NYU FRE 7773 - Week 13

Machine Learning in Financial Engineering

Jacopo Tagliabue

Evaluation time
(sorry, need to do it)!

RecSys part II

Machine Learning in Financial Engineering

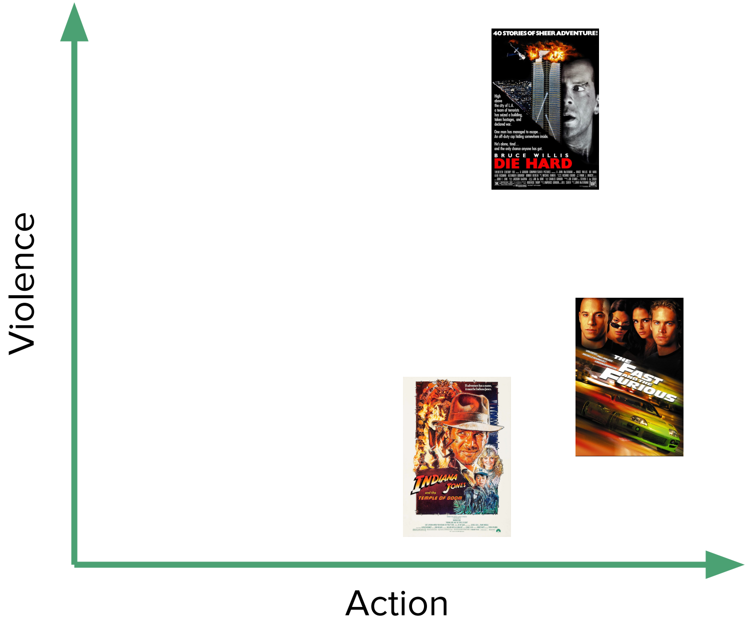
Jacopo Tagliabue

Recap on embeddings

Intuition: a (basic) recSys is like a GPS navigator

Consider a movie recommendation systems (user-item case)

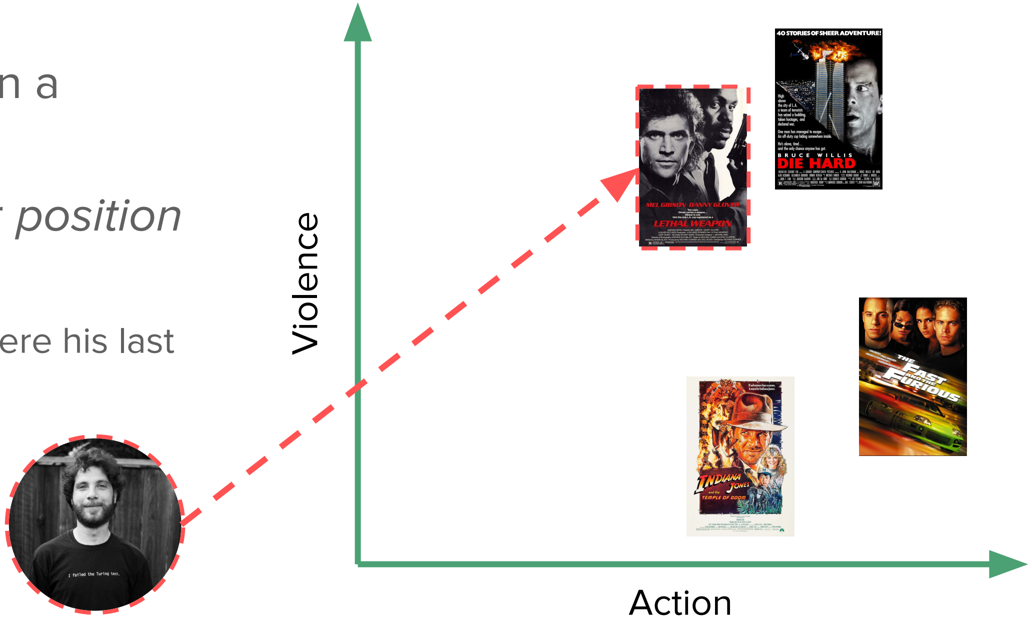
- **Step 1**: represent movies in a suitable space


40 STORIES OF SHEER ADVENTURE!
BRUCE WILLIS
DIE HARD
INDIANA JONES and the TEMPLE OF DOOM
THE FAST AND THE FURIOUS
Violence
Action

Intuition: a (basic) recSys is like a GPS navigator

Consider a movie recommendation systems (user-item case)

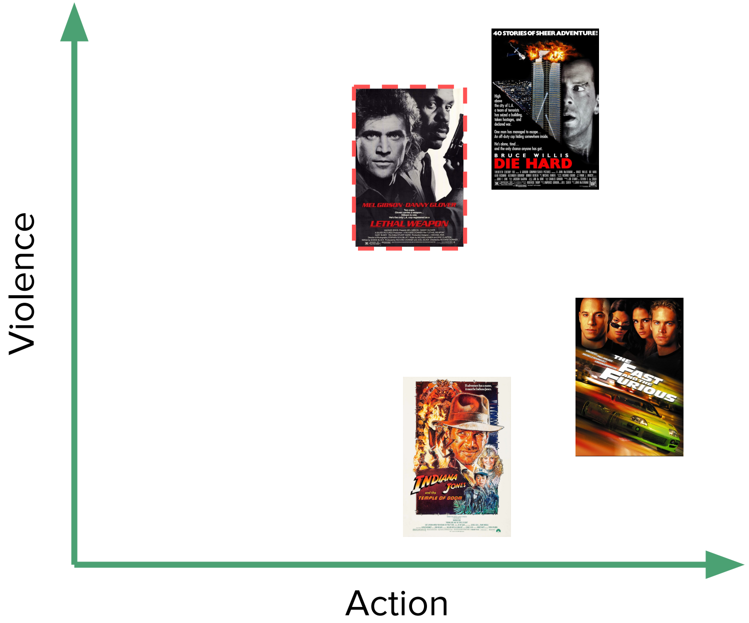
- **Step 1**: represent movies in a suitable space
- **Step 2**: represent the user *position* in the space
  - For example, Jacopo is “where his last movie is”



Intuition: a (basic) recSys is like a GPS navigator

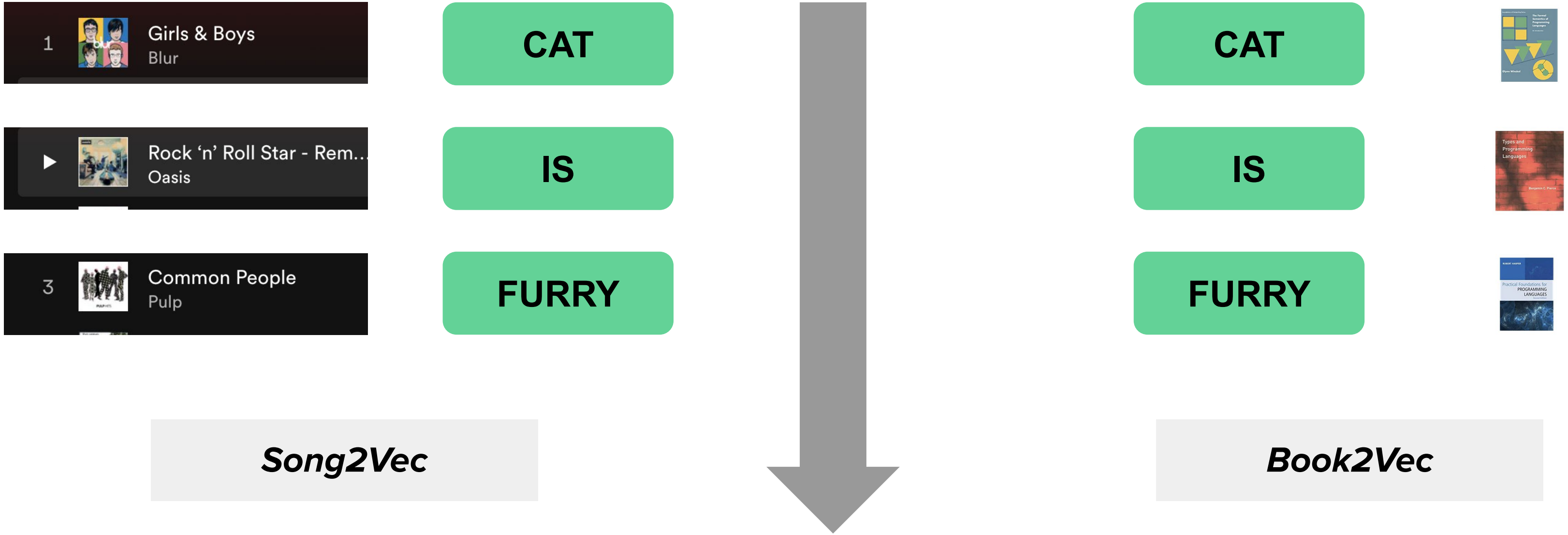
Consider a movie recommendation systems (user-item case)

- **Step 1**: represent movies in a suitable space
- **Step 2**: represent the user *position* in the space
- **Step 3**: recommend the closest K items (KNN search) to the user!
  - Recommendation: Die Hard!



Word2Vec, Song2Vec, Everything2Vec

- **Word2Vec** builds embeddings for words, leveraging their vicinity in a sequence.
- The same intuition can be applied whenever we have meaningful sequences of target items (e.g. playlist, shopping sessions etc.)



1 Girls & Boys
Blur

Rock 'n' Roll Star - Rem...
Oasis

3 Common People
Pulp

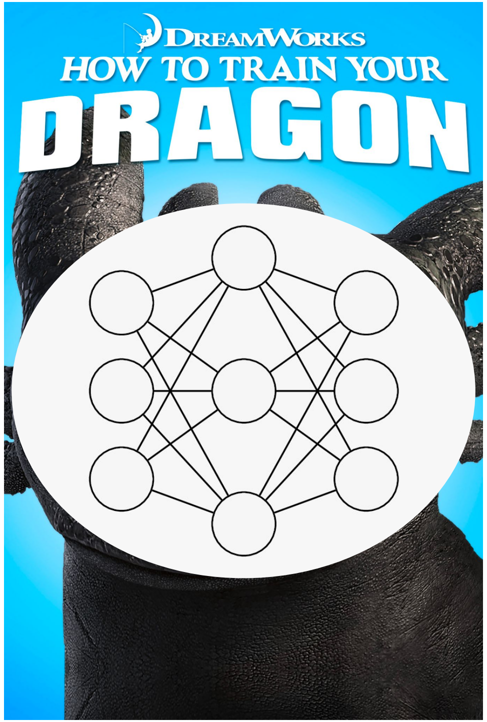
Song2Vec

Book2Vec

Building a song recommender

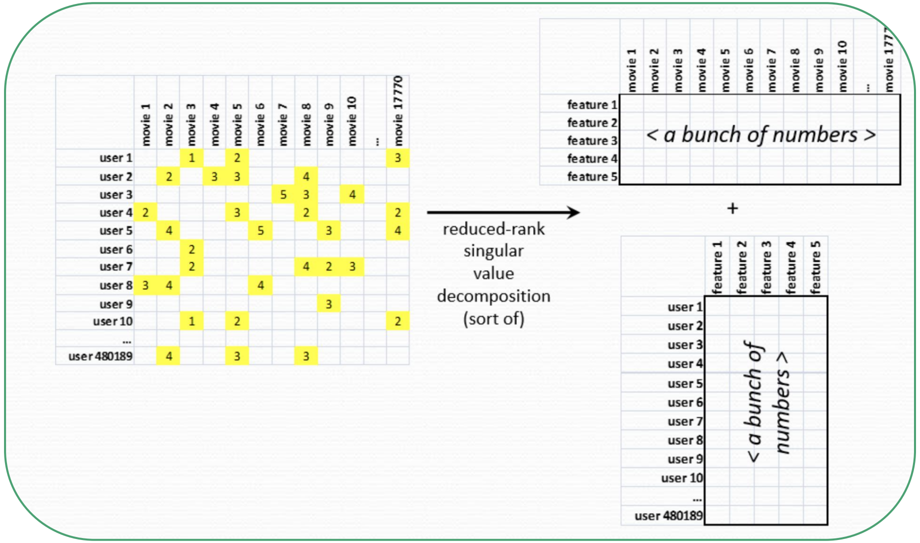
RecSys Data, revisited

- Input ("features"): as we discussed we divide them typically in **content-based** vs **behavioral-based**.
- Target: typically two types, ratings (stars on a movie) and interactions (click on ads, buying shoes):
    - Reviews are *explicit* (people tell you when you suck!), interaction is *implicit* (people buy stuff they like, but don't tell you what they don't!). The target will influence your loss function, typically mean squared error for ratings, logistic loss for binary.
    - As many (all?) hard ML problems, recSys are fundamentally a challenge in sparsity: in the Netflix challenge for example, the users / movies matrix has **100 million / ~8.5 billion ratings**.



Once upon a time: collaborative filtering

- **Use case:** user-item, Netflix prize
- **Intuition**: we can “reduce” the original matrix (sparse) to the combination of two smaller matrices, users/movies
    - Instead of listing all my ratings, we could say I like sci-fi and Tom Cruise movies
- **Prediction**: the dot product between user and item vectors is the estimate of the rating!
- **Training**: $\min_{p,q} \sum_{(u,i)\in\kappa} (r_{ui} - q_i^T p_u)^2 + \lambda(||q_i||^2 + ||p_u||^2)$



	movie 1	movie 2	movie 3	movie 4	movie 5	movie 6	movie 7	movie 8	movie 9	movie 10	...	movie 17770
user 1			1		2							3
user 2		2		3	3			4				
user 3							5	3		4		
user 4	2				3			2				2
user 5		4				5			3			4
user 6			2									
user 7			2					4	2	3		
user 8	3	4				4						
user 9									3			
user 10			1		2							2
...												
user 480189		4			3			3				

Decomposing the matrix into user and movie features

Q: wait a sec, what about cold start?

Once upon a time: collaborative filtering

- **Collaborative filtering** is great, but its performance crucially depends on behavioral data. Cold-start scenarios are not easily addressed in the original problem formulation!
- While you can think of fixing the problem in clever ways (for example, you can ask "few good users" to help you out by explicitly asking them initial ratings), the underlying "bug" of pure matrix factorization is the lack of content-based features. If the only type of data the model is using is interactions / ratings / purchases, we are missing the chance of leveraging important meta-data about our use case!
- While there are pure matrix-based methods that can help, more modern systems leverage deep learning and its inherent flexibility.

Song2Vec

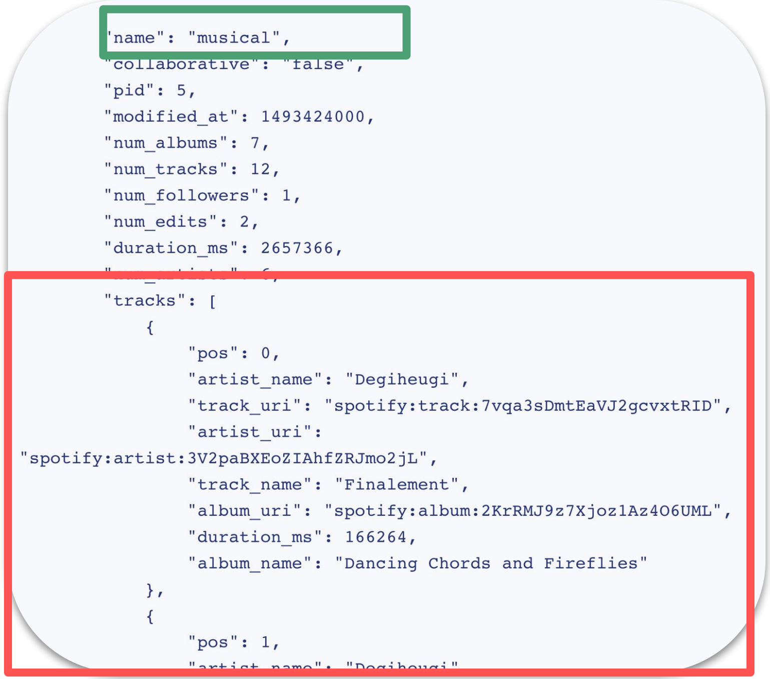
We build a Metaflow flow to recommend new songs: for example, you are listening to “Hey Jude” and want to know what to listen next, or you’re building a playlist and want to know what to add.

We need:

- Initial dataset
- Modelling assumption
- Train and test preparation
- The evaluation

The dataset

- Initial dataset: the Playlist Dataset from Spotify
  - Each playlist has a name and an array of tracks.
  - Each track contains basic metadata and a track id.



```
'name": "musical",
"collaborative": "false",
"pid": 5,
"modified_at": 1493424000,
"num_albums": 7,
"num_tracks": 12,
"num_followers": 1,
"num_edits": 2,
"duration_ms": 2657366,
"num_artists": 6,
"tracks": [
    {
        "pos": 0,
        "artist_name": "Degiheugi",
        "track_uri": "spotify:track:7vqa3sDmtEaVJ2gcvxtRID",
        "artist_uri":
"spotify:artist:3V2paBXEoZIAhfZRJmo2jL",
        "track_name": "Finalement",
        "album_uri": "spotify:album:2KrRMJ9z7Xjoz1Az4O6UML",
        "duration_ms": 166264,
        "album_name": "Dancing Chords and Fireflies"
    },
    {
        "pos": 1,
        "artist_name": "Degiheugi"
```

Introducing The Million Playlist Dataset and RecSys Challenge 2018

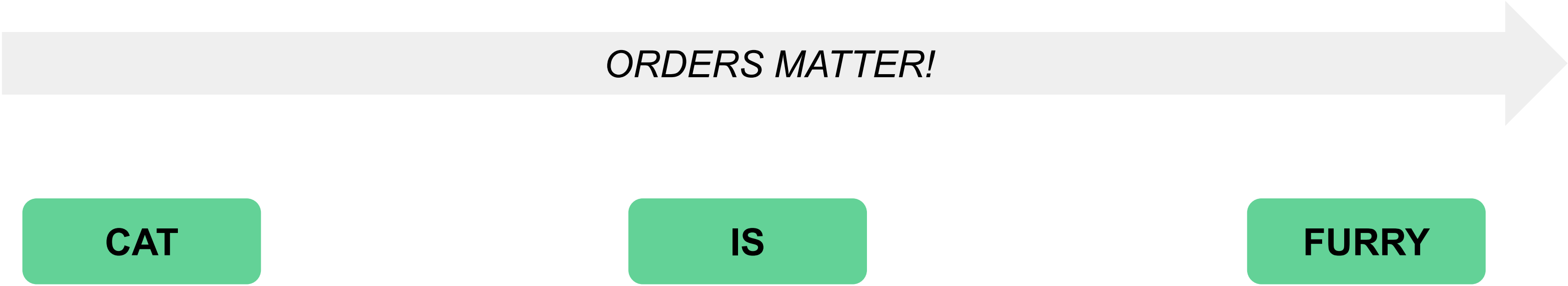
May 30, 2018
Published by Ching-Wei Chen

RecSys Challenge 2018

Welcome ACM RecSys Community! For this year's challenge, use the Spotify Million Playlist Dataset to help users create and extend their own playlists.

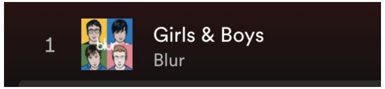
Modelling assumptions

- **Word2Vec** = "**words** that appear together in **sentences** are similar".
- **Song2Vec** = "**songs** that appear together in **playlists** are similar".
- *We can apply Word2Vec to songs*!

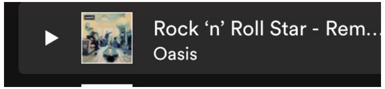


ORDERS MATTER!

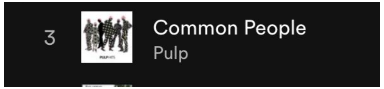
CAT **IS** **FURRY**



1 Girls & Boys
Blur



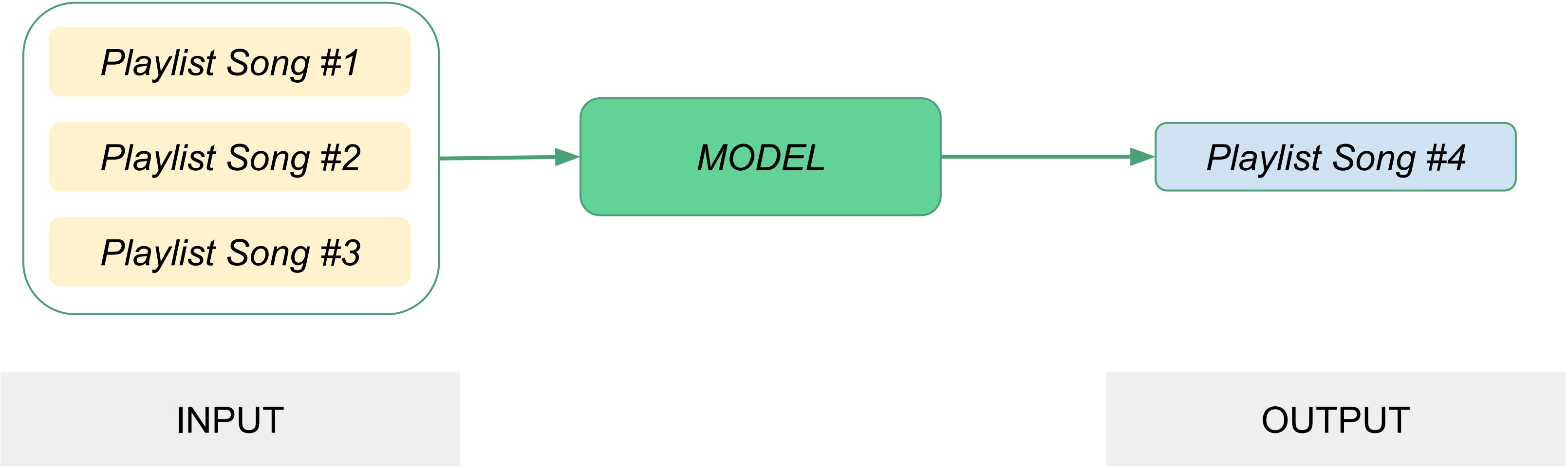
Rock 'n' Roll Star - Rem...
Oasis



3 Common People
Pulp

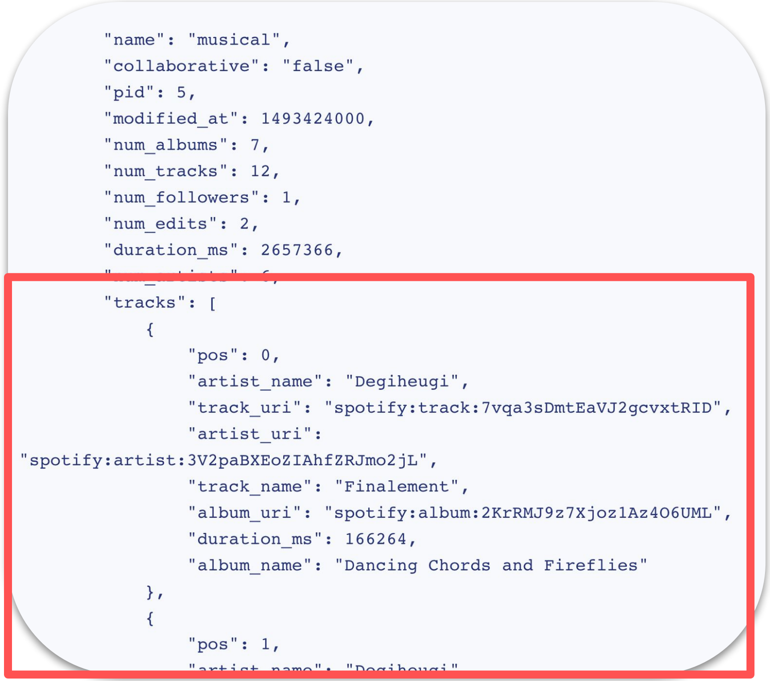
Modelling assumptions

- **Recommendation / Prediction**
  - **Input:** a list (possibly n=1) of songs in a playlist / user history
  - **Output:** predict the next song to listen / continue the playlist



Data preparation

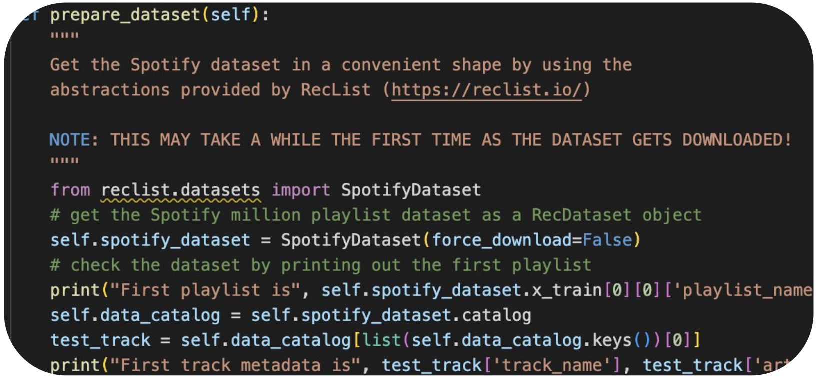
- We divide track data from metadata instead of replicating track metadata every time a track is found:
  - Playlists -> sequences of track Ids -> [“fasfa”, “fauofu”, “earfao”]
  - Catalog -> a mapping between track Ids and metadata ->
    - “fauofu”: { “name”: “Hey Jude” }
    - “earfao”: { “name”: “Imagine” }



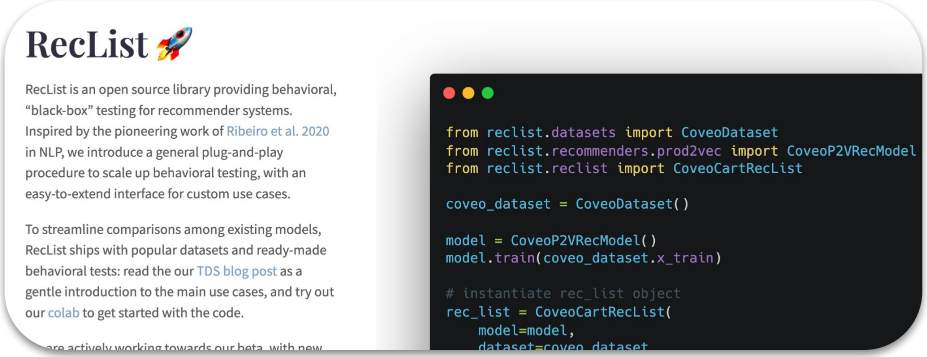
```
        "name": "musical",
        "collaborative": "false",
        "pid": 5,
        "modified_at": 1493424000,
        "num_albums": 7,
        "num_tracks": 12,
        "num_followers": 1,
        "num_edits": 2,
        "duration_ms": 2657366,
        "num_artists": 6,
        "tracks": [
            {
                "pos": 0,
                "artist_name": "Degiheugi",
                "track_uri": "spotify:track:7vqa3sDmtEaVJ2gcvxtRID",
                "artist_uri":
"spotify:artist:3V2paBXEoZIAhfZRJmo2jL",
                "track_name": "Finalement",
                "album_uri": "spotify:album:2KrRMJ9z7Xjoz1Az4O6UML",
                "duration_ms": 166264,
                "album_name": "Dancing Chords and Fireflies"
            },
            {
                "pos": 1,
                "artist_name": "Degiheugi",
```

Data preparation

- Instead of doing it manually, we rely on the dataset prepared by RecList, an open source library for evaluating Recommender Systems.
    - Using RecList we get a training set and data_catalog ready to use!



```
prepare_dataset(self):
"""
Get the Spotify dataset in a convenient shape by using the
abstractions provided by RecList (https://reclist.io/)

NOTE: THIS MAY TAKE A WHILE THE FIRST TIME AS THE DATASET GETS DOWNLOADED!
"""
from reclist.datasets import SpotifyDataset
# get the Spotify million playlist dataset as a RecDataset object
self.spotify_dataset = SpotifyDataset(force_download=False)
# check the dataset by printing out the first playlist
print("First playlist is", self.spotify_dataset.x_train[0][0]['playlist_name
self.data_catalog = self.spotify_dataset.catalog
test_track = self.data_catalog[list(self.data_catalog.keys())[0]]
print("First track metadata is", test_track['track_name'], test_track['ar
```



RecList 🚀

RecList is an open source library providing behavioral, "black-box" testing for recommender systems. Inspired by the pioneering work of Ribeiro et al. 2020 in NLP, we introduce a general plug-and-play procedure to scale up behavioral testing, with an easy-to-extend interface for custom use cases.

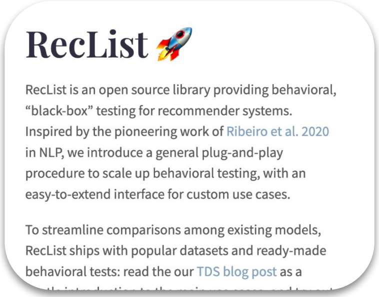
To streamline comparisons among existing models, RecList ships with popular datasets and ready-made behavioral tests: read the our TDS blog post as a gentle introduction to the main use cases, and try out our colab to get started with the code.

are actively working towards our beta, with new

```
from reclist.datasets import CoveoDataset
from reclist.recommenders.prod2vec import CoveoP2VRecModel
from reclist.reclist import CoveoCartRecList

coveo_dataset = CoveoDataset()

model = CoveoP2VRecModel()
model.train(coveo_dataset.x_train)

# instantiate rec_list object
rec_list = CoveoCartRecList(
    model=model,
    dataset=coveo_dataset
```

Train and test preparation

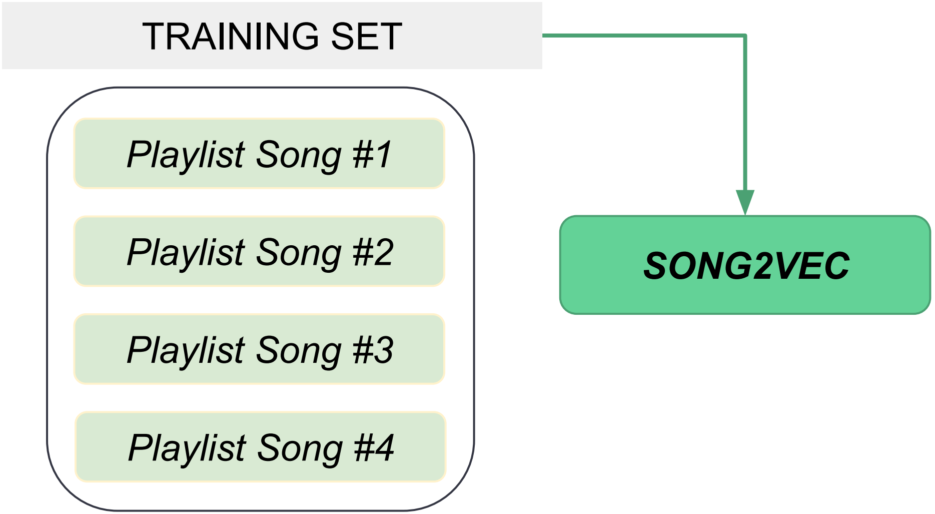
- We leverage RecList, which automatically treats the Spotify dataset (a subset!) as a sequential problem ready to be solved with a **Song2Vec** approach.
- We can prepare a training dataset *without labels*!



RecList 🚀

RecList is an open source library providing behavioral, "black-box" testing for recommender systems. Inspired by the pioneering work of Ribeiro et al. 2020 in NLP, we introduce a general plug-and-play procedure to scale up behavioral testing, with an easy-to-extend interface for custom use cases.

To streamline comparisons among existing models, RecList ships with popular datasets and ready-made behavioral tests: read the our TDS blog post as a



TRAINING SET

Playlist Song #1

Playlist Song #2

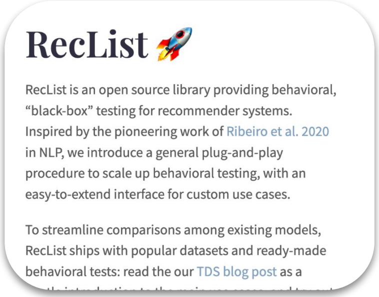
Playlist Song #3

Playlist Song #4

SONG2VEC

Train and test preparation

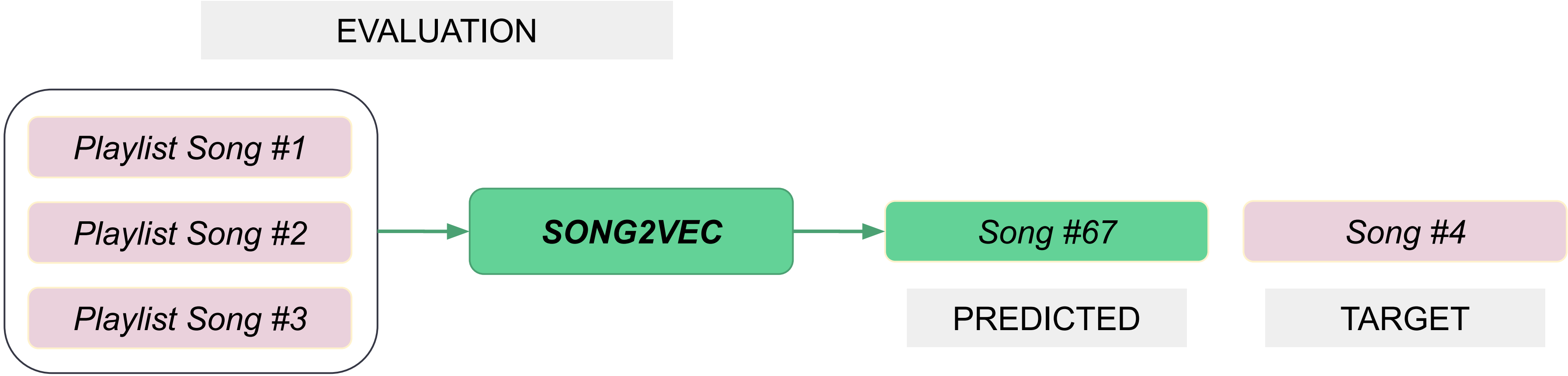
- We leverage RecList, which automatically treats the Spotify dataset (a subset!) as a sequential problem ready to be solved with a **Song2Vec** approach.
- We can prepare a training dataset *without labels*!



RecList 🚀

RecList is an open source library providing behavioral, “black-box” testing for recommender systems. Inspired by the pioneering work of Ribeiro et al. 2020 in NLP, we introduce a general plug-and-play procedure to scale up behavioral testing, with an easy-to-extend interface for custom use cases.

To streamline comparisons among existing models, RecList ships with popular datasets and ready-made behavioral tests: read the our TDS blog post as a



EVALUATION

Playlist Song #1

Playlist Song #2

Playlist Song #3

SONG2VEC

Song #67

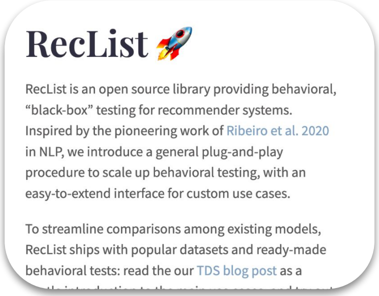
Song #4

PREDICTED

TARGET

Evaluation

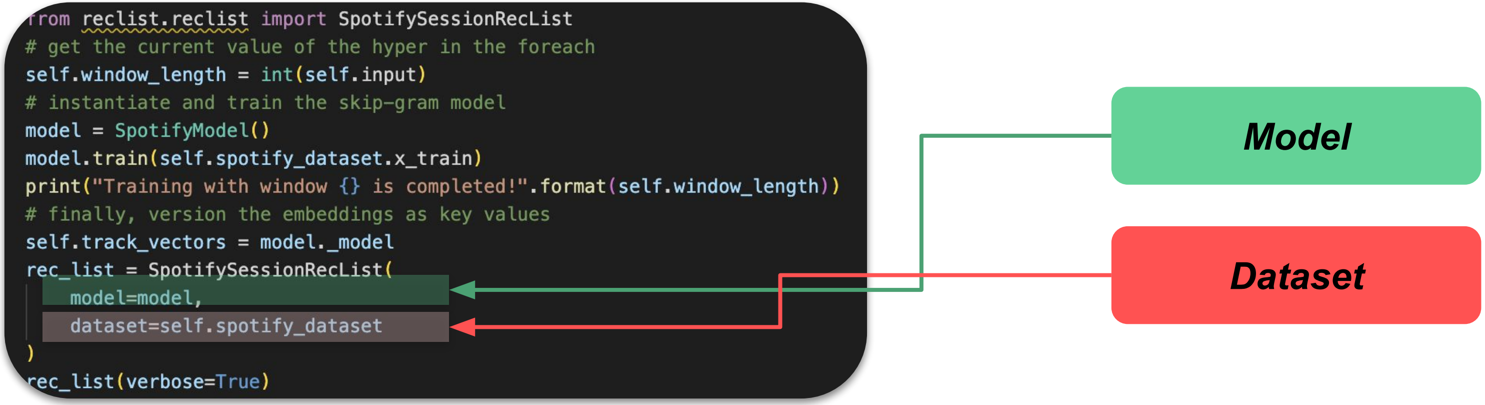
- We can evaluate our recommender model with a combination of quantitative and behavioral tests, using RecList once again.



RecList 🚀

RecList is an open source library providing behavioral, “black-box” testing for recommender systems. Inspired by the pioneering work of Ribeiro et al. 2020 in NLP, we introduce a general plug-and-play procedure to scale up behavioral testing, with an easy-to-extend interface for custom use cases.

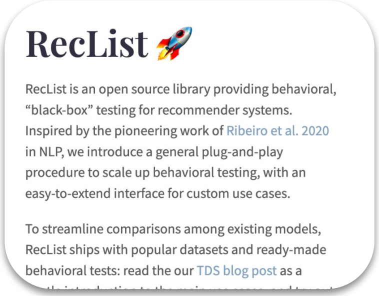
To streamline comparisons among existing models, RecList ships with popular datasets and ready-made behavioral tests: read the our TDS blog post as a



```
from reclist.reclist import SpotifySessionRecList
# get the current value of the hyper in the foreach
self.window_length = int(self.input)
# instantiate and train the skip-gram model
model = SpotifyModel()
model.train(self.spotify_dataset.x_train)
print("Training with window {} is completed!".format(self.window_length))
# finally, version the embeddings as key values
self.track_vectors = model._model
rec_list = SpotifySessionRecList(
    model=model,
    dataset=self.spotify_dataset
)
rec_list(verbose=True)
```

Evaluation

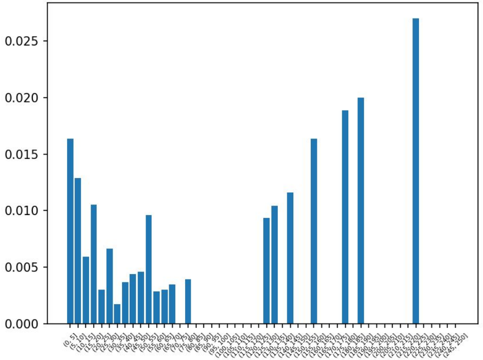
- We can evaluate our recommender model with a combination of quantitative and behavioral tests, using RecList once again.
    - For example, we compute the distribution of hit-rate across various slices of data.



RecList 🚀

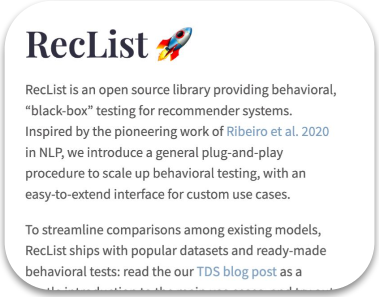
RecList is an open source library providing behavioral, "black-box" testing for recommender systems. Inspired by the pioneering work of Ribeiro et al. 2020 in NLP, we introduce a general plug-and-play procedure to scale up behavioral testing, with an easy-to-extend interface for custom use cases.

To streamline comparisons among existing models, RecList ships with popular datasets and ready-made behavioral tests: read the our TDS blog post as a



Evaluation

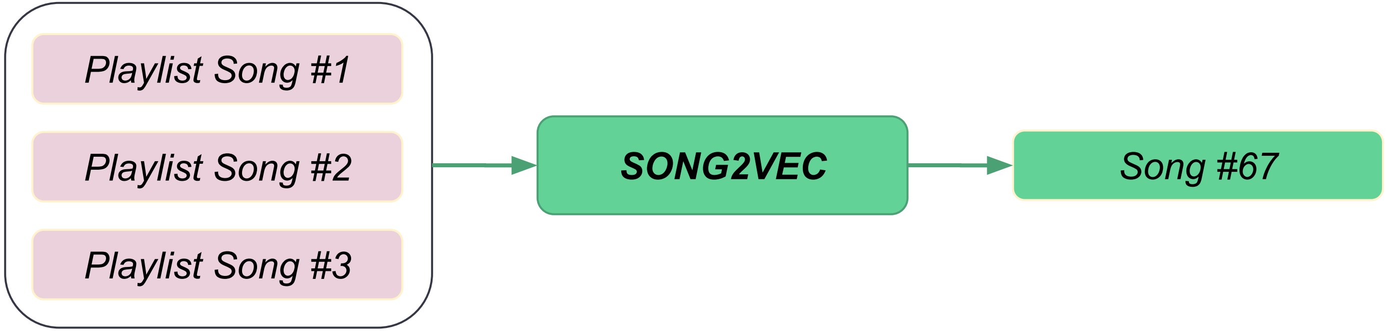
- We can evaluate our recommender model with a combination of quantitative and behavioral tests, using RecList once again.
    - For example, we could check the system robustness to “perturbation”.



RecList 🚀

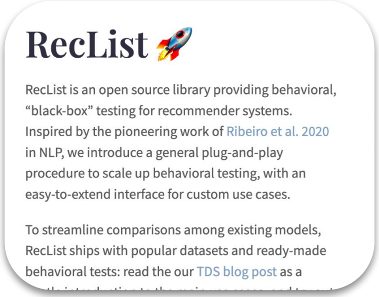
RecList is an open source library providing behavioral, “black-box” testing for recommender systems. Inspired by the pioneering work of Ribeiro et al. 2020 in NLP, we introduce a general plug-and-play procedure to scale up behavioral testing, with an easy-to-extend interface for custom use cases.

To streamline comparisons among existing models, RecList ships with popular datasets and ready-made behavioral tests: read the our TDS blog post as a



Evaluation

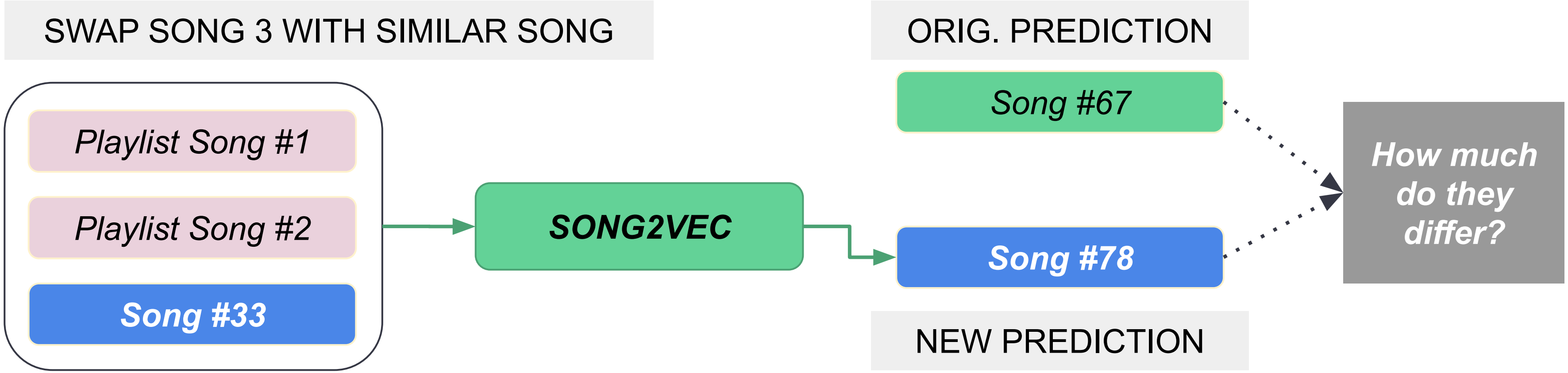
- We can evaluate our recommender model with a combination of quantitative and behavioral tests, using RecList once again.
  - For example, we could check the system robustness to "perturbation".



RecList 🚀

RecList is an open source library providing behavioral, "black-box" testing for recommender systems. Inspired by the pioneering work of Ribeiro et al. 2020 in NLP, we introduce a general plug-and-play procedure to scale up behavioral testing, with an easy-to-extend interface for custom use cases.

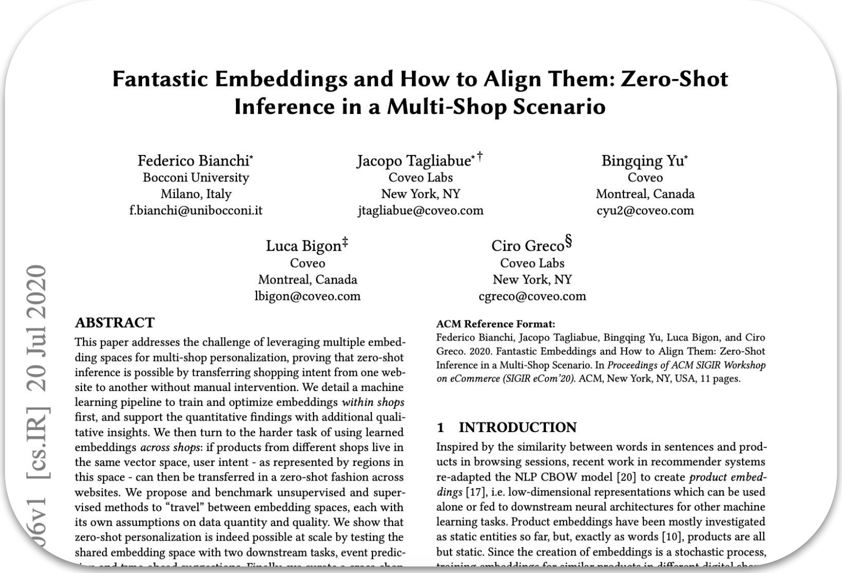
To streamline comparisons among existing models, RecList ships with popular datasets and ready-made behavioral tests: read the our TDS blog post as a



What did Song2Vec learn?

Did we learn a “good” space?

- We evaluated our recommender models on typical RecSys metrics:
  - Hit rate, MRR, sliced-based tests, etc.
- By “solving” the prediction problem, we obtained embeddings for songs, which could be used *also* as features for other models, *provided that these embeddings are “good”*.
  - What can we do to check the quality of the representational space?



Fantastic Embeddings and How to Align Them: Zero-Shot Inference in a Multi-Shop Scenario

Federico Bianchi[*]
Bocconi University
Milano, Italy
f.bianchi@unibocconi.it

Jacopo Tagliabue[*†]
Coveo Labs
New York, NY
jtagliabue@coveo.com

Bingqing Yu[*]
Coveo
Montreal, Canada
cyu2@coveo.com

Luca Bigon[‡]
Coveo
Montreal, Canada
lbigon@coveo.com

Ciro Greco[§]
Coveo Labs
New York, NY
cgreco@coveo.com

ABSTRACT

This paper addresses the challenge of leveraging multiple embedding spaces for multi-shop personalization, proving that zero-shot inference is possible by transferring shopping intent from one website to another without manual intervention. We detail a machine learning pipeline to train and optimize embeddings *within shops* first, and support the quantitative findings with additional qualitative insights. We then turn to the harder task of using learned embeddings *across shops*: if products from different shops live in the same vector space, user intent - as represented by regions in this space - can then be transferred in a zero-shot fashion across websites. We propose and benchmark unsupervised and supervised methods to “travel” between embedding spaces, each with its own assumptions on data quantity and quality. We show that zero-shot personalization is indeed possible at scale by testing the shared embedding space with two downstream tasks, event predic-

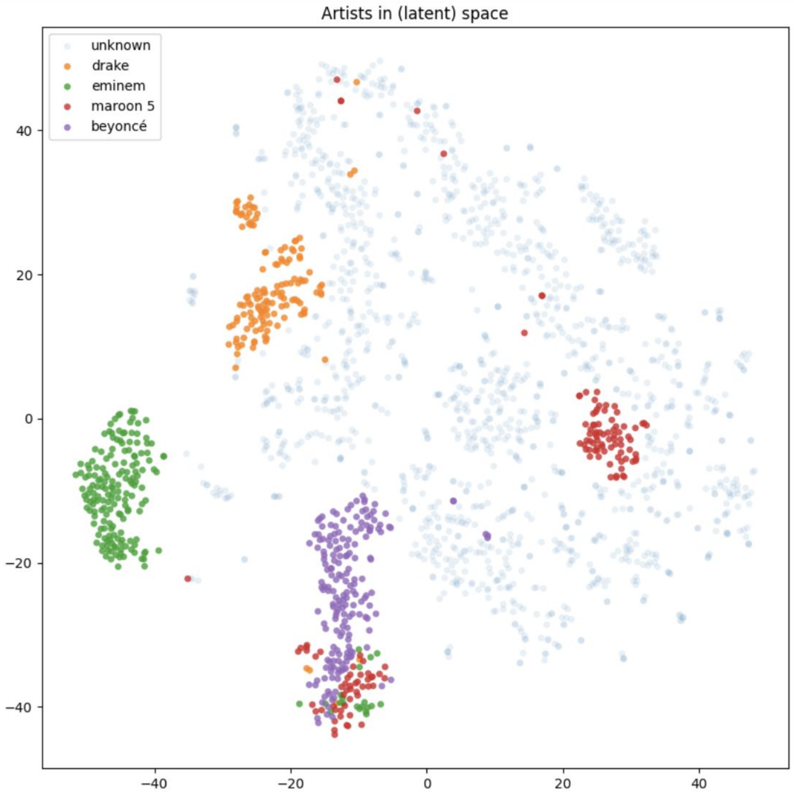
ACM Reference Format:
Federico Bianchi, Jacopo Tagliabue, Bingqing Yu, Luca Bigon, and Ciro Greco. 2020. Fantastic Embeddings and How to Align Them: Zero-Shot Inference in a Multi-Shop Scenario. In *Proceedings of ACM SIGIR Workshop on eCommerce (SIGIR eCom’20).* ACM, New York, NY, USA, 11 pages.

1 INTRODUCTION

Inspired by the similarity between words in sentences and products in browsing sessions, recent work in recommender systems re-adapted the NLP CBOW model [20] to create *product embeddings* [17], i.e. low-dimensional representations which can be used alone or fed to downstream neural architectures for other machine learning tasks. Product embeddings have been mostly investigated as static entities so far, but, exactly as words [10], products are all but static. Since the creation of embeddings is a stochastic process,

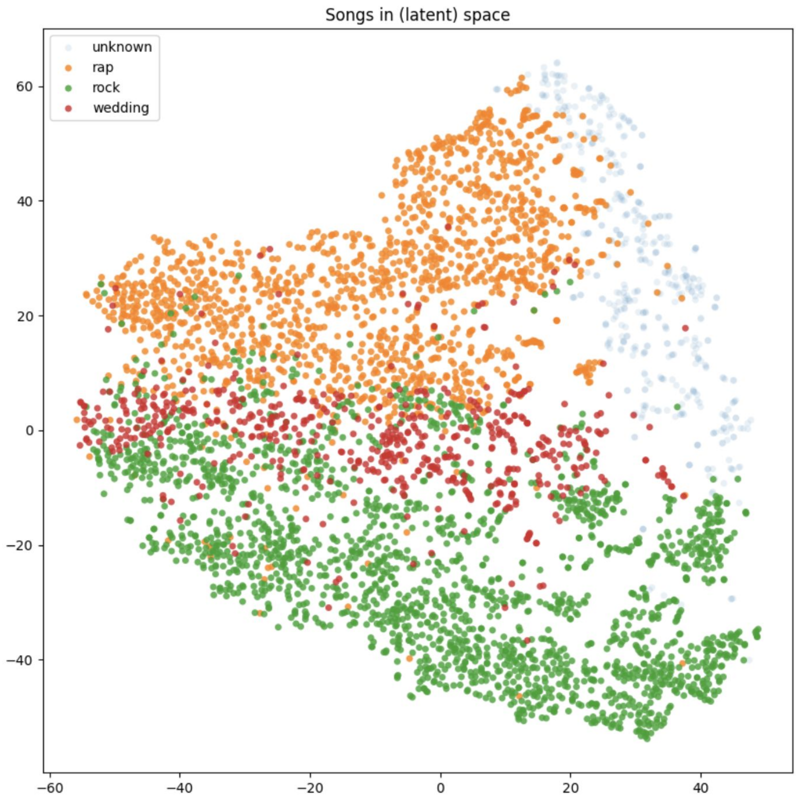
Did we learn a “good” space?

- Remember: a “good” space is a space where similar songs are close to each other!
- First check is a qualitative check: if we can label our entities (songs in this case) with human-readable concepts, we can inspect the space (with TSNE) to see if it matches our intuitions.
- **Example #1**: by tagging tracks with their artists, we can see beyonce and maroon 5 overlap more than beyonce and eminem.



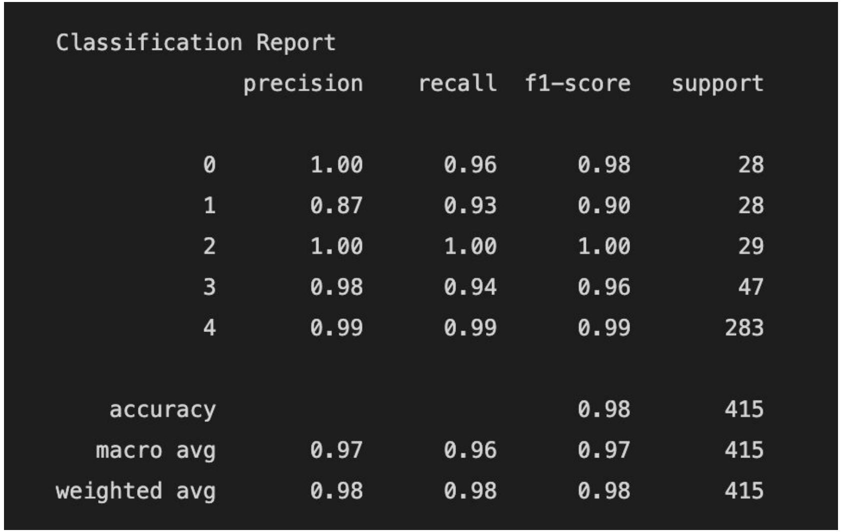
Did we learn a “good” space?

- Remember: a “good” space is a space where similar songs are close to each other!
- First check is a qualitative check: if we can label our entities (songs in this case) with human-readable concepts, we can inspect the space (with TSNE) to see if it matches our intuitions.
- **Example #2**: by tagging tracks with keywords in playlists, *rock* and *pop* are clustered naturally, *wedding* songs overlap both.



Did we learn a “good” space?

- Second check is a quantitative check: if we can label our entities with independently established categories, we can train a classifier: input song vector, output category.
- **Example**: we can train a multi-label classifier that takes as input the embeddings we trained, and classify the artist for the song.
    - Intuitively, the better the classification, the more it “resembles” our intuitions.
    - The “accuracy number” *per se* is not meaningful, but this may be useful *to compare between different embedding spaces.*



```
Classification Report
              precision    recall  f1-score   support


           0       1.00      0.96      0.98        28
           1       0.87      0.93      0.90        28
           2       1.00      1.00      1.00        29
           3       0.98      0.94      0.96        47
           4       0.99      0.99      0.99       283


    accuracy                           0.98       415
   macro avg       0.97      0.96      0.97       415
weighted avg       0.98      0.98      0.98       415
```

0 drake, 1 maroon 5, 2 eminem, 3 beyoncé, 4 is unknown.

Coding time!