github.com/FED32/Diplom_ds/blob/master/Diplom/notebooks/05_feature_engineering_p2_(ohe_date_log_scale_price).ipynb>

# 4 Моделирование

4.1 Выбор алгоритма

В качестве модели используем библиотеку LightFM.

LightFM – это реализация на Python ряда популярных алгоритмов рекомендаций как для неявной (implicit feddback), так и для явной обратной связи (explicit feedback).

Это также позволяет включать как элементы, так и метаданные пользователя в традиционные алгоритмы матричной факторизации. Он представляет каждого пользователя и элемент как сумму скрытых представлений их признаков, что позволяет обобщать рекомендации для новых элементов (с помощью функций элемента) и для новых пользователей (с помощью функций пользователя).

Взаимосвязь между Light FM и коллаборативной моделью MF определяется структурой наборов функций пользователя и элементов. Если наборы функций состоят исключительно из индикаторных переменных для каждого пользователя и элемента, LightFM сводится к стандартной модели MF. Если наборы функций также содержат функции метаданных, совместно используемые более чем одним элементом или пользователем, LightFM расширяет модель MF, позволяя скрытым факторам функций частично объяснять структуру взаимодействий пользователей.

Это важно по трем пунктам.

1. В большинстве приложений будет меньше функций метаданных, чем пользователей или элементов, либо потому, что используется онтология с фиксированной структурой типа/категории, либо потому, что при использовании необработанных текстовых функций поддерживается словарь наиболее распространенных терминов фиксированного размера. Это означает, что требуется оценить меньшее количество параметров на основе ограниченных данных обучения, что снижает риск переобучения и улучшает производительность обобщения.

2. Скрытые векторы для индикаторных переменных не могут быть оценены для новых пользователей или элементов с холодным запуском. Представление их в виде комбинаций функций метаданных, которые могут быть оценены на основе обучающего набора, позволяет делать прогнозы «холодного запуска».

3. Если присутствуют только функции индикатора, LightFM должен работать наравне со стандартной моделью MF.

Когда присутствуют только функции метаданных и отсутствуют переменные показателей, модель в целом не сводится к чистой системе, основанной на содержании. LightFM оценивает вложения функций путем факторизации матрицы совместного взаимодействия; это не похоже на системы, основанные на контенте, которые (при использовании уменьшения размерности) факторизуют матрицы совпадений чистого контента.

Один особый случай, когда LightFM сводится к чистой модели CB, заключается в том, что каждый пользователь описывается переменной индикатора и взаимодействовал только с одним элементом. В этом случае вектор пользователя эквивалентен вектору документа в формулировке LSI, и только функции, которые встречаются вместе в описаниях продуктов, будут иметь аналогичные вложения.

Тот факт, что LightFM содержит как чистую модель CB на конце спектра с разреженными данными, так и модель MF на плотном конце, предполагает, что она должна хорошо адаптироваться к наборам данных различной разреженности. Фактически, эмпирические результаты показывают, что она работает, по крайней мере, так же хорошо, как и соответствующая специализированная модель в каждом сценарии.

Применительно к данной рекомендательной системе, мы имеем данные о взаимодействиях пользователей, а также набор признаков элементов, поэтому использование LightFM является наиболее оптимальным, и полностью использует имеющиеся данные.

4.2 План тестирования

Работу с моделью будем выполнять в следующей последовательности:

* 1. Сборка датасета признаков элементов из ранее сохраненных csv-файлов.
  2. Считывание и предобработка тренировочных данных, преобразование идентификаторов к числовому типу.
  3. Формирование датасета в формате библиотеки:

- построение матрицы взаимодействий

- построение матрицы признаков элементов.

* 1. Задание параметров модели
  2. Обучение модели с разными вариантами параметров
  3. Формирование предсказаний обученных моделей и сохранение результатов для дальнейшей оценки.

4.3 Работа с моделью LightFM

4.3.1 Импортируем необходимые библиотеки

import numpy as np

import pandas as pd

import re

from tqdm import tqdm\_notebook

from tqdm import tqdm

from lightfm import LightFM

from lightfm.evaluation import precision\_at\_k

from lightfm.evaluation import recall\_at\_k

from lightfm.evaluation import auc\_score

from lightfm.data import Dataset

from lightfm.cross\_validation import random\_train\_test\_split

4.3.2 Загружаем входные данные, собираем датасеты

train\_labels = pd.read\_csv(“../data/external/train\_labels.csv”, sep=’;’)

test\_data = pd.read\_csv(“../data/external/test\_data.csv”, sep=’;’)

ohe\_fz = pd.read\_csv(“../data/intermid/ohe\_fz.csv”)

ohe\_okpd2 = pd.read\_csv(“../data/intermid/ohe\_okpd2.csv”)

ohe\_region = pd.read\_csv(“../data/intermid/ohe\_region.csv”)

ohe\_date\_scale\_price = pd.read\_csv(“../data/intermid/ohe\_date\_log\_scale\_price.csv”)

text\_vectors = pd.read\_csv(“../data/intermid/doc2Vec\_vectors.csv”)

text\_vectors = pd.read\_csv(“../data/intermid/svd\_vectors\_redused.csv”)

При этом в одном случае текстовые векторы загружаем из результата обработки методом TF-IDF с последующим снижением пространства методом TruncatedSVD, во втором случае текстовые векторы загружаем из результата обработки моделью Doc2vec.

Для удобства приводим идентификаторы к числовому типу

train\_labels[‘pn\_lot\_anon’] = train\_labels[‘pn\_lot\_anon’].apply(digits\_only)

train\_labels[‘participant\_inn\_kpp\_anon’] = train\_labels[‘participant\_inn\_kpp\_anon’].apply(digits\_only)

test\_data[‘pn\_lot\_anon’] = test\_data[‘pn\_lot\_anon’].apply(digits\_only)

Собираем датасет признаков элементов

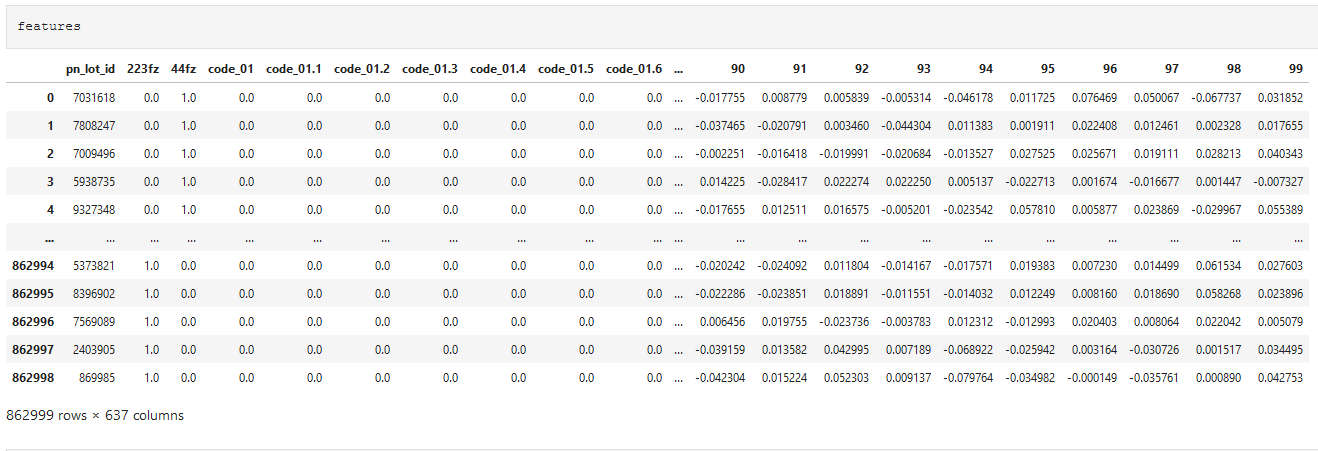
features = ohe\_fz.merge\

(ohe\_okpd2, how=’left’, left\_on=’pn\_lot\_id’, right\_on=’pn\_lot\_id’).merge\

(ohe\_region, how=’left’, left\_on=’pn\_lot\_id’, right\_on=’pn\_lot\_id’).merge\

(ohe\_date\_scale\_price, how=’left’, left\_on=’pn\_lot\_id’, right\_on=’pn\_lot\_id’).merge\

(text\_vectors, how=’left’, left\_on=’pn\_lot\_id’, right\_on=’pn\_lot\_id’)



4.3.3 Формируем списки с идентификаторами пользователей, идентификаторами элементов, список имен признаков элементов, передаем их в датасет и тренируем.

user\_ids = train\_labels.participant\_inn\_kpp\_anon.unique().tolist()

item\_ids = features.pn\_lot\_id.unique().tolist()

item\_features\_names = features.columns.tolist()[1:]

dataset = Dataset(user\_identity\_features=False, item\_identity\_features=True)

dataset.fit(users=user\_ids, items=item\_ids, item\_features=item\_features\_names)

Формируем словари соответствия индексов LightFM с исходными идентификаторами

user\_mappings1, \_, item\_mappings1, \_ = dataset.mapping()

4.3.4 Увеличиваем значение is\_winner на 1 чтобы уровень взаимодействия для новых процедур не сравнялся с участием в процедуре, но ее проигрышем.

train\_labels[‘is\_winner’] = train\_labels[‘is\_winner’]+1

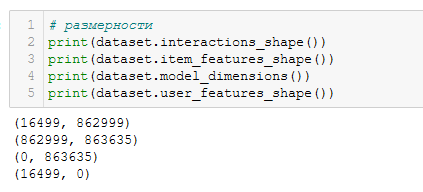
4.3.5 Строим матрицу взаимодействий

interactions = dataset.build\_interactions(np.array(train\_labels[[‘participant\_inn\_kpp\_anon’, ‘pn\_lot\_anon’, ‘is\_winner’]]))

4.3.6 Строим матрицу признаков элементов

item\_features = dataset.build\_item\_features(((row[0].astype(int), dict(zip(item\_features\_names, row[1:].tolist()))) for row in np.array(features)), normalize=False)

Посмотрим на размерности данных модели:



4.3.7 Создаем модель и определяем параметры

no\_components=100

epochs=10

no\_components (int, необязательно) – размерность скрытых вложений признаков;

epochs (int, необязательно) – количество эпох.

model = LightFM(no\_components=no\_components, loss=’warp’, item\_alpha=1e-6)

Функцию потерь выбираем «WARP».

WARP рассматривает каждое наблюдаемое взаимодействие пользователя с элементом (u, i), в свою очередь, выбирает другой “отрицательный” элемент i', который, по мнению модели, был более подходящим для пользователя, и выполняет градиентные обновления параметров модели, связанных с u, i и i', так что веса моделей корректируются. WARP взвешивает обновления градиента с использованием (функции) оценочного ранга элемента i для пользователя u. Таким образом, шаг увеличивается, если модель не верила, что взаимодействие (u, i) может когда-либо произойти, и ослабляется, если, с другой стороны, взаимодействие не является неожиданным для модели. Удобно, что ранг i для u можно оценить, подсчитав количество элементов выборки i', которые необходимо было рассмотреть, прежде чем было обнаружено, что модель (ошибочно) считалась более подходящей для пользователя u.

Параметры модели будем изменять в соответствии с таблицей 1:

Таблица 1

| Имя модели | text\_vectors | no\_components | epochs |
| --- | --- | --- | --- |
| lightfm\_svd\_full\_data\_100c\_10e.model | svd\_vectors\_redused.csv | 100 | 10 |
| lightfm\_svd\_full\_data\_50c\_10e.model | svd\_vectors\_redused.csv | 50 | 10 |
| lightfm\_svd\_full\_data\_75c\_10e.model | svd\_vectors\_redused.csv | 75 | 10 |
| lightfm\_svd\_full\_data\_100c\_5e.model | svd\_vectors\_redused.csv | 100 | 5 |
| lightfm\_svd\_full\_data\_100c\_20e.model | svd\_vectors\_redused.csv | 100 | 20 |
| lightfm\_doc2vec\_full\_data\_100c\_10e.model | doc2Vec\_vectors.csv | 100 | 10 |

4.3.8 Обучаем модель, изменяя параметры в соответствии с таблицей

model.fit(interactions[0], user\_features=None, item\_features=item\_features, sample\_weight=interactions[1], epochs=epochs, verbose=True)

4.3.9 Сохраняем модель на диск

import pickle

path = ‘../models/lightfm\_svd\_full\_data\_100c\_10e.model’

# path = ‘../models/lightfm\_svd\_full\_data\_50c\_10e.model’

# path = ‘../models/lightfm\_svd\_full\_data\_75c\_10e.model’

# path = ‘../models/lightfm\_svd\_full\_data\_100c\_5e.model’

# path = ‘../models/lightfm\_svd\_full\_data\_100c\_20e.model’

# path = ‘../models/lightfm\_doc2vec\_full\_data\_100c\_10e.model’

with open(path, ‘wb’) as f:

pickle.dump(model, f)

Строчку с именем файла используем в соответствии с параметрами модели.

4.3.10 Формируем предсказания для каждого пользователя и сохраняем в файл.

Для каждого пользователя рекомендуем 35 актуальных процедур.

Поля файла – inn\_kpp, actual\_recommended\_pn\_lot, similarity\_score. (inn\_kpp – анонимизированный ИНН\_КПП поставщика, actual\_recommended\_pn\_lot – анонимизированный номер связки процедура-лот, similarity\_score – числовой критерий, показывающий релевантность актуальной процедуры для поставщика)

# формируем список индексов актуальных процедур

user\_mappings = {k:d for d,k in user\_mappings1.items()}

item\_mappings = {k:d for d,k in item\_mappings1.items()}

actual\_item\_mapping = pd.Series(item\_mappings1)[test\_data.pn\_lot\_anon.tolist()]

items = actual\_item\_mapping.tolist()

# Формируем рекомендации для каждого пользователя

res = []

for user\_id, user\_index in tqdm(user\_mappings1.items()):

res1 = pd.Series(model.predict(user\_index, items, item\_features=item\_features), items).sort\_values(ascending=False).keys()[:35]

rec = pd.Series(item\_mappings)[res1].tolist()

for r in rec:

res.append((user\_id, r))

recommendations = pd.DataFrame(res)

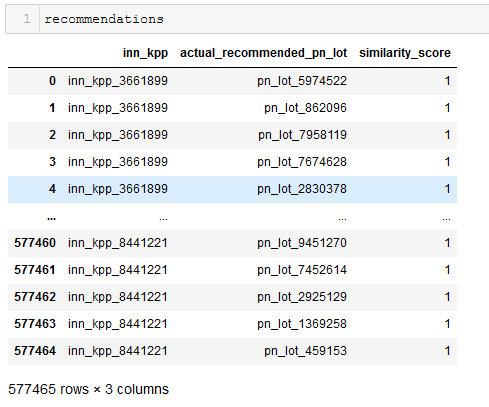
recommendations[‘similarity\_score’] = 1

recommendations.columns = [‘inn\_kpp’, ‘actual\_recommended\_pn\_lot’, ‘similarity\_score’]

recommendations[‘inn\_kpp’] = ‘inn\_kpp\_’ + recommendations[‘inn\_kpp’].astype(str)

recommendations[‘actual\_recommended\_pn\_lot’] = ‘pn\_lot\_’ + recommendations[‘actual\_recommended\_pn\_lot’].astype(str)

Вид датасета с рекомендациями:



Ссылка на полный код: <https://github.com/FED32/Diplom_ds/blob/master/Diplom/notebooks/06_Light_FM_(full_data).ipynb>

# 5 Оценка качества моделей

Оценка качества модели, разбиение выборки на тренировочную и тестовую, подсчет метрик (precision, recall, auc) средствами библиотеки LightFM не проводится на полных данных по причине ограниченности вычислительных ресурсов.

Оценка качества модели проводится методом подсчета метрик на основе данных о взаимодействии пользователей с актуальными процедурами.

Метрики:

1) Полнота – кол-во фактических поданных заявок из рекомендаций /кол-во участий.

2) Точность – кол-во фактических поданных заявок из рекомендаций /кол-во рекомендаций.

Основной бизнес-метрикой является полнота, при условии, что не более 35 актуальных процедур мы рекомендуем одному поставщику.

# загружаем необходимые файлы

recommendations = pd.read\_csv(“../data/processed/recommendations\_svd.csv”, sep=’;’)

*# recommendations = pd.read\_csv(“../data/processed/recommendations\_svd\_v2.csv”, sep=’;’)*

*# recommendations = pd.read\_csv(“../data/processed/recommendations\_svd\_v3.csv”, sep=’;’)*

*# recommendations = pd.read\_csv(“../data/processed/recommendations\_svd\_v4.csv”, sep=’;’)*

*# recommendations = pd.read\_csv(“../data/processed/recommendations\_svd\_v5.csv”, sep=’;’)*

*# recommendations = pd.read\_csv(“../data/processed/recommendations\_doc2vec.csv”, sep=’;’)*

test\_labels = pd.read\_csv(“../data/external/test\_labels.csv”, sep=’;’)

# формируем из данных множества

true = set((test\_labels[‘pn\_lot\_anon’] + “\_” + test\_labels[‘participant\_inn\_kpp\_anon’]).values)

pred = set((recommendations[‘actual\_recommended\_pn\_lot’] + “\_” + recommendations[‘inn\_kpp’]).values)

# количество совпадений

intersection = len(true.intersection(pred))

# рассчет метрик

print(f”Точность: {intersection / len(pred) \* 100}”)

print(f”Полнота: {intersection / len(true) \* 100}”)

Результаты расчёта метрик, и параметры моделей сводим в таблицу 2.

Максимальное значение полноты, которое удалось получить – 24,383.

Обучение проводилось на ноутбуке Lenovo Z510, Intel™ Core™ i7-4702MQ CPU @ 2.20GHz, RAM 8 Гб.

Ссылка на полный код: <https://github.com/FED32/Diplom_ds/blob/master/Diplom/notebooks/07_metrics.ipynb>

Таблица 2

| Имя модели | text\_vectors | no\_components | epochs | Время обучения | Время формир. предсказаний | intersections | Точность | Полнота |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| lightfm\_svd\_full\_data\_100c\_10e.model | svd\_vectors\_redused.csv | 100 | 10 | 1:30:01 | 15:55:15 | 26213 | 4,539 | 24,094 |
| lightfm\_svd\_full\_data\_50c\_10e.model | svd\_vectors\_redused.csv | 50 | 10 | 0:53:44 | 9:58:19 | 23952 | 4,148 | 22,016 |
| lightfm\_svd\_full\_data\_75c\_10e.model | svd\_vectors\_redused.csv | 75 | 10 | 1:12:49 | 12:59:52 | 25266 | 4,375 | 23,223 |
| lightfm\_svd\_full\_data\_100c\_5e.model | svd\_vectors\_redused.csv | 100 | 5 | 0:54:17 | 15:53:06 | 25028 | 4,334 | 23,005 |
| lightfm\_svd\_full\_data\_100c\_20e.model | svd\_vectors\_redused.csv | 100 | 20 | 2:37:35 | 15:52:30 | 26258 | 4,594 | 24,383 |
| lightfm\_doc2vec\_full\_data\_100c\_10e.model | doc2Vec\_vectors.csv | 100 | 10 | 1:33:48 | 15:52:27 | 18602 | 3,221 | 17,098 |

# 6 Оценка результатов

6.1 Сравнение с результатами участников хакатона

Результаты участников хакатона приведены в таблице 3.

Таблица 3

| № | Команда | Максимальная полнота |
| --- | --- | --- |
| 1 | MarMorRecSys | 33,70 |
| 2 | mlka | 33,27 |
| 3 | DatastaR | 29,41 |
| 4 | Sergeif | 28,94 |
| 5 | Объект 42 | 27,99 |
| 6 | Justice | 25,94 |
| 7 | Московские Зайцы | 25,15 |
| 8 | Melon Elsk | 22,01 |
| 9 | Siberia | 22,00 |
| 10 | Команда А | 21,35 |
| 11 | Stacy | 20,71 |
| 12 | abs | 16,92 |
| 13 | untitled team | 15,08 |
| 14 | hakaton\_me | 11,87 |
| 15 | Sentemel | 6,51 |
| 16 | Боты | 6,39 |
| 17 | Gornyaki | 3,16 |

6.2 Проведем визуальную оценку работы модели

# считываем данные

train\_labels = pd.read\_csv("../data/external/train\_labels.csv", sep=';')

train\_data = pd.read\_csv("../data/external/train\_data.csv", sep=';')

test\_data = pd.read\_csv("../data/external/test\_data.csv", sep=';')

recommendations\_svd = pd.read\_csv("../data/processed/recommendations\_svd.csv", sep=';')

recommendations\_doc2vec = pd.read\_csv("../data/processed/recommendations\_doc2vec.csv", sep=';')

# заполняем пропуски, объединяем окпд2 и дополнительный код, объединяем описания

train\_data = train\_data.fillna('None')

test\_data = test\_data.fillna('None')

train\_data['okpd2\_or\_additional\_code'] = train\_data[['okpd2\_code', 'additional\_code']].apply(lambda x: x[0] if x[1] == 'None' else x[1], axis=1)

test\_data['okpd2\_or\_additional\_code'] = test\_data[['okpd2\_code', 'additional\_code']].apply(lambda x: x[0] if x[1] == 'None' else x[1], axis=1)

train\_data['text\_description\_tender'] = train\_data['purchase\_name'] + " " + train\_data['lot\_name'] + " " + train\_data['okpd2\_names'] + " " + train\_data['additional\_code\_names'] + " " + train\_data['item\_descriptions']

test\_data['text\_description\_tender'] = test\_data['purchase\_name'] + " " + test\_data['lot\_name'] + " " + test\_data['okpd2\_names'] + " " + test\_data['additional\_code\_names'] + " " + test\_data['item\_descriptions']

def show\_lots(user\_id):

'''

функция для показа истории и рекомендаций для участника

'''

history\_ = train\_labels[train\_labels['participant\_inn\_kpp\_anon'] == user\_id].merge(train\_data, how='left', left\_on='pn\_lot\_anon', right\_on='pn\_lot\_anon')[['fz\_x', 'region\_code', 'okpd2\_or\_additional\_code', 'text\_description\_tender']]

recomended\_lots\_1 = recommendations\_svd[recommendations\_svd.inn\_kpp == user\_id].merge(test\_data, how='left', left\_on='actual\_recommended\_pn\_lot', right\_on='pn\_lot\_anon')[['fz', 'region\_code', 'okpd2\_or\_additional\_code', 'text\_description\_tender']]

recomended\_lots\_2 = recommendations\_doc2vec[recommendations\_doc2vec.inn\_kpp == user\_id].merge(test\_data, how='left', left\_on='actual\_recommended\_pn\_lot', right\_on='pn\_lot\_anon')[['fz', 'region\_code', 'okpd2\_or\_additional\_code', 'text\_description\_tender']]

history\_.to\_csv("../data/processed/history.csv", index=False)

recomended\_lots\_1.to\_csv("../data/processed/rec\_1.csv", index=False)

recomended\_lots\_2.to\_csv("../data/processed/rec\_2.csv", index=False)

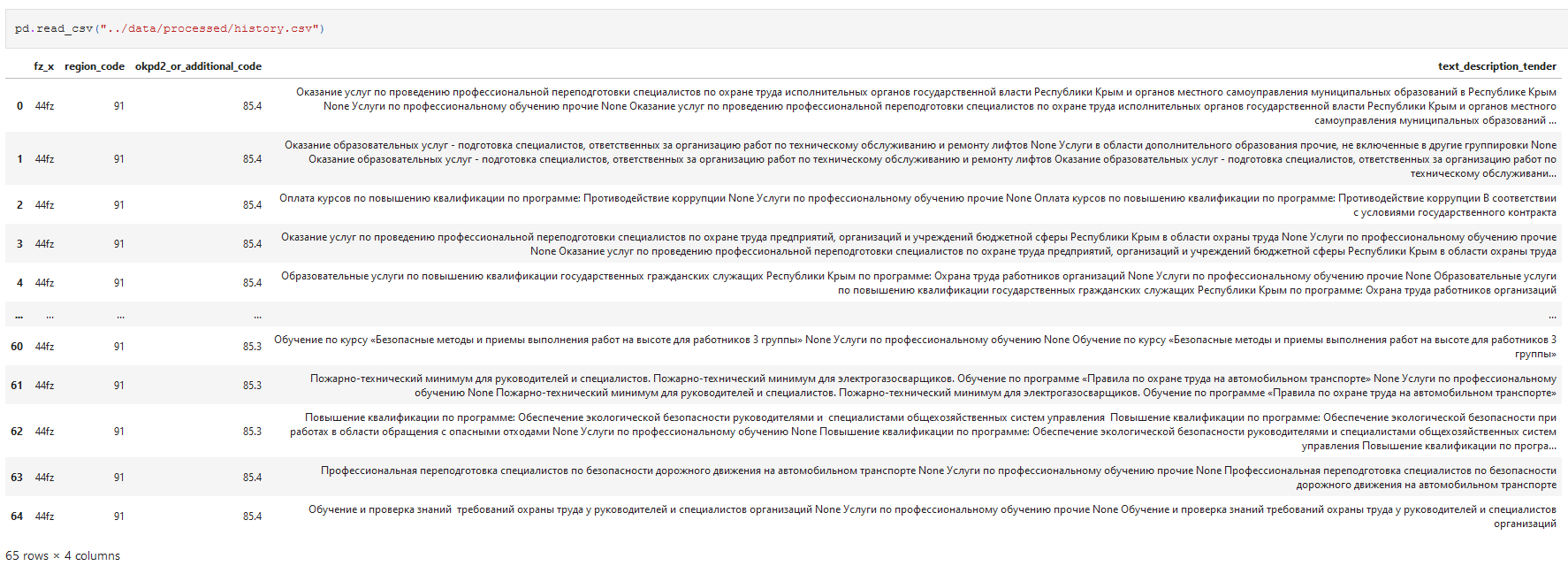
return 'Done'

# рекомендации для случайного участника

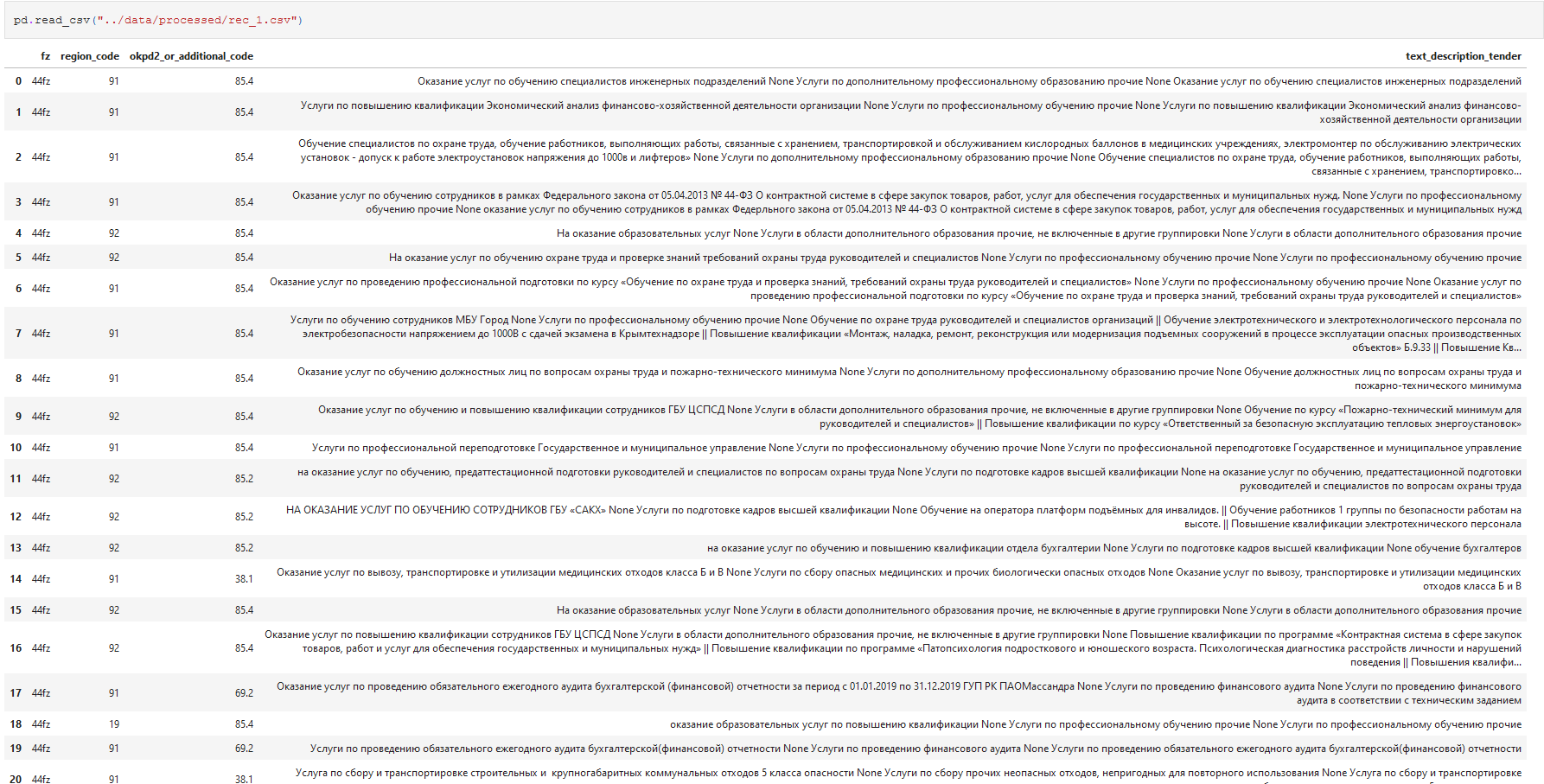
show\_lots(recommendations\_svd.inn\_kpp.sample().values[0])

Ссылка на полный код: <https://github.com/FED32/Diplom_ds/blob/master/Diplom/notebooks/08_Visulisation.ipynb>

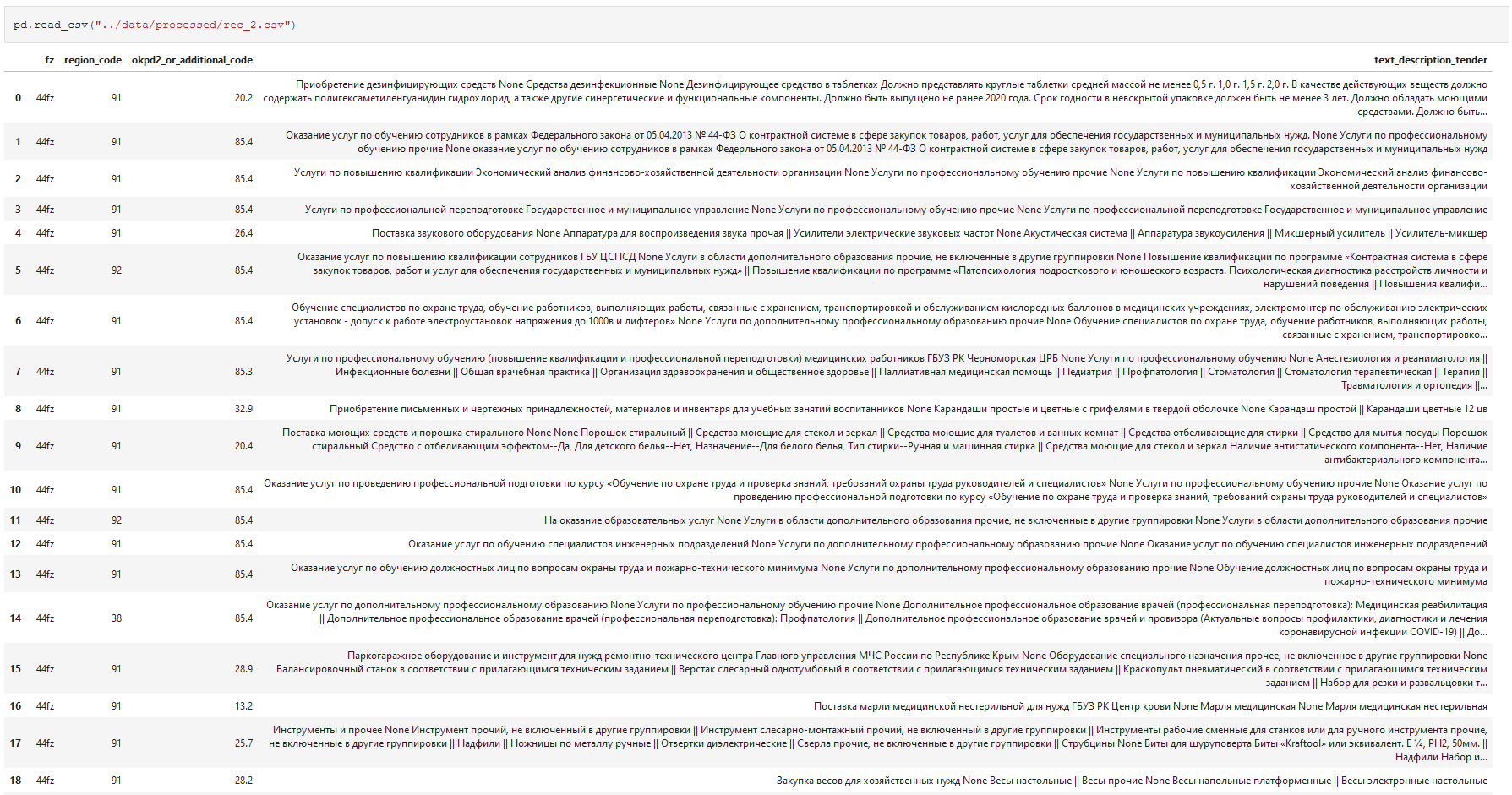
История участий случайного поставщика:



Рекомендации для случайного поставщика (на основе TF-IDF-SVD-описаний):



Рекомендации для случайного поставщика (на основе Doc2Vec модели описаний):



# Заключение

В данной работе была разработана модель рекомендательной системы для рекомендации актуальных процедур закупок поставщикам на основе их истории участия.

Для сравнения, векторизация текстовых описаний была проведена при помощи TF-IDF с последующим снижением размерности методом TruncatedSVD и с использованием модели Doc2Vec. В нашем случае первый способ показал лучшие результаты. Doc2Vec является моделью связанных векторных представлений, что в случае описаний тендеров имеет мало преимуществ, по сравнению с моделью TF-IDF, основанной на «мешке слов».

Модель LightFM на большом объеме данных требовательна к вычислительным ресурсам. На качество обучения в большей степени влияет размерность скрытых вложений признаков (no\_components), при этом страдает производительность.

Код проекта доступен по ссылке <https://github.com/FED32/Diplom_ds>

Список литературы:

1 Metadata Embeddings for User and Item Cold-start Recommendations. Maciej Kula, <https://arxiv.org/abs/1507.08439>

2 <https://www.roseltorg.ru/zakupkihack>

3 Обзор решений победителей Zakupki.Hack. <https://telegra.ph/Obzor-reshenij-pobeditelej-ZakupkiHack-06-29>

4 [https://making.lyst.com/lightfm/docs/home.html#](https://making.lyst.com/lightfm/docs/home.html)

5 <https://building-babylon.net/2016/03/18/warp-loss-for-implicit-feedback-recommendation/>

6 <https://radimrehurek.com/gensim/auto_examples/tutorials/run_doc2vec_lee.html>