














Расположение файлов

 .ipynb_checkpoints	07.06.2020 2:21	Папка с файлами	
 __pycache__	08.06.2020 16:04	Папка с файлами	
 ThirtyMusic	02.06.2020 12:12	Папка с файлами	
 artists.csv	02.06.2020 18:09	Файл Microsoft Ex...	21 201 КБ
 baseline_vectors.npy	08.06.2020 17:39	Файл "NPY"	162 376 КБ
 extraction_to_csv.ipynb	03.06.2020 17:11	Файл "IPYNB"	30 КБ
 item2vec.npy	08.06.2020 19:41	Файл "NPY"	81 188 КБ
 item2vec.py	08.06.2020 16:04	Python File	3 КБ
 models.ipynb	08.06.2020 21:03	Файл "IPYNB"	274 КБ
 playlists.csv	03.06.2020 17:10	Файл Microsoft Ex...	15 536 КБ
 sessions.csv	02.06.2020 23:15	Файл Microsoft Ex...	314 822 КБ
 tracks.csv	02.06.2020 18:10	Файл Microsoft Ex...	135 043 КБ
 users.csv	02.06.2020 18:10	Файл Microsoft Ex...	2 968 КБ

Протокол оценки качества

Метрика

Так как основная задача - векторизовать музыкальных исполнителей так, чтобы *похожие* музыкальные исполнители оказались ближе к другу в векторном пространстве, чем *непохожие*, необходимо определиться с тем, как мы определяем похожесть двух исполнителей. Это можно сделать с помощью множества естественных метрик или их комбинаций, например:

- процент пересечения аудитории,
- жанровая схожесть,
- возраст исполнителя,
- и пр.

Но так как данные векторные представления нужны для решения задачи **Artist Recommendation**, я считаю, что сравнивать *похожесть* двух исполнителей, нужно с точки зрения этой задачи, например с помощью косинусного расстояния между вектором порекомендованного артиста и вектором выбранного, или бинарно - угадан артист или нет.

Поскольку у нас нет данных реальных рекомендаций и выборов пользователей, я предполагаю, что для моделирования выбора пользователем исполнителя **В** наиболее похожего к исполнителю **А** можно использовать пользовательские плейлисты и сессии* прослушивания, предполагая, что в них представлена неслучайная последовательность треков.

Валидация

Валидировать построенные векторные представления я буду на плейлистах и сессиях, убрав повторы исполнителей из этих последовательностей, следующим образом: берем ближайший вектор исполнителя к исполнителю **п** и проверяя верно ли предсказан исполнитель **п+1** или нет. Результат - Ассигасу по плейлисту.

Здесь можно было бы использовать более сложные подходы или ML/DL алгоритмы, с помощью которых мы предсказывали бы исполнителя, но т.к. основной задачей является построение векторных представлений, и в задании описано, что рекомендация в идеале должна сводиться к knn, я считаю, что в этом нет необходимости

Проверка статистической значимости

Полученные значения (список значений Ассигасу по плейлистам) получены на одной и той же тестовой выборке, и распределение этих значений не нормально. Для проверки стат. значимости разности результатов можели я буду использовать односторонний критерий Вилкоксона.

*но это не точно

Модели

Бейзлайн

Берем векторное пространство размера n , где n это количество артистов. Базисом в этом векторном пространстве рассмотрим набор артистов, которые сделаны из данных n артистов, путем удаления пересечений между артистами, кроме пересечений самих с собой. После этого строим матрицу Количество исполнителей \times Количество исполнителей, где на пересечении исполнителей записано отношение количества их общей аудитории к минимуму из количеств слушателей этих артистов. При таком разложении разнотипные артисты у нас *перпендикулярны* -> скалярное произведение равно 0.

$$r(x,y) = (|x \cup y| / \min(|x|, |y|)).$$

Для каждого артиста мы получим разложение по базису этого пространства. Таким образом мы можем использовать скалярное произведение для подсчета косинусного расстояния между векторами.

Проблемы

Нормировка каждого элемента такой матрицы занимает огромное количество времени на моем компьютере, и чтобы сохранить свойство разложения по базису, я нормирую $x_{i,j}$ на количество слушателей артиста i . Чтобы потом можно было использовать эти вектора, мне приходится понижать их размерность.

Идея 1. Матричные разложения

Естественной кажется идея применить стандартные подходы к построению рекомендаций, такие как SVD / ALS разложения. У нас есть история прослушиваний пользователей и их оценки трекам ("love"). Если сматить треки в их авторов, мы можем сделать матрицу Исполнитель-Пользователь, а затем приблизить ее каким-нибудь разложением. А затем из получившегося разложения взять матрицу Исполнителей, с соответствующими им векторами.

Проблемы

- 1. Даже sparse матрица Количество пользователей \times Количество исполнителей не влезает в память :(
- 2. Медленно :(

Идея 2. Эмбединги

Рядом с вышеизложенным способом валидации сразу напрашивается моделька, которая бы предсказывала Исполнителей по контексту (Плейлисту). + было бы здорово, если бы она учитывала, а как часто какие Исполнители встречаются друг с другом, т.е. еще и имела бы какие-то черты SVD - давайте возьмем за основу GloVe!

Проблемы

- 1. Большая sparse матрица Количество исполнителей \times Количество исполнителей
- 2. Медленно

Решение1 - использовать GPU (например реализацию на торче), но тогда возникает та же проблема с sparse матрицей. Быстро решить эту проблему и завести GloVe у меня не удалось :(

Решение2 - использовать word2vec, т.е. в нашем случае cbow и negative sampling для artist2vec задачи.

In [1]:

```
1 # from google.colab import drive
2 # drive.mount('/content/drive')
3 # root = '/content/drive/My Drive/'
```

In [2]:

```
1 # if running locally
2 root = ''
```

```
In [119]: 1 import pandas as pd
2 import numpy as np
3 import random
4
5 import umap
6
7 import ast
8 import urllib
9 from collections import Counter
10
11 import os
12 import tqdm
13 from IPython.display import clear_output
14 import matplotlib.pyplot as plt
15
16 from scipy.sparse import lil_matrix
17 from sklearn.decomposition import TruncatedSVD
18 from sklearn.feature_extraction.text import CountVectorizer
19
20 import bokeh.models as bm, bokeh.plotting as pl
21 from bokeh.io import output_notebook
22
23 import item2vec
24 import torch
25 torch.cuda.current_device()
26 import torch.nn as nn
27 import torch.optim as optim
28
29 import scipy.stats as st
```

```
In [4]: 1 EMBEDDING_DIM = 150
```

1. Считываем треки для мапинга **трек -> автор(ы)**

```
In [78]: 1 tracks = pd.read_csv(root+'tracks.csv')
2 track_to_author = dict(zip(tracks.ID.values, tracks.author_id.values))
3 del tracks
```

D:\Anaconda3\lib\site-packages\IPython\core\interactiveshell.py:3049: DtypeWarning: Columns (1) have mixed types.Specify dtype option on import or set low_memory=False.
interactivity=interactivity, compiler=compiler, result=result)

```
In [79]: 1 playlists = pd.read_csv(root+'playlists.csv')
2 sessions = pd.read_csv(root+'sessions.csv')
3 playlists.tracks_ids = playlists.tracks_ids.apply(lambda x: ast.literal_eval(x))
4 sessions.tracks = sessions.tracks.apply(lambda x: ast.literal_eval(x))
```

```
In [80]: 1 def convert_to_unicode(text):
2     if '%' in text:
3         text = urllib.parse.unquote(text)
4     return text.replace('+', ' ')
5
6 artists = pd.read_csv(root+'artists.csv')
7 artists['name'] = artists['name'].apply(lambda x: convert_to_unicode(x))
8 artists = artists.drop_duplicates(subset=['ID'])
9 id_to_author = dict(zip(artists.ID.values, artists.name.values))
10 author_to_id = dict(zip(artists.name.values, artists.ID.values))
```

```
In [81]: 1 RANDOM_STATE = 40
2 TEST_SIZE = 4000
3 EXTRACTED_SESSIONS = 120000
```

```

In [82]: 1 sessions_sample = sessions.sample(EXTRACTED_SESSIONS, random_state = RANDOM_STATE)
2 test_playlists_idx = playlists.sample(TEST_SIZE, random_state = RANDOM_STATE).index.values
3 test_sessions_idx = sessions_sample.sample(TEST_SIZE, random_state = RANDOM_STATE).index.values
4
5 train_playlists = []
6 test_playlists = []
7
8 full_set = set()
9 train_set = set()
10 test_set = set()
11 users_set = set()
12 artists_counter = {}
13
14
15 def get rid_of_unknown_authors(train_set, test_list):
16     ids_to_pop = []
17     test_set = set()
18     for i, playlist in enumerate(test_list):
19         kek = train_set
20         kek.update(playlist)
21         if len(train_set) != len(kek):
22             ids_to_pop.append(i)
23             continue
24         test_set.update(playlist)
25     for i in ids_to_pop:
26         test_list.pop(i)
27     return test_list, test_set
28
29
30 def extract_train_test(data, tracks_str, test_id,
31                       test_set=test_set, train_set=train_set, full_set=full_set,
32                       train_playlists=train_playlists, test_playlists=test_playlists, users_set=users_set):
33
34     for playlist, index, user_id in zip(data[tracks_str].values, data.index.values, data.user_id.values):
35         users_set.add(user_id)
36         artists_playlist = []
37         for song in playlist:
38             for authors in ast.literal_eval(track_to_author[song]):
39                 artists_playlist.append(authors)
40                 if artists_counter.get(authors):
41                     artists_counter[authors] += 1
42                 else:
43                     artists_counter[authors] = 1
44         if index in test_id:
45             test_playlists.append(artists_playlist)
46             test_set.update(artists_playlist)
47         else:
48             train_playlists.append(artists_playlist)
49             train_set.update(artists_playlist)
50     return train_set, test_set, train_playlists, test_playlists, users_set

```

```

In [83]: 1 train_set, test_set, train_playlists, test_playlists, users_set = extract_train_test(playlists, 'tracks_ids', test_id,
2 train_set, test_set, train_playlists, test_playlists, users_set = extract_train_test(sessions_sample, 'tracks', test_id,

```

```

In [84]: 1 test_playlists, test_set = get rid_of_unknown_authors(train_set, test_playlists)

```

```

In [85]: 1 assert len(test_set) == len(test_set & train_set)

```

```

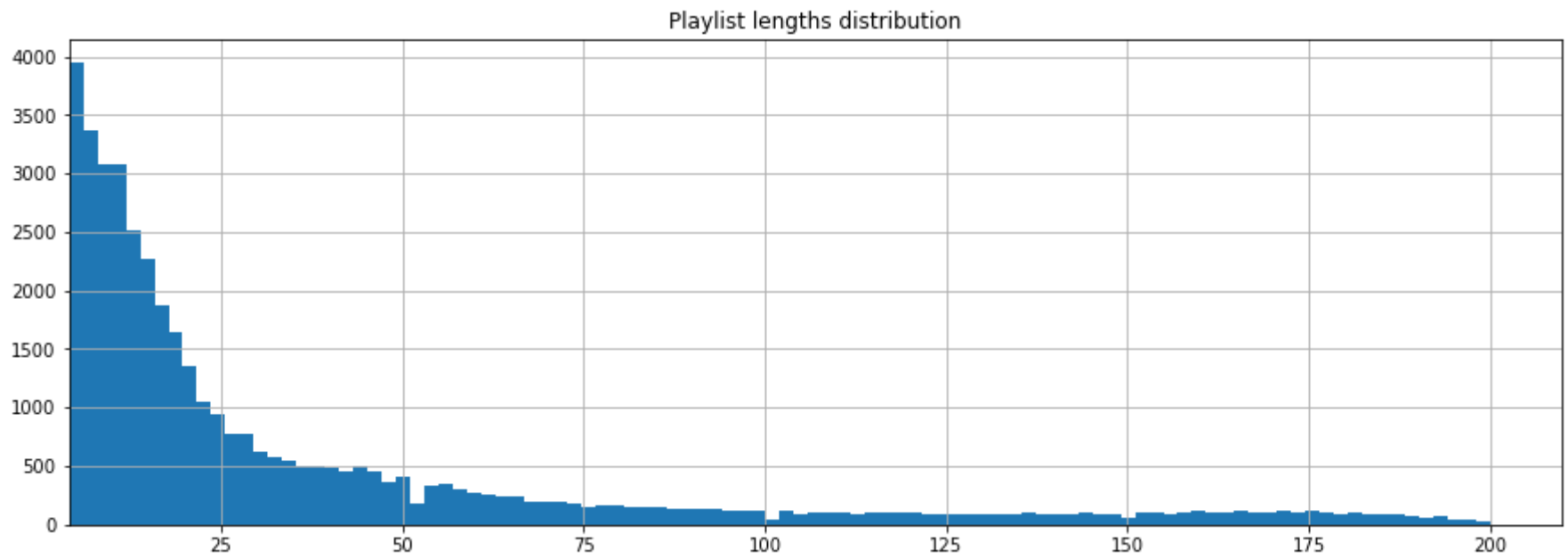
In [86]: 1 print('Всего артистов: ', len(set(author_to_id.values())))
2 print('Артистов в извлеченных плейлистах и сессиях: ', len((train_set | test_set) & set(author_to_id.values())))

```

Всего артистов: 560927

Артистов в извлеченных плейлистах и сессиях: 138560

```
In [87]: 1 plt.figure(figsize=(15, 5))
2 playlists.numtracks.hist(bins=100)
3 plt.title('Playlist lengths distribution')
4 plt.xlim(left=4)
5 plt.show()
```



```
In [88]: 1 print('Средняя длина сессии: ', sessions.numtracks.mean())
2 print('Мода длины сессии: ', sessions.numtracks.mode()[0])
3 print('Максимальная длина сессии: ', sessions.numtracks.max())
4 print()
5 print('Средняя длина плейлиста: ', playlists.numtracks.mean())
6 print('Мода длины плейлиста: ', playlists.numtracks.mode()[0])
7 print('Максимальная длина плейлиста: ', playlists.numtracks.max())
```

Средняя длина сессии: 17.071055122512117

Мода длины сессии: 4

Максимальная длина сессии: 4914

Средняя длина плейлиста: 38.2287082952193

Мода длины плейлиста: 4

Максимальная длина плейлиста: 200

```
In [89]: 1 # technical column
2 artists = artists[artists.ID.isin(test_set|train_set)]
3 artists['new_ID'] = [i for i in range(artists.shape[0])]
4
5 old_id_to_new_id = dict(zip(artists.ID.values, artists.new_ID.values))
6 id_to_author = dict(zip(artists.new_ID.values, artists.name.values))
7 author_to_id = dict(zip(artists.name.values, artists.new_ID.values))
8
9
10 for playlist in train_playlists:
11     for i in range(len(playlist)):
12         playlist[i] = old_id_to_new_id[playlist[i]]
13
14 for playlist in test_playlists:
15     for i in range(len(playlist)):
16         playlist[i] = old_id_to_new_id[playlist[i]]
```

Baseline vectors

```

In [113]: 1 if 'baseline_vectors.npy' in os.listdir():
2         del playlists
3         del sessions
4         del artists
5         result_vecs = np.load('baseline_vectors.npy')
6
7     else:
8         tracks_to_boa = pd.concat((sessions_sample[~sessions_sample.index.isin(test_sessions_idx)][['user_id', 'tracks']]
9
10        # делаем bag of artists для каждого пользователя
11        for i, tracks in enumerate(tracks_to_boa):
12            tracks_to_boa[i] = [old_id_to_new_id[id_] for id_ in list(set(sum([ast.literal_eval(track_to_author[song]) for song in tracks])))
13
14        del playlists
15        del sessions
16        del artists
17
18        boa = lil_matrix((len(users_set), len((train_set|test_set))), dtype=np.int8)
19        for i, artists in enumerate(tracks_to_boa):
20            for artist in artists:
21                boa[i, artist] = 1
22
23        # делаем матрицу совместной встречаемости артистов
24        result_vecs = boa.T @ boa
25
26        del boa
27
28
29        # получившиеся вектора очень большие, нормировка их компонент занимает очень много времени
30        # также матрица предпочитанных косинусных занимает очень много места, и выисляется долго
31
32        # чтобы получить векторы поменьше которые влезли бы в память, понизим размерность
33        svd = TruncatedSVD(n_components=EMBEDDING_DIM, n_iter=2)
34        result_vecs = svd.fit_transform(result_vecs) / result_vecs.diagonal()
35
36        np.save('baseline_vectors.npy', result_vecs)

```

Item2Vec

```

In [18]: 1 total_count = sum([len(play) for play in train_playlists])
2         artists_counter = {artists:count/total_count for artists, count in zip(artists_counter.keys(), artists_counter.values())}

```

```

In [19]: 1 def get_target(words, idx, window_size=5):
2         R = np.random.randint(1, window_size+1)
3         start = idx - R if (idx - R) > 0 else 0
4         stop = idx + R
5         target_words = words[start:idx] + words[idx+1:stop+1]
6         return list(target_words)

```

```

In [20]: 1 def get_batches(words, batch_size, window_size=5):
2         song_count = 0
3         train = []
4         target = []
5         for batch_i, playlist in enumerate(words):
6             if song_count < batch_size or playlist == words[-1]:
7                 x, y = [], []
8                 for ii in range(len(playlist)):
9                     batch_x = playlist[ii]
10                    batch_y = get_target(playlist, ii, window_size)
11                    y.extend(batch_y)
12                    x.extend([batch_x]*len(batch_y))
13                train += x
14                target += y
15                song_count += len(playlist)
16            else:
17                song_count = 0
18                print("Playlist: {}/{}".format(batch_i, len(words)))
19                yield train, target
20                target = []
21                train = []

```

```

In [21]: 1 def cosine_similarity(embedding, sample=None, find_n=False, valid_size=10, valid_window=100, device='cpu'):
2         device = 'cuda' if torch.cuda.is_available() else 'cpu'
3
4         if sample is None:
5             valid_examples = np.array(random.sample(range(valid_window), valid_size//2))
6             valid_examples = np.append(valid_examples,
7                                         random.sample(range(1000,1000+valid_window), valid_size//2))
8             valid_examples = torch.LongTensor(valid_examples).to(device)
9             valid_vectors = embedding(valid_examples)
10            embed_vectors = embedding.weight
11        elif find_n:
12            valid_examples = torch.LongTensor(np.array(sample)).to(device)
13            valid_vectors = embedding(valid_examples)
14            embed_vectors = embedding.weight
15
16        else:
17            valid_examples = torch.LongTensor(np.array(sample)).to(device)
18            valid_vectors = embedding(valid_examples)
19            embed_vectors = valid_vectors
20
21
22        magnitudes = embed_vectors.pow(2).sum(dim=1).sqrt().unsqueeze(0)
23        similarities = torch.mm(valid_vectors, embed_vectors.t())/magnitudes
24        return valid_examples, similarities

```

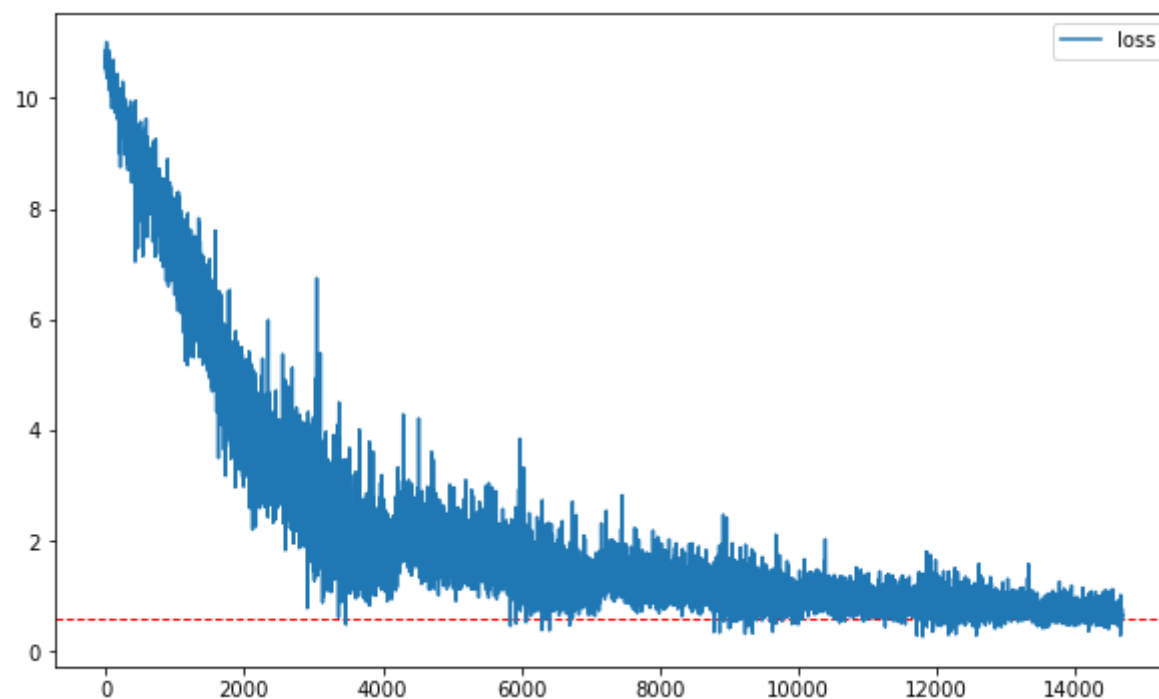
```

In [22]: 1 device = 'cuda' if torch.cuda.is_available() else 'cpu'
2
3 word_freqs = np.array(sorted(artists_counter.values(), reverse=True))
4 unigram_dist = word_freqs/word_freqs.sum()
5 noise_dist = torch.from_numpy(unigram_dist**(.75)/np.sum(unigram_dist**(.75)))
6
7 embedding_dim = EMBEDDING_DIM
8 model = item2vec.SkipGramNeg(len(train_set|test_set), embedding_dim).to(device)
9
10 criterion = item2vec.NegativeSamplingLoss()
11 optimizer = optim.Adam(model.parameters(), lr=0.002)
12 epochs = 5
13

```


In [23]:

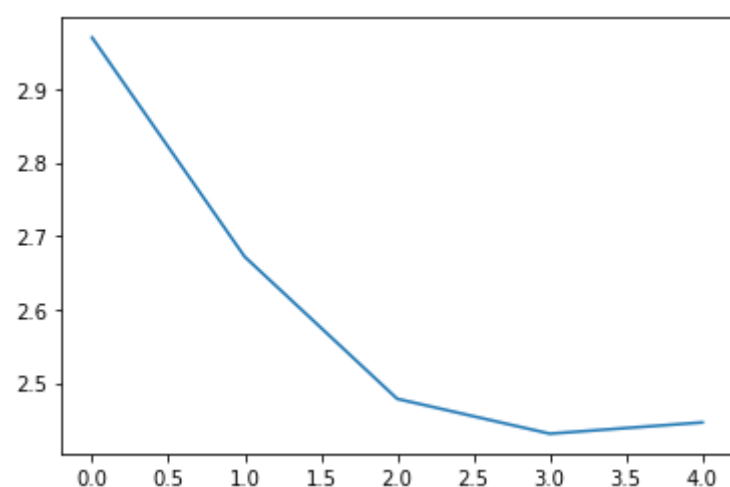
```
1 history = []
2 test_history = []
3 for e in range(epochs):
4     for input_words, target_words in get_batches(train_playlists, 1024):
5         inputs, targets = torch.LongTensor(input_words), torch.LongTensor(target_words)
6         inputs, targets = inputs.to(device), targets.to(device)
7
8         input_vectors = model.forward_input(inputs)
9         output_vectors = model.forward_output(targets)
10        noise_vectors = model.forward_noise(inputs.shape[0], 5)
11
12        loss = criterion(input_vectors, output_vectors, noise_vectors)
13
14        optimizer.zero_grad()
15        loss.backward()
16        optimizer.step()
17
18        print("Epoch: {}/{}".format(e+1, epochs), loss.item())
19        history.append(loss.data.cpu().numpy())
20
21        clear_output(True)
22        plt.figure(figsize=(10,6))
23        plt.axhline(y=history[-1], ls='--', linewidth=1, color='r')
24        plt.plot(history, label='loss')
25        plt.legend()
26        plt.show()
27
28        valid_examples, valid_similarities = cosine_similarity(model.in_embed, device=device)
29        _, closest_idxxs = valid_similarities.topk(5)
30
31        valid_examples, closest_idxxs = valid_examples.to('cpu'), closest_idxxs.to('cpu')
32        for ii, valid_idx in enumerate(valid_examples):
33            closest_words = [id_to_author[idx.item()] for idx in closest_idxxs[ii][1:]]
34            print(id_to_author[valid_idx.item()] + " | " + ', '.join(closest_words))
35        print("...\n")
36    np.save('item2vec.npy', np.array(model.in_embed.weight.cpu().detach().numpy()))
37    test_history.append(np.nanmean(np.array([np.linalg.norm(cosine_similarity(model.in_embed, sample=playlist)[1].cpu().numpy() for playlist in test_playlists])))
```



"Chain of Strength" | "Tolis Voskopoulos", "Foie Gras", "A Girl Called Eddy", "Six Cents and Natalie"
"Saint Cava" | "Samurai Champloo", "Fat Tulips", "Raul Seixas", "Saint Etienne Daho"
"Little Jimmy Scott" | "Madeleine Peyroux & William Galison", "B Bumble & The Stingers", "Kel Assouf", "Cal Tjader & Carmen McRae"
"Mortem" | "J.A.M.", "London After Midnight", "I:Scintilla", "Mr. Boogie Woogie"
"Chris Brown feat. Rihanna" | "JennyAnyKind", "Blue Peter", "Rose Rovine e Amanti", "Neck"
"Dungeons" | "Krinjah", "Manu Gavassi", "Makeup and Vanity Set", "Vertical Church Band"
"Group X" | "Restoring Poetry in Music", "Nother", "Artento Divini", "Spheric Universe Experience"
"Dubledge" | "Hammercult", "Breaking Pangaea", "Agent Ribbons", "Mongol Horde"
"Aghast View" | "Manny Marc, Corus 86 & DJ Reckless", "Gregg Anthe", "The Brendan Hines", "The Asteroids Galaxy Tour"
"Anbu" | "Swarms ft. Holly Prothman", "Easy M", "Choir of Kings College, Cambridge", "Winter Severity Index"
...


```
In [24]: 1 del train_playlists
2 # тестовое внутри-плейлистное косинусное расстояние
3 plt.plot(test_history)
4 # наблюдаем оверфит
```

Out[24]: [matplotlib.lines.Line2D at 0x22997c37cf8>]



```
In [25]: 1
2 output_notebook()
3
4 def draw_vectors(x, y, radius=10, alpha=0.25, color='blue',
5                 width=600, height=400, show=True, **kwargs):
6     """ draws an interactive plot for data points with auxiliary info on hover """
7     if isinstance(color, str): color = [color] * len(x)
8     data_source = bm.ColumnDataSource({ 'x' : x, 'y' : y, 'color': color, **kwargs })
9
10    fig = pl.figure(active_scroll='wheel_zoom', width=width, height=height)
11    fig.scatter('x', 'y', size=radius, color='color', alpha=alpha, source=data_source)
12
13    fig.add_tools(bm.HoverTool(tooltips=[(key, "@" + key) for key in kwargs.keys()]))
14    if show: pl.show(fig)
15    return fig
```

(<http://localhost:5041/surge>) successfully loaded.

```
In [26]: 1 embedding = umap.UMAP(n_neighbors=5).fit_transform(model.in_embed.weight.cpu().detach().numpy())
```

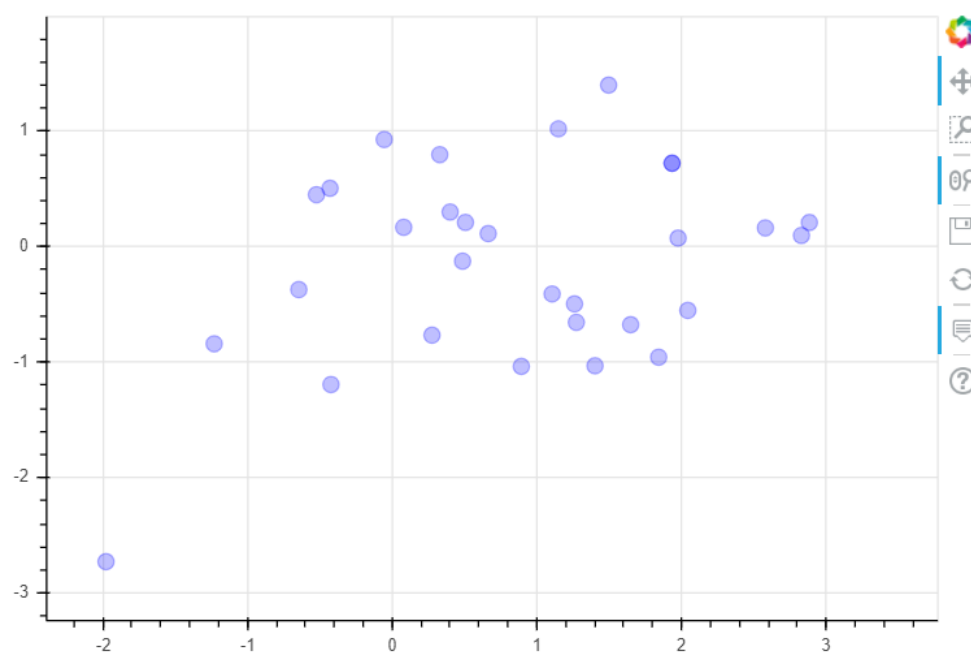
```
In [69]: 1 # Нарисуем Кино, Фрэнка Синатру и 50 cent
2 valid_examples, valid_similarities = cosine_similarity(model.in_embed, sample=[6671, 3373, 161], find_n = True)
3 _, closest_idx = valid_similarities.topk(10)
```

```
In [70]: 1 closest_idx = closest_idx.cpu().numpy().flatten()
2 tokens = [id_to_author[id_] for id_ in closest_idx]
```

```
In [71]: 1 tokens
```

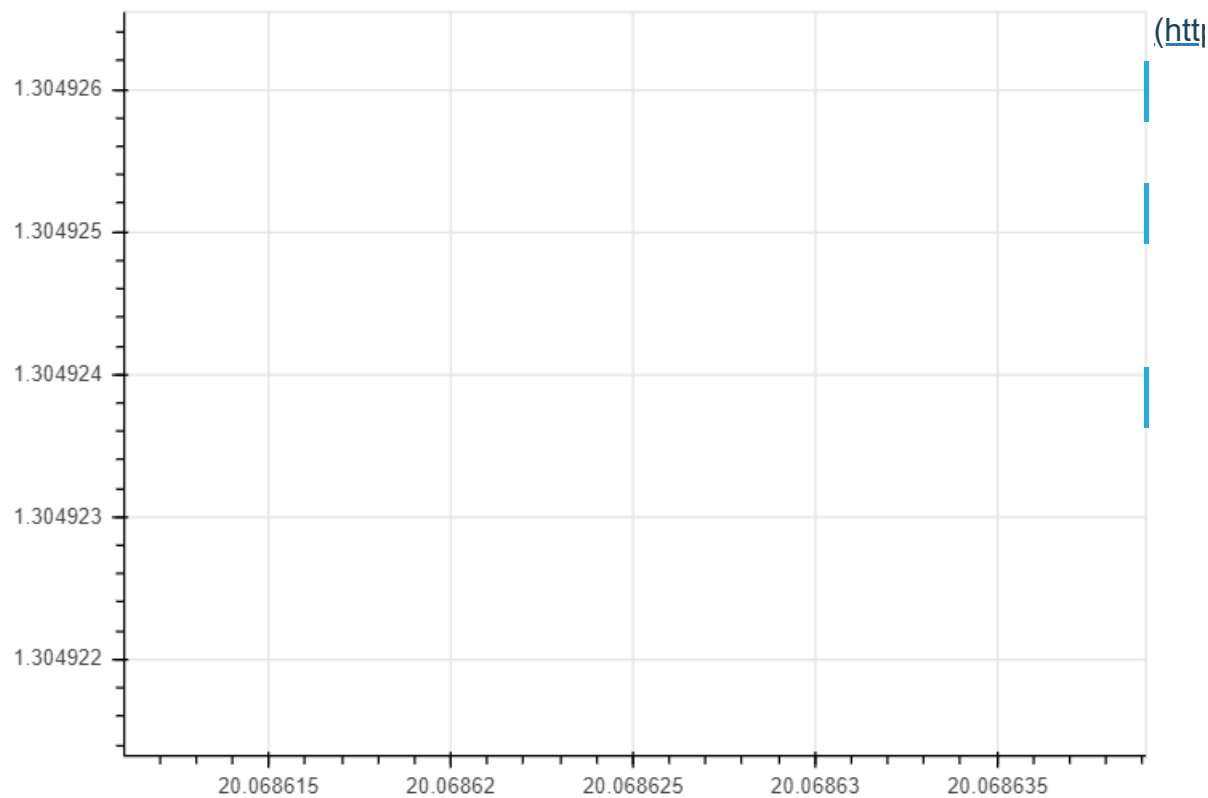
```
Out[71]: ["Группа Кино",  
         "Ночные Снайперы",  
         "ДДТ",  
         "In Strict Confidence",  
         "Земфира",  
         "Би-2 & Brainstorm",  
         "Пилот",  
         "Splin",  
         "Stare Dobre Małżeństwo",  
         "Him & Her",  
         "Киш",  
         "Rammstein",  
         "Пилот",  
         "Marilyn Manson & Sneaker Pimps",  
         "Pain",  
         "Slipknot",  
         "Kittie",  
         "Austrian Death Machine",  
         "Ляпис Трубецкой & Грув",  
         "Агата Кристи",  
         "Frank Sinatra & Tanya Tucker",  
         "Nat King Cole & Dean Martin",  
         "Dean Martin With Nelson Riddle & His Orchestra",  
         "Louis Armstrong & Ella Fitzgerald with Sy Oliver's Orchestra",  
         "Bing Crosby & Bob Hope",  
         "Ella Fitzgerald & The Four Hits And A Miss",  
         "Elvis Presley & Martina McBride",  
         "Peggy Lee",  
         "Brenda Lee",  
         "Ray Charles & George Michael"]
```

```
In [73]: 1 # здесь визуализация  
2 draw_vectors(embedding[:, 0][closest_idxs], embedding[:, 1][closest_idxs], token=tokens)
```



```
Out[73]: Figure(id = '1230', ...)
```

```
In [73]: 1 # здесь визуализация
2 draw_vectors(embedding[:, 0][closest_idxs], embedding[:, 1][closest_idxs], token=tokens)
```



Out[73]: Figure(id = '1230', ...)

Мы видим, что кластеризуются исполнители плоховато, но ближайшие соседи не такие уж и плохие

Группа Кино - преимущественно русскоязычные исполнители (на размере эмбедингов 300 результат еще лучше)

```
print("...\n")
↳ "Gruppa+Kino" | "Splin", "DDT", "Пикник", "Аквариум & Дживан Гаспарян", "Чиж & Со", "Земфира", "Мумий Троль", "Ночные Снайперы", "Алиса"
...
```

```
In [33]: 1 valid_examples, valid_similarities = cosine_similarity(model.in_embed, sample=[6671], find_n=True)
2 _, closest_idxs = valid_similarities.topk(10)
3 closest_idxs = closest_idxs.cpu().tolist()[0]
4 closest_words = [id_to_author[idx] for idx in closest_idxs[1:]]
5 print(id_to_author[6671] + " | " + ', '.join(closest_words))
6
```

"Gruppa Kino" | "Ночные Снайперы", "DDT", "In Strict Confidence", "Земфира", "Би-2 & Brainstorm", "Пилот", "Splin", "Stare Dobre Mażeństwo", "Him & Her"

Классика зарубежного металла

```
In [52]: 1 valid_examples, valid_similarities = cosine_similarity(model.in_embed, sample=[1014], find_n=True)
2 _, closest_idxs = valid_similarities.topk(10)
3 closest_idxs = closest_idxs.cpu().tolist()[0]
4 closest_words = [id_to_author[idx] for idx in closest_idxs[1:]]
5 print(id_to_author[1014] + " | " + ', '.join(closest_words))
```

"Metallica, Michael Kamen & San Francisco Symphony" | "Iron Maiden & Dream Theater", "Guns 'N' Roses", "AC-DC", "Megadeth", "Deep Purple & KISS", "Pantera", "Scorpions - Moment Of Glory", "Judas Priest", "Aerosmith"

Рэп

```
In [55]: 1 valid_examples, valid_similarities = cosine_similarity(model.in_embed, sample=[588], find_n=True)
2 _, closest_idxs = valid_similarities.topk(10)
3 closest_idxs = closest_idxs.cpu().tolist()[0]
4 closest_words = [id_to_author[idx] for idx in closest_idxs[1:]]
5 print(id_to_author[588] + " | " + ', '.join(closest_words))
6
```

"50 Cent feat. Snoop Dogg & Pre" | "Jay-Z & Kanye West", "Snoop Dogg feat. The Doors", "Eminem feat. Nate Dogg", "Kanye West ft. Frank Ocean", "T.I.", "Friday/Dr. Dre", "Lil Wayne feat. Drake & Rick Ross", "Drake & Coldplay", "DMX & Rakim Ft Shontelle & Aleks D"

Validation

```
In [74]: 1 item2vec = np.load('item2vec.npy')
```

```
In [114]: 1 def get_error(id_, real_vec, method, verbose=False):
2         if method == 'baseline':
3
4             similarities = (result_vecs[id_, :] @ result_vecs.T)/magnitudes_base
5
6             nonzero = np.nonzero((similarities / magnitudes_base[id_]) < (similarities[id_] / magnitudes_base[id_]))
7
8             if len(nonzero[0]) > 0 :
9                 id_in_nonzero = np.argmax(similarities[nonzero])
10
11             if verbose:
12                 print()
13                 print('BASELINE')
14                 print('Predicted - ', id_to_author[id_in_nonzero])
15                 print()
16
17             return int(real_vec == id_in_nonzero)
18         return 0
19
20     else:
21         similarities = (item2vec[id_, :] @ item2vec.T)/magnitudes_item
22
23         nonzero = np.nonzero((similarities / magnitudes_item[id_]) < (similarities[id_] / magnitudes_item[id_]))
24         id_in_nonzero = np.argmax(similarities[nonzero])
25
26         if verbose:
27             print('ITEM2VEC')
28             print('Predicted - ', id_to_author[id_in_nonzero])
29             print()
30
31         return int(real_vec == id_in_nonzero)
32
```

```
In [115]: 1 method_1 = []
2 method_2 = []
3
4 magnitudes_base = np.sqrt(np.power(result_vecs, 2).sum(axis=1))
5 magnitudes_item = np.sqrt(np.power(item2vec, 2).sum(axis=1))
6
7 for playlist in tqdm.tqdm_notebook(test_playlists):
8     m1 = []
9     m2 = []
10    playlist = list(dict.fromkeys(playlist).keys())
11    if len(playlist) > 1:
12        for i in range(len(playlist)-1):
13
14            # print('Current - ', id_to_author[playlist[i]])
15            # print('Next - ', id_to_author[playlist[i+1]])
16            m1.append(get_error(playlist[i], playlist[i+1], method='baseline', verbose=False))
17            m2.append(get_error(playlist[i], playlist[i+1], method='i2w', verbose=False))
18
19    method_1.append(sum(m1)/len(m1))
20    method_2.append(sum(m2)/len(m2))
```

D:\Anaconda3\lib\site-packages\ipykernel_launcher.py:7: TqdmDeprecationWarning: This function will be removed in tqdm==5.0.0

Please use `tqdm.notebook.tqdm` instead of `tqdm.tqdm_notebook`
import sys

100% 8000/8000 [48:38<00:00, 2.74it/s]

D:\Anaconda3\lib\site-packages\ipykernel_launcher.py:4: RuntimeWarning: invalid value encountered in true_divide
after removing the cwd from sys.path.

D:\Anaconda3\lib\site-packages\ipykernel_launcher.py:6: RuntimeWarning: invalid value encountered in less

D:\Anaconda3\lib\site-packages\ipykernel_launcher.py:6: RuntimeWarning: invalid value encountered in true_divide

```
In [116]: 1 print('Среднее случайных предсказаний :', 1 / item2vec.shape[0])
```

Среднее случайных предсказаний : 7.2170900692840645e-06

```
In [120]: 1 print('Метод baseline')
2 print('Среднее :', np.mean(method_1))
3 print('Границы 95% дов. интервала :', st.t.interval(0.95, len(method_1), loc=np.mean(method_1), scale=st.sem(method_1)))
```

Метод baseline

Среднее : 8.82312433288572e-05

Границы 95% дов. интервала : (-2.503249972483828e-05, 0.00020149498638255268)

Среднее случайных предсказаний попадает в 95% доверительный интервал baseline

```
In [121]: 1 print('Метод item2vec')
          2 print('Среднее :', np.mean(method_2))
          3 print('Границы 95% дов. интервала :', st.t.interval(0.95, len(method_2), loc=np.mean(method_2), scale=st.sem(method_2)))
```

Метод item2vec
Среднее : 0.004886978726047396
Границы 95% дов. интервала : (0.003949691725504531, 0.0058242657265902605)

Т.к. выборки зависимы, проверить гипотезу о положительной разности средних предсказаний метода два и метода один можно с помощью одностороннего критерия Вилкоксона

H0: Разница между выборками нулевая H1: Разница между выборками больше нуля

```
In [122]: 1 print(st.wilcoxon(np.array(method_2) - np.array(method_1)))
```

WilcoxonResult(statistic=717.5, pvalue=1.820358424796903e-64)

Нулевая гипотеза отвергается в пользу альтернативы на достигаемом уровне значимости 5%

Выводы

Полученные векторные представления с помощью item2vec подхода получились гораздо лучше "глупого" бейзлайна. Толковый подбор гиперпараметров сделать не удалось, т.к. когда размер векторов >200 мне не хватает памяти на моем компьютере. А Google Collaby не хватает памяти чтобы загрузить все вектора себе в память.

Из того что удалось понять - вектора длиной 300 лучше чем 200 и 150. Вектора длиной 300 переобучаются на 5ой эпохе. Learning rate 0.003 - 0.005 для старта оптимальный.

Наверное лучшим решением такой задачи будет SVD по матрице пользователь-артист. Но в таком решении, как и в моем, существует проблема **холодных** артистов, которых еще никто не слушал. Сейчас такие артисты просто выбрасываются из теста, если их не было в трейне.

Вектора SVD таких артистов можно приближать с помощью нейронки, которая на вход получала бы какую-нибудь информацию о артистах или их песнях (мел разложение например), и предсказывала бы вектор item2веса или svd-разложения.

Неиспользованные данные 'events' и 'love' можно было бы применить для разложения. А данные 'albums' и 'tags' для приближения векторов холодных артистов

```
In [ ]: 1
```