10805 Final Project Report

Yilin Yang **Ni Zhan**

Music Recommender System

Introduction

Recommender systems are an area of interest for music, shopping, and other content. With popularity of digital music platforms such as Spotify, Apple music, personalized recommender systems have high commercial value. An effective system should account for users' previous preferences and recommend novel music which the user is likely to enjoy. Some challenges are the large library of available songs, high dimensionality and heterogeneity of data representing the songs. Collaborative filtering (CF) is widely used in recommender systems, and uses feedback from many users on the items (in our case, songs). Some CF methods disregard the meta-data of items, while recent context-aware methods utilize meta-data. Including richer data may boost performance, and meta-data of songs may be genre, mood, artist, audio signals, etc. This work uses the Million Song Dataset (MSD) [1], and associated Taste Profile [12] with user listening history to develop recommender systems. We use simple recommendation baselines and CF. This work aims to build an effective recommender system in Spark and evaluate the recommender system methods.

We use simple benchmarks of recommending the most popular songs (no personalization), and recommending the most popular songs of same artists. We use weighted matrix factorization CF on Spark. We test dimensionality reduction using song meta-data as input, generating latent factors for songs. The latent factors can be combined with clustering such as k-means for item-based recommendation. We quantitatively evaluate our recommender system using mean average precision as a metric [12] and test-split on Taste Profile data, and qualitatively evaluate recommendations based on visualization and human inspection of nearby songs.

Related Work In CF, the feedback matrix $M \in \mathbb{R}^{m \times n}$ with m users and n items contains feedback from each user for the items. The feedback could be explicit such as ratings, or implicit such as number of listens, number of clicks, views, etc. In matrix factorization, we create a low-rank approximation of the feedback matrix M.

$$M \approx \hat{M} = U^T V$$

where $U \in \mathbb{R}^{k \times m}$ is a low-rank approximation of users, and $V \in \mathbb{R}^{k \times n}$ approximates items [12]. Some matrix factorization methods include bayesian personalized feedback from implicit feedback [14] and weighted matrix factorization [6]. Both of these matrix factorization algorithms are aimed at implicit feedback datasets, which do not have explicit ratings for items. One context-aware method combined with CF was proposed by Oord, Dieleman, and Schrauwen. They used convolutional neural networks with songs' audio signals as input to generate latent factors. The loss function was mean squared difference between CNN output and latent factor of song obtained from weighted matrix factorization. Liang, Zhan, and Ellis used generative model with latent factors for users and songs.

Dataset The MSD full dataset is around 280 GB, with one million songs. Data for each song is stored in HDF5 format with 55 fields, with various data structures. The main audio signal features are timbre, loudness, and pitch, and other data includes artist, song hotness, release year. The Taste Profile dataset is around 500 MB, and includes song play history for over one million users, over 380,000 songs from the MSD, and over 48 million user, song, play-count triplets in a text-file format.

Methodology

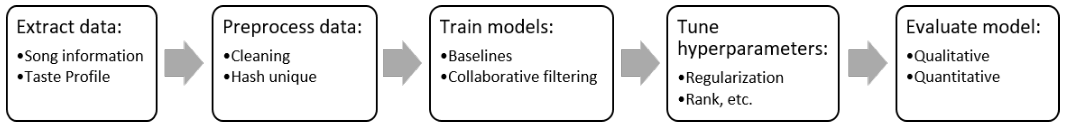

Figure 1: Pipeline for music recommendation system

Fig. 1 shows the pipeline for building the music recommender system. Here, we describe the invididual steps. We extracted certain fields of song data from the HDF5 files into csv format readable by Spark. The fields extracted were artist_name, artist_hotness, artist_familiarity, song_title, song_id, hotness, duration, loudness, year, tempo, key, mode, time_signature, track_id, and segments_timbre, because these fields seemed useful for our pipeline. The Taste Profile data was in a readily usable text-format. Data preprocessing included removing duplicate entries, and filling missing data. We filled missing hotness data with an arbitrary 0.5 hotness. We also assigned unique integer values to unique users and songs in the dataset to allow use of CF.

The baseline models were recommending the most popular songs (no personalization), and recommending the most popular songs of artist a user listened to. CF is trained by iteratively computing item factors and user factors by method in [6]. For model training, we tested different ways to split train-test data. For baseline and CF, we trained on one random song per user and evaluated recommendations for all users in the dataset. We also tested how increasing songs per user in training data and randomly splitting train-test data would affect CF performance. We recommend the top 10 songs per user, and all models were evaluated quantitatively using mean average precision (MAP) and precision at k ($p(k)$) for k=1,...,10. MAP and $p(k)$ are standard metrics used in ranking/recommender systems, and defined in Eqs. 1 and 2. MAP and $p(k)$ are especially relevant in our case because we have implicit feedback (play count) rather than explicit (such as user assigned ratings). It would not make sense to use an error metric such as mean squared error unless explicit ratings were available. For m users $U = \{u_1, ...u_m\}$, each user u_i has p ground truth items $D_i = \{d_1, ...d_p\}$ and a list of q recommended items in decreasing relevance $R_i = [r_1, ...r_q]$. We define relevance score as

$$rel_D(r) = \begin{cases} 1 & \text{if } r \in D \\ 0 & \text{otherwise} \end{cases}.$$

$$MAP = \frac{1}{m} \sum_{i=1}^{m} \frac{1}{|D_i|} \sum_{j=1}^{q} \frac{rel_{D_i}(R_i(j))}{j} \tag{1}$$

$$p(k) = \frac{1}{m} \sum_{i=1}^{m} \frac{1}{k} \sum_{j=1}^{min(|D_i|,k)} rel_{D_i}(R_i(j)) \tag{2}$$

Intuitively, MAP measures how many recommended items are in the ground truth set, with order of relevance taken into account. Precision at k measures how many first k recommended items are in ground truth set without accounting for relevance order.

Hyperparameter tuning for CF was based on the quantiative MAP and $p(k)$ metrics. The hyperparameters were rank, regularization, alpha, and max-iterations. Rank refers to rank of the low-dimension matrix representation, and increasing rank tends to improve performance up to a plateau. The regularization parameter tends to have an optimal value. Alpha relates to confidence of implicit rating, higher alpha indicates stronger preference of item with feedback vs. items without feedback. Increasing iterations should improve quantitative metric performance but requires more computation time, requiring a tradeoff.

For qualitative evaluation, we inspected a sample of recommended songs given for different users. We also used principal components analysis (PCA) to dimensionally reduce the vector representation of songs from the CF model and visualize songs' location in reduced space. We would like to see good dispersion and clustering of similar songs. Qualitative evaluation is useful because song recommendation is subjective, users have diverse musical taste, and effective recommenders should be personalized and recommend novel songs to users. We also explored a PCA representation of song features including hotness, familiarity, duration, loudness, year, and tempo.

Computation

One of the goals of this project was building the recommender system in Spark. We used AWS Elastic Map Reduce (EMR) clusters, jupyter notebooks, and pyspark for steps preprocess, train, tune, and evaluate. Extracting song information was done with EC2 instance and python code, which required around 24 hours. The data was stored in S3 bucket with shared access between AWS accounts. All subsequent steps were completed with AWS EMR instances using one m5.xlarge driver and four m5.xlarge workers. EMR release was emr-5.32.0, hadoop distribution was Amazon 2.10.1, and Spark version was 2.4.7 with python3 and maximizing Spark resource allocation. The most time-consuming operation was recommending songs for all users using CF, which required around one hour. Quantitative evaluation for one CF model required around 20 minutes. PCA was quite fast because we had tall-and-skinny matrices. We used built-in Spark methods as much as possible for increased computational efficiency, such as pyspark.mllib, pyspark.sql, and pyspark.ml. We were unable to create plots in EMR and jupyter, so we exported PCA representation to csv and created plots locally. To save computational costs, we used spot instances and total cost was $37.5.

Computational Challenges In data preprocessing, one computational challenge of large scale data is missing entries. Errors occured in subsequent steps if entries were Null or duplicated rows of songs, because we used IndexedRowMatrix for PCA. To avoid these, we filled arbitrary values for Null values and removed duplicate entries. One of the time-consuming calculations was recommending songs for all users with the CF model, which required around one hour using four m5.xlarge workers. Without cacheing the result, Spark will recalculate the recommendation every time it is used for further analysis, therefore cacheing after intense computes saves computation time. We attempted to use cosine similarity to recommend nearby songs as an alternative recommender method. However the implementation scales quadratically with number of songs. Calculation for 30,000 songs was not complete after one hour, and estimated completion time for 300,000 songs is at least four days, which is prohibitively expensive. In future work, a different method for selecting nearby items, such as locality sensitive hashing, can be used. Another computational challenge was hyperparameter tuning. State of the art methods such as Hyperband are difficult to implement, and random/grid search in Spark are difficult to customize to specialized metrics and CF model. Therefore the hyperparameter tuning was done by manual random search.

Results

Table 1 shows quantitative metrics for different recommender models, with evaluations based on top 10 model recommendations. "Top 10 songs" refers to always recommending the 10 songs which were listened to by the most unique users. For "same artist songs", one artist which the user had listened to was selected randomly and the artists' hottest songs were recommended in order. For artists with less than 10 songs, the top 10 songs filled in remaining recommendations. For CF models, 80/20 split indicates train/test random split, while remaining indicate 1, 2, or 3 songs per user in the trainset. These are reasonable trainsets because each user listened to at least 10 unique songs, with around 60% of users listening to 10-47 unique songs. We see that CF performance greatly improves when trained on more data per user. "Same artist" has highest quantitative performance overall, however a user may want to listen to different artists than one they already know. Our results are comparable with similar Bayesian personalized ranking matrix factorization (BPR-MF) [12]. Fig. 2 shows $p(k)$ for k=1,...,10. The precision decreases with further recommended items. The steeper decline for CF indicates that the first few recommended items are much more relevant than later items. In contrast, the ordering for "top 10 songs" contains much less relevance information.

"CF 80/20", "CF 1song", and "CF 2song, no tune" were not hyperparameter tuned. The tuning greatly increases performance, as evidenced from comparing "CF 2song, no tune" and "CF 2song". There were six random hyperparameters configurations tested, and the final configuration was rank=25, max iterations=8, regularization=0.2, and alpha=10.

Table 2 shows some of the most popular songs and sample of recommendations for CF 3song. For example, the model was trained on user listening to "Air Force Ones" by Nelly and two songs by Jack Johnson. The CF recommended two songs by Jack Johnson, another song of OneRepublic (an artist in top 10). For another user, the model was trained on three somewhat diverse songs (hip-hop, 80s, pop-punk), and the model recommended another Lil Wayne song and one of the top 10 songs (Horn Concerto No 4). The model tends to recommend similar songs if there are strong affinities to

Table 1: MAP and precision metrics for recommender models

	MAP	$p(1)$	$p(10)$
Top 10 songs	0.0138	0.108	0.0776
Same artist songs	**0.0375**	0.302	**0.167**
CF 80/20 split	0.0152	0.0341	0.0258
CF 1song	0.0078	0.0895	0.0493
CF 2song	0.0216	0.265	0.0785
CF 3song	0.0306	**0.334**	0.104
CF 2song, no tune	0.0138	0.142	0.0692
BPR-MF [12]	0.031	-	0.043

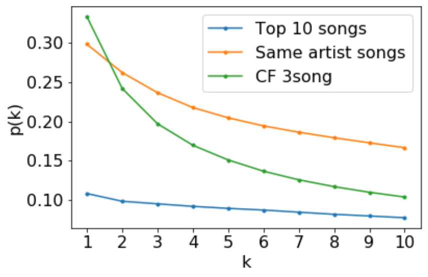

Figure 2: $p(k)$ for $k = 1, ..., 10$

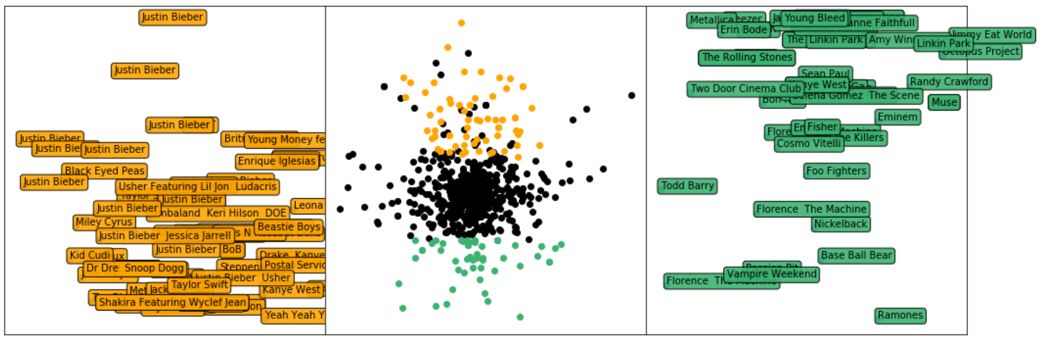

Figure 3: PCA representation of song latent factors from collaborative filtering.

songs in the training data, while also recommending generally popular songs. From both quantitative and qualitative evaluation, we note one benefit of CF: recommendations' quality and personalization improve as the model is trained on more data per user.

Fig. 3 shows the 25-dimensional song latent factors from CF reduced to two dimensions using PCA. The black points include 187,000 songs, which are mostly clustered and overlapping in the center. We highlight some regions where similar songs are clustered: Justin Bieber, pop in orange and Florence + the Machine, rock in green. We also checked PCA representation of song meta-data such as hotness, familiarity, duration, loudness, year, and tempo, which did not cluster similar songs together. In both cases, PCA did not disperse most songs, and future work can explore visualizations with t-SNE or other methods. In addition, learning similarities from song audio data such as timbre and combining with CF could be used for recommendation.

Table 2: Most popular songs and CF recommended songs based on user listens

Most popular songs	1. Sehr Kosmich - Harmonia 2. Undo - Bjork 3. Dog Days Are Over - Florence + the Machine 4. You're The One - Dwight Yoakam 5. Revelry - Kings of Leon 6. Secrets - OneRepublic 7. Horn Concerto No 4 - Barry Tuckwell
User songs	CF Recommended
Air Force Ones - Nelly Better Together - Jack Johnson You And Your Heart - Jack Johnson	1. All The Right Moves - OneRepublic 2. Bulletproof - La Roux 3. Do You Remember - Jack Johnson 4. Times Like These - Jack Johnson
Got Money - Lil Wayne, T-Pain Crazy In The Night - Kim Carnes The Curse of Curves - Cute Is What We Aim For	1. Drop The World - Lil Wayne, Eminem 2. Horn Concerto No 4 - Barry Tuckwell 3. Nothin On You - Bob 4. Invalid - Tub Ring

Image Dimensionality Reduction

Introduction

Dimensionality reduction and visualization is a basic but valuable topic in machine learning and data mining community. Extensive methods have been developed over fast decades to improve the performance of dimensionality reduction or visualization for high dimensional data. These methods can be roughly divided into two types: linear (e.g. PCA [17], Multidimentional Scaling [16]) and nonlinear dimension reduction (e.g. Locally Linear Embedding [15], AutoEncoder [5], t-SNE [11]). While all of them can be applied for dimension reduction, they hold various assumptions and they are suitable for different datasets. Desired dimension reduction techniques for recent large-scale dataset should satisfy some criteria: 1) They could capture the high dimensional global pattern of the whole dataset. 2) They persist as much useful information as possible during reduction process. 3) They are computationally efficient and scalable for large dataset.

In the machine learning community, lots of research has been conducted on the dimension reduction for high dimensional data. PCA [17] is one of the most used linear dimension reduction tools. It tries to find the orthogonal variables that minimizing the information loss through linear mapping using the original features. By doing that, the PCA process reduces to the singular value decomposition of the data matrix. The computational complexity of the classical PCA through eigenvalue decomposition is $O(nk^2 + k^3)$, where n is the number of data points and k is the dimensions of the data, which could be a concern for high dimensional data. Also, because of the intrinsic assumption of PCA, its performance for nonlinear data is not as promising as for linear data.

In addition to the linear reduction methods, there are also many tools to process nonlinear data efficiently. Among them, t-SNE [11] is a typical example. It works by preserving the similarities among data points or neighboring distances in high dimensional space when projecting them into low dimensional space. This neighboring similarity relationships are kept by minimizing the Kullback–Leibler divergence between the gaussian distribution in original space and the t-student distribution in low dimensional space. From this aspect, it is suitable for visualize high dimensional data while keeping global distribution pattern in high dimensional space. However, the exact t-SNE has a computational complexity of $O(n^2)$, which constraints its application for large dataset. Also, t-SNE is conducted in a nondeterministic way. Thus, it is possible to get differet outputs for the same input under different random conditions. Therefore, some evolutions of t-SNE have been proposed over the previous years. Maaten proposed BH-t-SNE to scale down the time complexity to $O(n \log n)$ and memory to $O(n)$ from $O(n^2)$. Fu et al. came up with the anchor t-SNE which preserves the global information using anchor points generated by K-Means. The local information is kept by neighboring distances among ordinary data points and the distances to the anchor points. What's more, the implementation of anchor t-SNE is highly parallel and could take the advantage of GPU.

Autoencoder [5] is another type of dimensionality reduction technique which is based on neural networks. One neural network is used to encode the input data and the other neural network is for decoding. It enforces the encoding features to store as much information as possible to minimize the reconstruction error between the original data and the output of the decoder. After the autoencoder, Kingma and Welling proposed variational autoencoder, which not only tries to learn the embedding of the data in low dimensional space, but also the distribution of the latent features, which could be used as a generative method.

In this project, we aim to empirically compare different dimensionality reduction methods from various aspects, such that we could have an overview on the advantages and disadvantages of these methods. More specifically, we are curious about the quality of the data pattern in low dimensional space, the required time to conduct the dimension reduction as well as the easiness to use these tools. People tend to use the tools that are easy to use and widely available. We also want to understand the underline reasons that cause different performance for these methods.

As a example dataset, CIFAR-100 [8] is explored in our project. It contains 60000 RGB images of 32×32 with 10 different coarse classes and 100 refine classes. Each image is a $32 \times 32 \times 3$ tensor, and it requires some preprocessing before feed it into different dimensionality reduction methods.

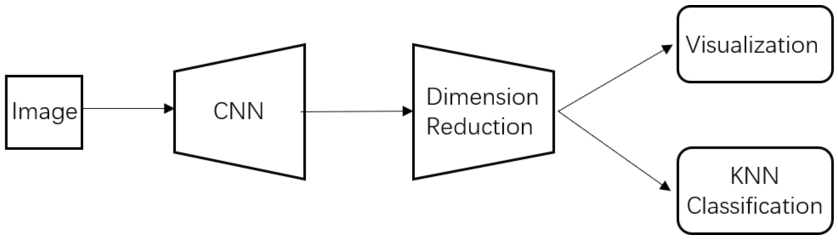

Figure 4: Diagram for CIFAR100 dimension reduction. An image is preprocessed by a pretrained Resnet-50 to extract a vector encoding, followed by various dimention reduction to mapping it to lower space. Visualization and KNN classification happen in the low dimensional space.

Methodology

The entire pipline of this project is shown in Fig 4. In this pipline, each image with shape of $32 \times 32 \times 3$ is firstly upsampled to $224 \times 224 \times 3$ to satisfy the requirement of the pretrained ResNet-50 [4] since this model was pretrained on the ImageNet dataset. At the output side of the network, we extract two kinds of features to represent each image. The first type is a vector of 2048 dimensions that is directly from the output of the ResNet model. The other type is a vector of 1024 dimensions after we fine tune the pretrained model to the CIFAR-100 datasets. CNN is used to extract the latent features is that it is suitable for dealing with image data. Most dimension reduction techniques require the input data to be a vector instead of a matrix or tensor. Thus, we choose CNN to preprocess the tensor image data into a vector. Also, the pretrained CNN could generate reasonable latent features that we rely on in the next part of the pipeline. After extracting these latent features, we feed them into six different dimensionality reduction methods: PCA [17], LLE [15], t-SNE-CUDA [2], Anchor-t-SNE [3], Autoencoder [5] and variantional Autoencoder [7]. These methods cover the common linear and non-linear dimension reduction tools. The obtained 2D latent features from these tools are used for two tasks: visualization and classification, which aim to test different properties of the methods.

We use three metrics to evaluate these dimensionality reduction tools. The first one is that if the methods could preserve the global pattern in high dimensional space. This could be evaluated by the visualization effect in 2D space, which is a qualitative metric. The second one is the required time to conduct the dimension reduction. We simply measure the running time to process 60000 CIFAR-100 images by different models despite they may be run on various hardwares like CPUs and GPUs. Since the ability to utilize suitable hardware is also an attribute we want to evaluate. For example, we run PCA and LLE on CPUs. The t-SNE-CUDA, Anchor-t-SNE, Autoencoder and variational autoencoder are run on GPUs. The last metric we focus on is the classification accuracy using KNN on the projected low dimensional space. This metric provides a quantitative sense about the quality of the dimention reduction. High KNN accuracy means relevant images are clustering together in new space and it is highly probable that this mapping preserve the relationships among images in original space. The reason for choosing KNN as the classfier is that KNN conducts the classification based on the neighboring environment around the test sample. Also, it is a non-parametric model that only depends on the input features of the data points. We use cross validation to choose the optimal K for the latent features obtained by each method, and report the highest accuracy dimension reduction method.

Computation

There are three computational phases in this project. The first phase is retriving the CIFAR-100 dataset and preprocessing it into proper type. We download the dataset from one of the Keras modules. The ResNet-50 model pretrained on ImageNet is also obtrained from Keras. Before fed into ResNet model, the original images are upsampled using the resize module of cv2 package. The fine tuning process are completed by adding a fully connected layer on the top of the ResNet. This preprocess phase is run on Colab utilizing the free GPU provided by Google.

The second phase is the dimension reduction based on the extracted features in the first phase. For PCA and LLE, there are already implemented package in sklearn, and we just use them directly. Since they do not support GPU, so we conduct them using CPU only. For t-SNE-CUDA, it is installed using pip module. The usage of t-SNE-CUDA is similar to the standard t-SNE in sklearn, which is easy to use. Unlike t-SNE-CUDA, anchor-t-SNE does not support pip to install. The only way to install it is clone from the original github and compile it locally. There was a problem when compiling its cuda component on Colab, thus we used p2.xlarge EC2 on AWS to install and use it. For autoencoder and variational autoencoder, we take the advantage of GPU on AWS to train the neural networks efficiently. The last phase is to plot the embedded features and conduct KNN to do the classification. Since the data is reduced to 2D space, which is quite small in size, so we just use local machine to plot the data points and conduct the KNN package from sklearn.

As a summary, we use Colab with GPU to preprocess the CIFAR-100 dataset to extract the vector features. The preprocess takes about 1 hour to get the final features. Then, for PCA and LLE, we use one m5.large machine on AWS to do the dimension reduction. For t-SNE-CUDA, anchor-t-SNE, autoencoder and variational autoencoder, we utilize one p2.xlarge machine on AWS to reduce the dimensionalities. The dimension reduction phase takes about 15 hours to finish with LLE as the bottleneck. The cost for m5.large is about 1.44 dollars, which is 1.8 dollars for p2.xlarge. We did not use any storage service for this project since all files are small enough to fit the default storage in these machines. All of these tasks are completed using Python. Some component of anchor-t-SNE are written in CUDA, but the authors have provided a python wrapper to use them.

Results

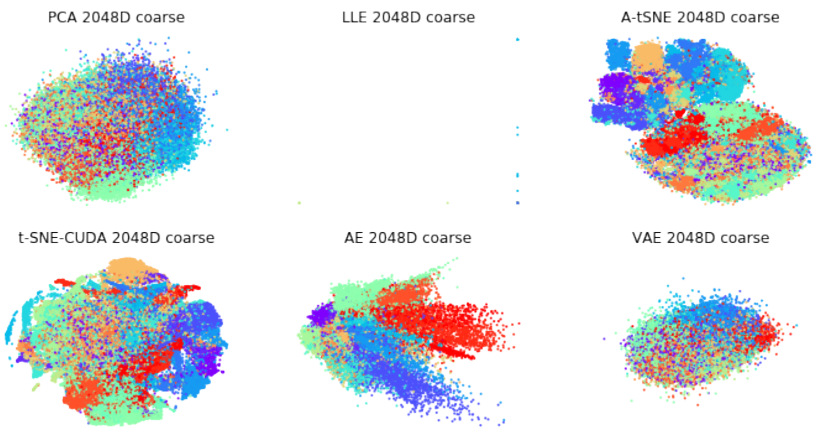

Figure 5: Plots for reduced dimensions from vectors with 2048 dimensions extracted from ResNet-50 without fine tuning. 10 coarse classes are labeled by different colors.

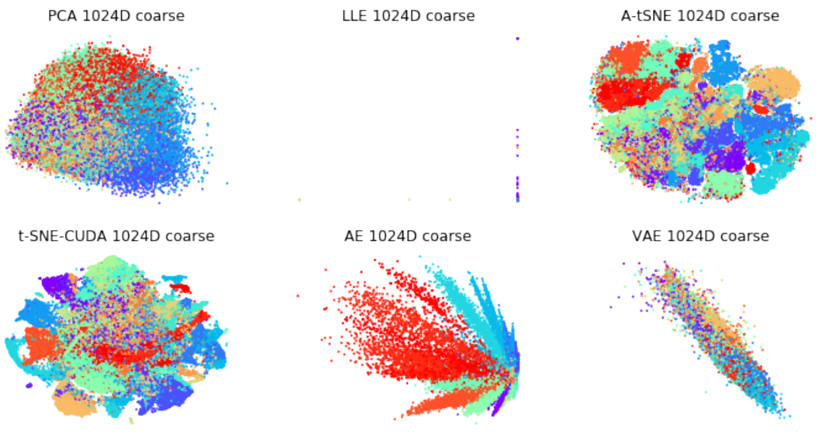

Figure 6: Plots for reduced dimensions from vectors with 1024 dimensions extracted from ResNet-50 with fine tuning. 10 coarse classes are labeled by different colors.

We present the plots of the reduced dimensions for six forementioned dimensionality reduction methods in Fig 5 and Fig 6. The input features for these two figures are different. Fig 5 are vector with 2048 dimensions directly extracted from ResNet-50 without fine tuning. Fig 6 are from vectors with 1024 dimensions with fune tuning. Solely from the visual comparison, Fig 6 looks slightly better, points with the same class are gathered closer compared to Fig 5, but the difference is not so apparent. Among different dimensionality reduction methods, LLE has the worst performance on the visualization effect. All points are distributed across a line and it is hard to tell these points. PCA and VAE have the similar performance, there is clustering among the points with the same label but the boundary between two classes is not apparent. Data overlapping is also an issue for these two methods. Autoencoder generates a special results among these methods. The clusters are not in a circular shape like in other methods, but look like spindles. Maintaining large distances among different classes might help to decrease the reconstruction loss during the training process. Anchor-t-SNE and the t-SNE-CUDA are the best methods to conduct the dimensionality reduction based on the visualization effect. The whole dataset is almost evenly distributed around the 2D space. Points with the same class are gathered together while keep a relatively apparent boundary with different classes. The generated plots of these two methods are also similar to each other since intrinsically, they are conducting the same t-SNE algorithm.

Table 3: Running time for different dimension reduction methods

Methods	2048D (s)	1024D (s)
PCA	**2.42**	**1.20**
LLE	51814.28	33631.92
t-SNE-CUDA	34.19	21.00
Anchor-t-SNE	46.89	36.52
AE	71.57 (50 epochs)	50.54 (50 epochs)
VAE	51.49 (25 epochs)	37.74 (25 epochs)

The running time for these methods are listed in Table 3. PCA is the fastest method to conduct the dimension reduction over 60000 images. This is reasonable since PCA scales linearly with the number of data points. LLE takes longest time to do the reduction task. It scales $O(n^2)$ with respect to the number of data points. What's more, LLE in sklearn has no acceleration from GPU. Anchor-t-SNE, t-SNE-CUDA, AE and VAE take similar time in this task. They share the acceleration from GPUs.

Table 4: KNN Classification Accuracy

Methods	2048D (coarse)	2048D (refine)	1024D (coarse)	1024D (refine)
PCA	0.2218	0.0743	0.2785	0.0937
LLE	0.3626	0.1808	0.3807	0.2113
t-SNE-CUDA	0.5868	0.4082	0.6610	0.4887
Anchor-t-SNE	**0.6811**	0.**5224**	**0.7483**	**0.5898**
AE	0.5048	0.2825	0.5968	0.3570
VAE	0.1798	0.0660	0.1903	0.0585

KNN classification is used to quantitatively evaluate the quality of feature embedding. The results are shown in Table 4. Anchor-t-SNE have the best accuracy in these methods. This effect might be related to the anchor points used in the dimension reduction process. These anchor points help the points sharing the same class to be clustered closer around each centers. t-SNE-CUDA also has a relatively high accuracy compared to other methods. Nonlinear dimention reduction methods AE and LLE are better than the linear method PCA. Although LLE has the worse visualization effect than PCA, but the result in KNN classification shows that similar points are still be embedded close in LLE. VAE has the lowest accuracy, this might be resulted from the KL divergence term against the standard distribution in its loss function. All points are enforces to be gathered around a center, which decreased its classification accuracy based on the neighboring environment.

References

[1] Thierry Bertin-Mahieux et al. "The Million Song Dataset". In: *Columbia University* (2011). DOI: 10.7916/D8NZ8J07. URL: https://doi.org/10.7916/D8NZ8J07.

[2] David M. Chan et al. "t-SNE-CUDA: GPU-Accelerated t-SNE and its Applications to Modern Data". In: *CoRR* abs/1807.11824 (2018). arXiv: 1807.11824. URL: http://arxiv.org/abs/1807.11824.

[3] Cong Fu et al. "AtSNE: Efficient and Robust Visualization on GPU through Hierarchical Optimization". In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2019, pp. 176–186.

[4] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *CoRR* abs/1512.03385 (2015). arXiv: 1512.03385. URL: http://arxiv.org/abs/1512.03385.

[5] Geoffrey E Hinton and Ruslan R Salakhutdinov. "Reducing the dimensionality of data with neural networks". In: *science* 313.5786 (2006), pp. 504–507.

[6] Yifan Hu, Yehuda Koren, and Chris Volinsky. "Collaborative Filtering for Implicit Feedback Datasets". In: *2008 Eighth IEEE International Conference on Data Mining*. Dec. 2008. DOI: 10.1109/icdm.2008.22. URL: https://doi.org/10.1109/icdm.2008.22.

[7] Diederik P Kingma and Max Welling. *Auto-Encoding Variational Bayes*. 2014. arXiv: 1312.6114 [stat.ML].

[8] Alex Krizhevsky, Geoffrey Hinton, et al. "Learning multiple layers of features from tiny images". In: (2009).

[9] Dawen Liang, Minshu Zhan, and Daniel PW Ellis. "Content-Aware Collaborative Music Recommendation Using Pre-trained Neural Networks." In: *ISMIR*. 2015, pp. 295–301.

[10] Laurens van der Maaten. "Barnes-Hut-Sne". In: *CoRR* (2013). arXiv: 1301.3342 [cs.LG]. URL: http://arxiv.org/abs/1301.3342v2.

[11] Laurens van der Maaten and Geoffrey Hinton. "Visualizing data using t-SNE". In: *Journal of machine learning research* 9.Nov (2008), pp. 2579–2605.

[12] Brian McFee et al. "The million song dataset challenge". In: *Proceedings of the 21st International Conference on World Wide Web*. 2012, pp. 909–916.

[13] Aaron van den Oord, Sander Dieleman, and Benjamin Schrauwen. "Deep content-based music recommendation". In: *Advances in Neural Information Processing Systems 26*. Ed. by C. J. C. Burges et al. Curran Associates, Inc., 2013, pp. 2643–2651. URL: http://papers.nips.cc/paper/5004-deep-content-based-music-recommendation.pdf.

[14] Steffen Rendle et al. "Bpr: Bayesian Personalized Ranking From Implicit Feedback". In: *CoRR* (2012). arXiv: 1205.2618 [cs.IR]. URL: http://arxiv.org/abs/1205.2618v1.

[15] Sam T Roweis and Lawrence K Saul. "Nonlinear dimensionality reduction by locally linear embedding". In: *science* 290.5500 (2000), pp. 2323–2326.

[16] Warren S Torgerson. "Multidimensional scaling: I. Theory and method". In: *Psychometrika* 17.4 (1952), pp. 401–419.

[17] Svante Wold, Kim Esbensen, and Paul Geladi. "Principal component analysis". In: *Chemometrics and intelligent laboratory systems* 2.1-3 (1987), pp. 37–52.