1. Introduction

1.1.Business Understanding

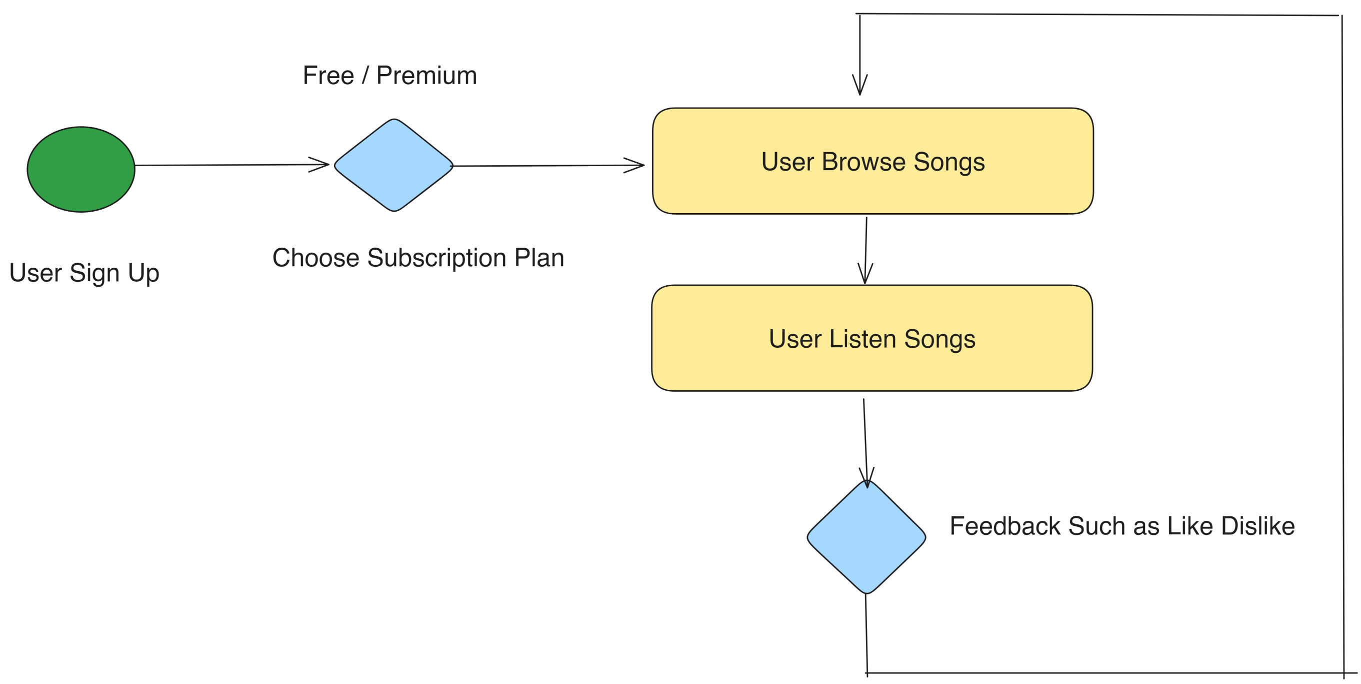
Currently music on-line platform is on the rise. However people still can't fully developed shared their taste. Our Founder feel those and start to develop Music Streaming Platform called **ontrack**. Our Business is based on Subscription Model. Our Catalogue is Quite Huge contain ~17632 Artists.

Our business is on decline, previous month we serve nearly 5000 users, however there is significant **churn** leaving our platform only having ~2000 users.

We are in data science team,asked to assist business development team to tackle this situation.

1.2. Problem Definition

Business Process



Problem :

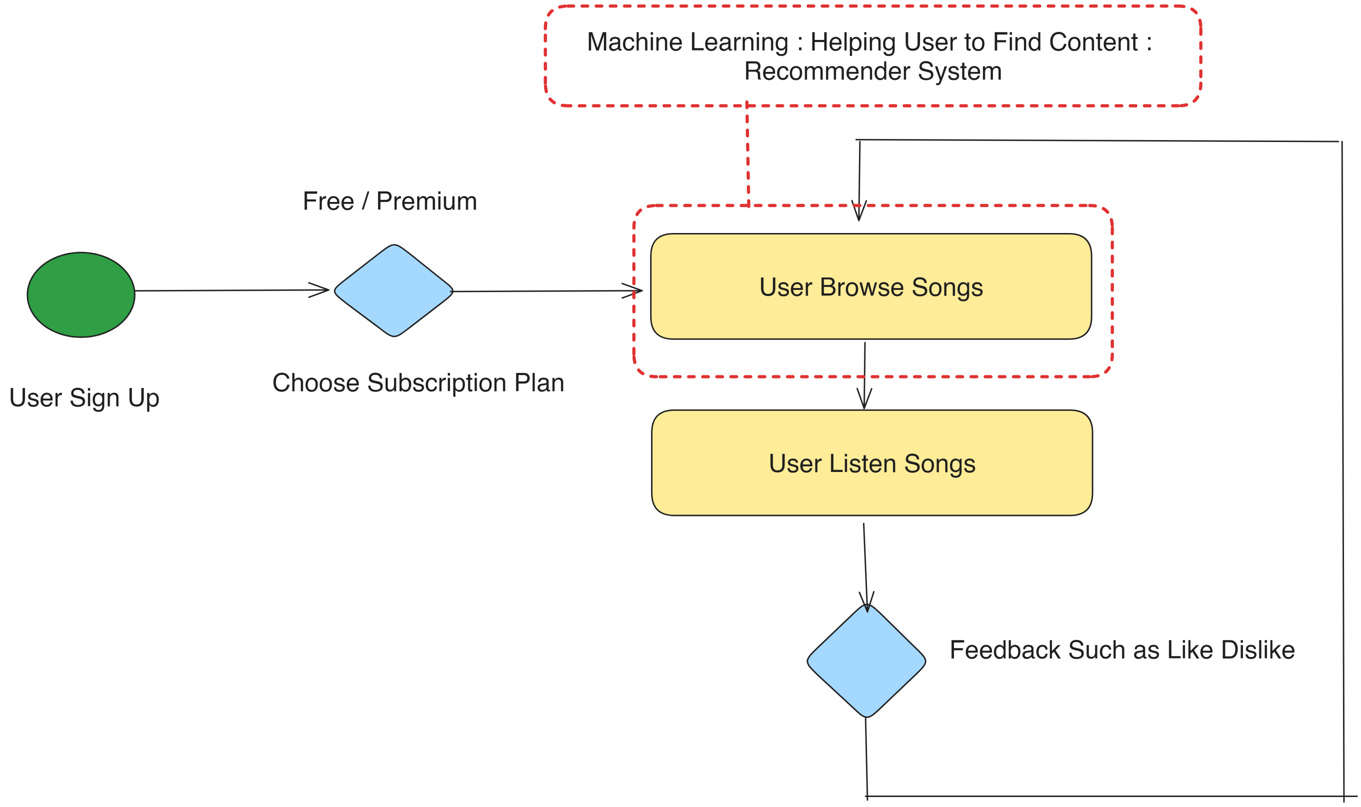
User Churn ~60% !

1.3. Business Metrics

Business Metrics : Churn Rate

1.4. Identifying Machine Learning Problem

Process that could be helped with machine learning :



Possible Solutions :

No	Solution	Task	Detailed Task	Metrics
1	Predict number of user play count on each artist	Regression	Count Regression	Prediction Error (RMSE,MAE,etc)
2	Predict whether user will like the artist or not	Classification	Binary Classification	Decision Support Metrics (Precision,Recall,AUC,F1,etc)
3	Predict condfidence scale (0-1) how user like an item	Regression	-	Prediction Error (RMSE,MAE,etc)
4	Predict the ranking from user to artists	Ranking	Pairwise Ranking	Ranking Metrics (ex : NDCG,MAP,MRR,etc)

1.5 Objective Metrics

In this approach we will approach to Predict the ranking from user to artists.

No	Metrics	Advantage	Disadvantage
1	Normalized Discounted Cumulative Gain (Measure average corrent prediction of each metrics)	1.Discounting mechanism, it will penalty upper rank if incorrectly predicted (Higher ranked item more important)	1. Not Quite Intuitive in Explanation 2. Not directly optimizeable

No	Metrics	Advantage	Disadvantage
2	Mean Average Precision (Measure Rolling Precision between all recommended items)	1.Intuitive to be explained	1. Does not care about ordering

DCG Formulas :

```
\begin{equation}\\
\begin{split}\\
1. \text{DCG}= \frac{\sum_{i=1}^{k} 2^{rel[i]}-1}{log_{2}([i]+2)} \\
\text{DCG = Discounted Cumulative Gain} \\
k= \text{number of items in recommendation}\\
rel[i] = \text{relevance score at item in position i th}\\
\text{relevance score could be any function / could be customized}

\end{split}\\
\end{equation}\\
```

```
\begin{equation}\\
\begin{split}\\
1.NDCG \text{= Scaled Version of DCG (0 to 1)}\\
NDCG = \frac{DCG}{IDCG}\\
\text{IDCG = perfect DCG score when item ranked correctly.}\\
\end{split}\\
\end{equation}\\
```

MAP Formulas :

```
\begin{equation}\\
\begin{split}\\
1. AP @K = \frac{\sum_{i=1}^{K} \text{Precision@K=i}}{K} \\
AP \text{@K}= \text{Average Precision at K-items} \\
\end{split}\\
\end{equation}\\
```

```
\begin{equation}\\
\begin{split}\\
\text{MAP@K} = \frac{\sum_{u \in U} \text{Average Precision(u)}}{K}\\
U = \text{all users}\\

\end{split}\\
\end{equation}\\
```

With aforementioned consideration, we choose **NDCG** as our model metrics.

2. Related Work

---

1. Hu, Y., Ogihara, M.: Nextone player: A music recommendation system based on user behaviour. In: Int. Society for Music Information Retrieval Conf. (ISMIR'11) (2011)

2. Hariri, N., Mobasher, B., Burke, R.: Context-aware music recommendation based on latenttopic sequential patterns. In: Proceedings of the sixth ACM conference on Recommender systems, pp. 131–138. ACM (2012)

3. Dataset and Features

The dataset obtained from http://www.last.fm, online music system. The dataset itself contains several files

1. `artists.dat`

   Contains features :

   - `name`: string : contain artistname
   - `id` : integer : contain artistID
   - `url`:string : link to artist page
   - `pictureURL`:string : link to artist picture

2. `user_artists.dat`

   Contains features :

   - `userID`: string : contain userID
   - `artistsID`: string : contain artistID
   - `weight`: int: number of playcounts song from given `artistID`

4. Methods

4.1. Recommender System Introduction

A recommender system is a type of information filtering system that predicts and suggests items or content that a user might be interested in. Its main objective is to provide personalized recommendations based on the user's preferences, historical data, and patterns of behavior.

Recommender systems are commonly used in e-commerce platforms, streaming services, social media platforms, and various other applications where there is a large amount of data and a need to assist users in finding relevant items or content. These systems employ algorithms and techniques to analyze user data, item characteristics, and other relevant factors to generate recommendations. We have implicit feedback dataset that reflects number of user plays from certain artist.

Recommender Problem

```
   Given item from all catalogue, we will predict the utility of each item
to each user, we want to maximize the utility.
```

Data Requirement

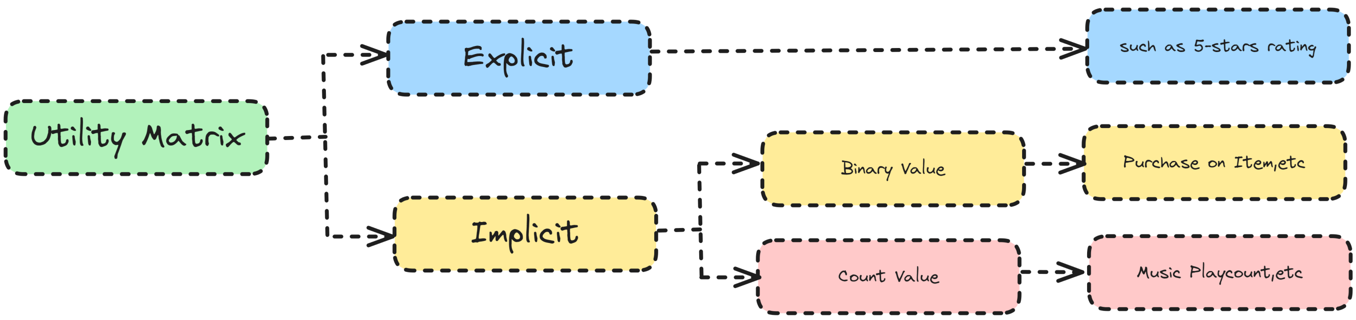
To develop recommender system, we need data, contain preferences /utility, it's called as utility matrix.

A utility matrix, also known as a preference matrix or rating matrix, is a fundamental concept in recommender systems. It represents the preferences or ratings of users for different items in a structured matrix format.

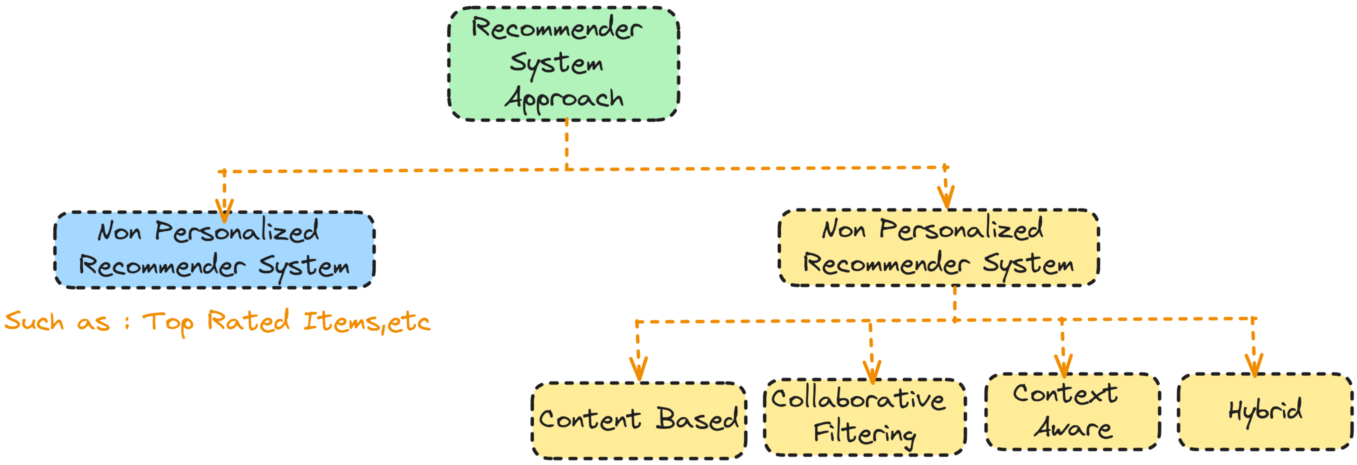
In a utility matrix, the rows represent users, and the columns represent items. Each cell in the matrix corresponds to a user's rating or preference for a particular item. The ratings can be explicit, such as numerical scores or ratings given by users, or implicit, such as purchase history, views, or clicks.

src

Types of utility matirx :



Recommender System Approach



In this approach we are dealing with implicit feedback data, hence we can utilize collaborative filtering approach.

4.2.Baseline (Popularity Recommendation)

How to generate recommendation : recommend item that has the most interaction (most played in this context.)

4.3. Alternating Least Square (Implicit Feedback Matrix Factorization)

Hu, Yifan, Yehuda Koren, and Chris Volinsky. 2008. “Collaborative Filtering for Implicit Feedback Datasets.” In 2008 Eighth IEEE International Conference on Data Mining, 263–72. IEEE.

Implicit Data does not like explicit data, does not have scale of preference from like to dislike, in implicit scenario we can only sure about it scales, commonly objective are to predict the utility (such as rating)

```
\begin{equation}\\
\begin{split}\\
Minimize \rightarrow \sum_{u \in U} \sum_{i \in I}( r_{user=u,item=i}-
\hat{r_{user=u,item=i}})^2 \\
with\\
\hat{r_{u,i}} = \text{predicted ratings on user u and item i}\\
\end{split}\\
\end{equation}\\
```

```
\begin{equation}\\
\begin{split}\\
\text{ if we come from matrix factorization model,then } \\
Minimize \rightarrow \sum_{u \in U} \sum_{i \in I}( r_{user=u,item=i}- \mu -
b_{user}-b_{item}-q_{u}\cdot p_{i})^2 \\
with\\
b_{user} = \text{user bias}\\
b_{item}= \text{item bias}\\
p_{u}= \text{user preference factor}\\
q_{i} = \text{item  factor}\\
```

```
p_{u} & q_{i} \text{called as latent factor }\\
\end{split}\\
\end{equation}\\
```

in implicit feedback our confidence ranging from `0` to `1`, so we need to adjust our Objectives

```
\begin{equation}\\
\begin{split}\\
\text{ if we come from matrix factorization model,then } \\
Minimize \rightarrow \sum_{u \in U} \sum_{i \in I} c_{ui}(p_{ui}- x_{u}^T
\cdot y_{i})^2 + \lambda (\sum_{u \in U}||x_{u} || + \sum_{i \in I}||y_{i}
||) \\
with\\
(c_{ui} = 1 + \alpha r_{ui} )\text{ denotes as scaling factor from implicit
interaction, 1 if } r_{ui} = 0 \\

\alpha = \text{weight (hyperparameter)}\\
x_{u}= \text{user preference factor}\\
y_{i} = \text{item  factor}\\
\text{remember that our confidence ranging from 0 to 1, meanwhile our
interaction such as count had value from 0 to} \infty \\
\text{we can convert as } p_{ui} \\
    \begin{dcases}
        r_{ui} > 0 \rightarrow p_{ui} = 1 \\
         r_{ui} = 0 \rightarrow p_{ui} = 0 \\
    \end{dcases} \\

x_{u} & y_{i} \text{called as latent factor }\\
\end{split}\\
\end{equation}\\
```

the

4.3. Bayesian Personalized Ranking

Rendle, Steffen, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. 2009. “BPR: Bayesian Personalized Ranking from Implicit Feedback.” In Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence, 452–61. AUAI Press.

4.4. Logistic Matrix Factorization

Johnson, Christopher C. 2014. “Logistic Matrix Factorization for Implicit Feedback Data.” Advances in Neural Information Processing Systems 27.

The concept is similar to Alternating Least Squares, such which weight interaction with alpha, it only modified the c by adding the log and some epsilon

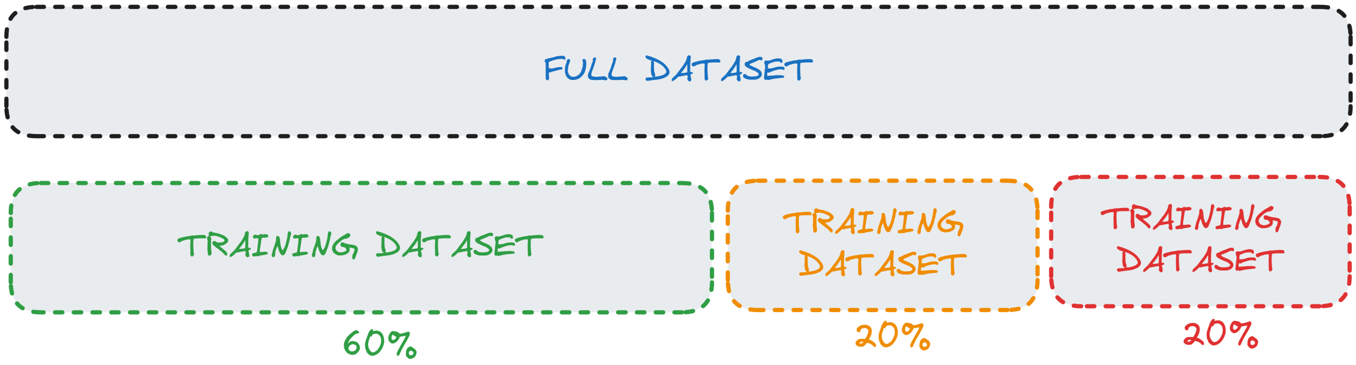
```
\begin{equation}
\begin{split}
```

```
l_{ui} = \text{similar to } p_{ui} \text{in ALS} \\
l_{ui} \\
    \begin{dcases}
        \text{user interact with item} \rightarrow l_{ui} = 1 \\
        \text{user not interact with item} \rightarrow l_{ui} = 0
    \end{dcases} \\

\text{from this we could approach this through probabilistic approach,
simply find function that output probability, yes you are right, Sigmoid
function } \\
\text{Recall again , we still use matrix factorization approach}\\
\text{Hence you will meet again } b_{user},b_{item},\text{latent factors : }
x_{user},y_{item}\\
\text{Hence, probability of user interacting with an item,given factors+bias
is linear function which converted as probability approx. through sigmoid}\\
p(l_{ui}|x_{user},y_{item},b_{user},b_{item}) = \frac{exp((x_{user}^T\cdot
y_{item}) + b_{user}+b_{item})}{1+exp((x_{user}^T\cdot y_{item}) +
b_{user}+b_{item})}\\

\text{we are not done yet, we just output the probability, now we need to
optimize weight and factors } x_{user},y_{item},b_{user},b_{item} \\
\text{we can use Maximal Likelihood Estimation}\\
(c_{ui} = 1 + \alpha \cdot log(1+(r_{ui} / \epsilon)) )\text{ denotes as
scaling factor from implicit interaction}\\
\text{Add another term, Bernoulli Trials, in this case we consider user and
items interaction are independent}
\text{our objective become}\\
\text{Maximize} \rightarrow(ProbaInteractions(R)|X,Y,bias) =
\Pi_{u,i}p(l_{ui}|x_{user},y_{item},b_{user},b_{item})^{\alpha r_{ui}}(1-
p(l_{ui}|x_{user},y_{item},b_{user},b_{item}))^{1-(\alpha r_{ui})}\\
\text{Hence the computation are expensive we can use Log --> Log
Likelihood}\\
\text{to obtain parameter that maximize probability of interaction}\\
log p(X,X,bias|R) = \sum_{u,i} \alpha r_{u,i}((x_{u}^T\cdot y_{i}) +
b_{u}+b_{i})-(1+\alpha r_{u,i}) log(1+exp(x_{u}^T\cdot y_{i}) +
b_{u}+b_{i})-\text{some regulatizaton on latent factors (ridge
regularization)} (\lambda/2 ||x_{u}||^2) - (\lambda/2 ||y_{i}||^2)
\end{split}
\end{equation}
```

To update the weight (bias,latent factors) we can use Gradient Descent , most common in Recommender System is **Stochastic Gradient Descent**.

5.Experiments / Results / Discussion

5.1. Data Splitting Strategy



5.2. Data Preprocessing

5.2.1 Mapping `userID` & `artistsID` into Ordered ID

Our data requirement is in utility matrix form, it would be hard to access each element in utility matrix since utility matrix has ordered id. We need to create mapping for both `userID, artistsID to orderedID` and vice versa.Our Mapping is in python dictionary object.

```
user_id_to_ordered_id = {
    userID:ordereduserID
}

example :
user_id_to_ordered_id = {
    2:1
}
```

for later usage (such as : API) we will serialize object using `joblib.dump` function as pickle file (`.pkl`), named :

1. `user_id_to_ordered_id.pkl`
2. `ordered_id_to_user_id.pkl`
3. `artist_id_to_ordered_id.pkl`
4. `ordered_id_to_artist_id.pkl`

5.3. Model Selection

Result :

No	precision @10	map@10	ndcg@10	auc@10	model
1	0.135773	0.062366	0.134182	0.565975	AlternatingLeastSquares
2	0.112814	0.053603	0.115200	0.553911	BayesianPersonalizedRanking
3	0.014199	0.004576	0.013120	0.506782	LogisticMatrixFactorization

from several metrics above we can see that `AlternatingLeastSquares` outperform other models --> move to Hyperparameter Tuning Phase

5.4. Hyperparameter Tuning

In this process we will find the best parameters pair for our best selected models,`AlternatingLeastSquares`.

Hyperparameters that are available in `AlternatingLeastSquares` models are :

1. `factors`

   number of latent factors, commonly found in matrix factorization model, including `AlternatingLeastSquares`, for this hyperparameter we will pick candidate value `[100,200,300,400,500]`

2. `alpha`

   In our `AlternatingLeastSquares` model we have weight `alpha` as confidence magnitude from user implicit interactions such as number of clicks, etc. values from alpha we would like to choose, ranging from `0.01` to `1.0`.

3. `regularization`

   number of how strong we imposed regularization on weights, this due to the sparsity of the data, we dont want the weight become so big --> prone to overfitting. For this hyperparameter we will try some values ranging from `0.01` to `0.2`

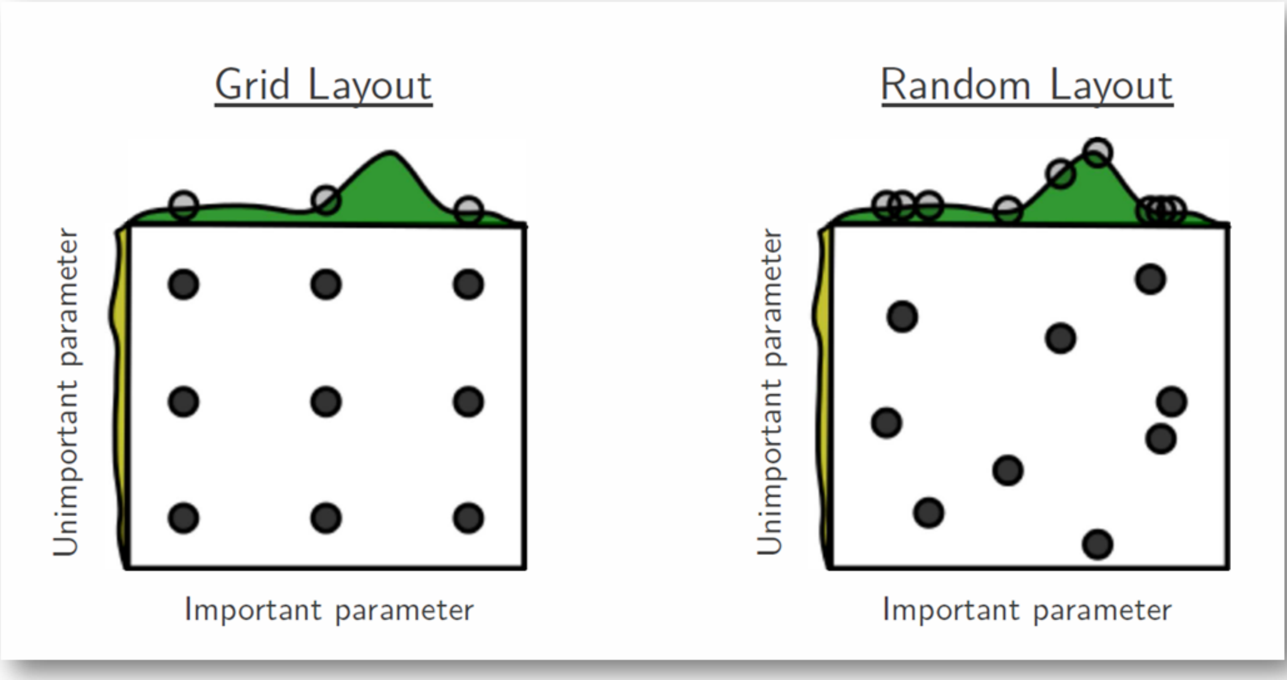
Some approach in hyperparameter tuning :

1. **GridSearch**

This approach simply fit all combinations from hyperparameter candidate, let say we have 3 hyperparameter with each candidate value of 3, number of model fitting is 3*3*3 = 27 times fitting.This is not **efficient** approach, especially with recommender system model with huge size of data.

2. **RandomizedSearch**

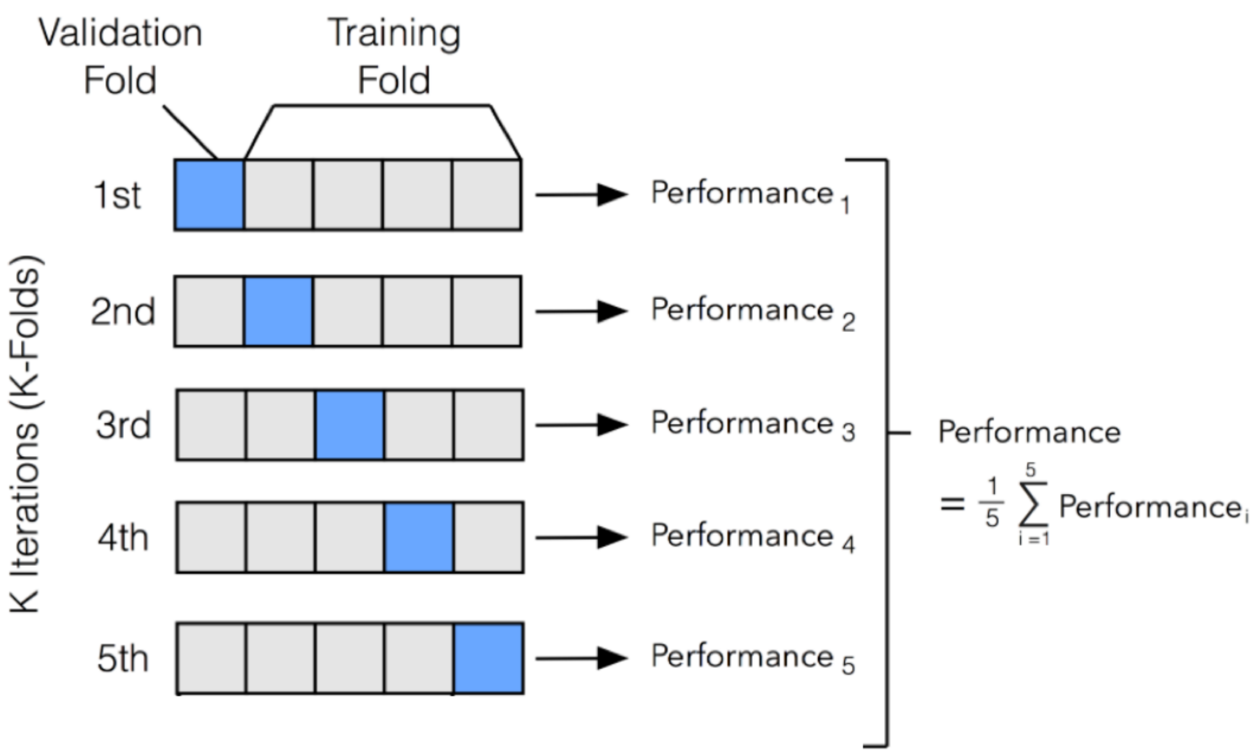
This approach perform better than **GridSearch** approach in terms of computation because it only samples / subset from our hyperparameter.



3. **Bayesian Optimization**

   This approach hyperparameter value as Gaussian Process problem where the hyperparemeter value is the product of Surrogate function (such as Gaussian Process), we will use this approach bcause it efficient in terms of computation and provide better result. Don't worry we don't have to understand all right now, and we will not coding it from scratch, we will use **optuna** package for now.

For hyperparameter set up we will perform Cross-Validation with --> **K-Fold Cross Validation**



source

Best parameters :

1. `factors` : **100**
2. `alpha` : **0.5097051938957499**
3. `regularization` : **0.16799704422342204**

5.5 Evaluation

On final evaluation we measured tuned model on test set, the result

precision @10	map@10	ndcg@10	auc@10
0.16635468872652617	0.0844339689737369	0.17261162377361844	0.574831593418623

5.6. Sanity Check on Recommendation

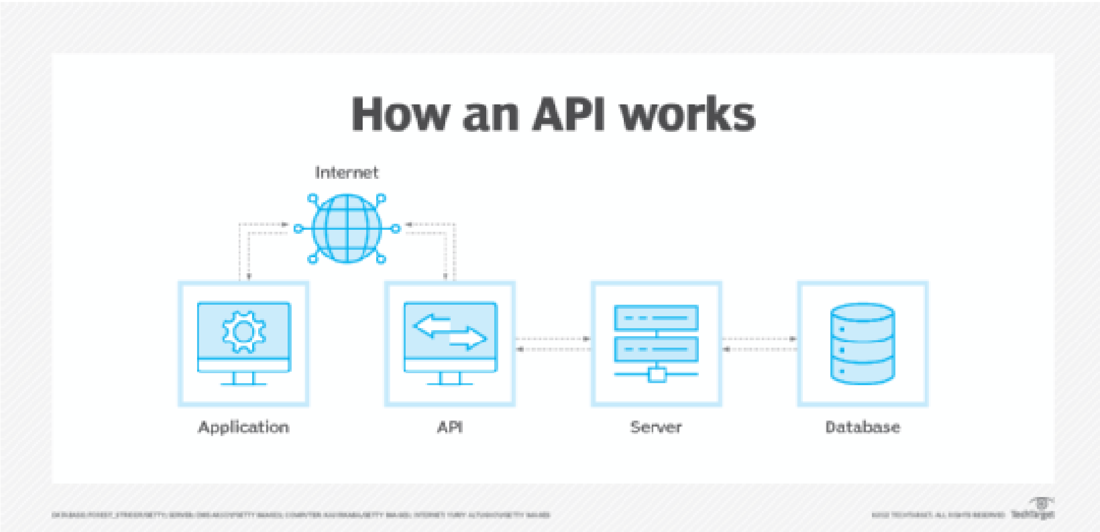
6.Conclusion

6.1. Further Work

1. Using more user oriented metrics such as **Diversity, Serendipity, Novelty**
2. Using Multistage Approach
3. Apply Graph Data (Friends)
4. Use Metadata as Features,for Using Factorization Machine

7. Product

7.1. API



src

API (Application Programming Interface) is a set of rules that allows different software applications to communicate with each other. It provides a standardized interface for accessing and utilizing functionalities or data from external services, enabling seamless integration and interoperability between software components.

7.1.1 Running API

To run API :

```
cd song_recommender
python3 src/api.py
```

check localhost:8000/docs for documentation.

7.1.2 Request Format

```
curl -X 'POST' \
  'http://localhost:8080/recommend/' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
  "userid": 2,
  "item_to_recommend": 10
}'
```

7.1.3 Response Format

```
{
  "recommended_artist": [
    "Michael Jackson",
    "Roxette",
    "a-ha",
    "Lily Allen",
    "Annie Lennox",
    "Björk",
    "Rammstein",
    "Elvis Presley",
    "Norah Jones",
    "Vangelis"
  ]
}
```

8. Experiment with your own.

To Run retraining process

```
python3 src/retrain_model.py --experiment_name='somename' --
training_filename='file.csv'
```

Output from retraining

1. Mapper `userID` and `artistID` --> mapping folder {experiment_name}_.pkl
2. trained model saved in `f"../models/{experiment_name}_als_tuned_model.pkl"`