CSCI S-89, Summer 2022

TO: Professor Kurochkin and TAs
FROM: Annetta Zheng
DUE: August 5, 2022
RE: Final Project - Deep Learning for Recommender Systems

Abstract

The use of Recommendation Systems has increased dramatically in the world and has a significant role as the Recommendation Systems help the companies to reduce Costs and increase Revenues, by targeting and fulfilling the business' various goals such as in the AARRR Model, Consumer Segmentation, Ad Targeting, etc. Recommendation Systems are essential for the companies in Marketing which is an advertisement technique meant to deliver ads automatically by using specialized software and algorithms that place ads depending on the user's recent searches. One efficient way to design such algorithm is to predict how a user would rate a given item.

There are mainly 4 types of the recommendations systems, and based on the metrics in our dataset, there are two key methods co-exist tackling this issue: Review-based Collaborative Filtering..

Collaborative filtering (CF) a successful approach commonly used by many recommender systems, which is based on gathering and analyzing data on user's behavior. This includes that people like things similar to other things they like, and things that are liked by other people with similar taste. It is mainly of two types: a) User-User b) Item-Item. Conventional CF-based methods use the ratings given to items by users as the sole source of information for learning to make recommendation. However, the ratings are often very sparse in many applications, causing CF-based methods to degrade significantly in their recommendation performance. To address this sparsity problem, auxiliary information such as item content information may be utilized.

Music data set contains 74347 albums along with the attributes to the albums. Reviews data set contains 1584082 reviews along with the attributes and details of the

reviews in the music department on Amazon. In this project, data pre-processing involves extracting texts from the reviews and encoding the categorical features.

There are a few challenges during the processing procedure: (1) full text analysis requires more than 20G disk space and takes some time; (2) training deep neural networks with LSTM on 126k observations (70% of total observations) takes a few days; (3) the data set is extremely imbalanced and contains a lot of invaluable features.

After preprocessing and cleaning, the Amazon review dataset on Music contains 182826 data samples and 14 useful features; I later train the models on 70% of the dataset and test the performance of the models on the remaining 30% of the dataset.

I use the free bundle the Kaggle Server provides, 4 cores CPU, 1 GPU, 16GB RAM to pre-process data, visualize and verify results.

Several neural network architectures are used when I have to tackle the problem: CNN, autoencoder, and deep neural network model. To mitigate the issues such as misbalancing data and train set overfitting, techniques such as dropout, data augmentation and sampling are also applied. Deeper neural network with transfer learning may help to improve the performance and achieve a better F-score.

Keywords: Autoencoder, Batch Normalization, Convolutional Neural Network
Link to Presentation: https://youtu.be/Q6OW4i40Bkg
Dataset Source: http://deepyeti.ucsd.edu/jianmo/amazon/

Contents

1. Overview

My project uses Keras to try several different deep neural network architectures for this problem with the lecture materials as the guideline.

Recommendation systems also advise users on which items they are more likely to be interested in. For instance, if we are looking to buy a new album on the Internet, our recent searches are based on music suggestions, and soon we will start seeing ads on websites offering discounts on albums or new songs. There are also some good music which may be really good but have not gained popularity since they have not been advertised, so recommendation systems help such items gain popularity by bringing such items to one's notice.

There are a few challenges during the processing procedure: (1) full text analysis requires more than 20G disk space and takes some time; (2) training deep neural networks with LSTM on 126k observations (70% of total observations) takes a few days; (3) the data set is extremely imbalanced and contains a lot of invaluable features.

This project illustrates how deep neural networks with convolution layers have the potential for sentiment analysis, and how auto encoder reduce dimensionality, and finally how deep neural networks contributed to the Recommendations System. Several neural network architectures are used to compare their performance. To mitigate the challenging issues such as imbalancing data and train set overfitting, techniques such as embedding, batch normalization, data augmentation and sampling are applied. The best classification for sentiment is above 91%. The Recommendations System's MAE below 0.4 and MSE below 40.

Practical projects using deep neural networks require domain knowledge to get a good understanding of data. The sheer large size of the data might be challenging, where data pre-processing is crucial and may require tremendous efforts to get it ready for modeling. Working through a project is not a one way process and we need to move back and forth to make the process more efficient (say a better pipeline for feeding train set batches), reduce data dimensionality and training time, and improve model performance by using proper performance metrics.

2. Understanding Data

The first dataset about music contains the following features:

* asin - A unique identifier for each album.
* title - The name of the album.
* price - The price of the album.
* rank - The rank of this album.
* brand - The singer of this album.
* also_buy - The most popular albums the buyers also buy.
* also_view - The most popular albums the buyers also view.

Table 1: Sample Slices from Meta Music Dataset after Cleaning

	title	also_buy	brand	rank	also_view	price	asin	buy_count	view_count
0	Master Collection Volume One	[B000002UEN, B000008LD5, B01J804JKE, 7474034352, B004ZLBTXW, B000008LDH, B000TGKXJ8, B0000AM6KG, B0001XJ372, B001CFLHMC, B0007VO57G, B000005KVI, B0000251O2, B07FDMZ233, B000008LD9, B000008LDL, 000...	john michael talbot	58291.0	[B000002UEN, B000008LD5, 7474034352, B000008LDH, B004ZLBTXW, B0001XJ372, B001CFLHMC, B01J804JKE, B000008LDC, B000TGKXJ8, 0819815802, B0007VO57G, B0000AM6KG, B000CD3LOM, B0000251O2, B000008LDL, B00...	18.99	0001377647	43	33
1	Hymns Collection: Hymns 1 & 2	[5558154950, B00014K5V4]	second chapter of acts	93164.0	[B000008KJ3, B000008KJ0, 5558154950, B000UN8KZE, B000008KJ4]	NaN	0001529145	2	5

The second dataset about reviews has the following features:

* reviewerID - A unique identifier for each reviewer.
* overall - The rating of the given review.
* reviewText - Full text of the given review.
* asin - A unique identifier for each album.
* summary - Summary of the given review.
* reviewTime - The date of review.
* day_of_week - The weekday of review.
* style - The type of the album.
* verified - If is the verified buyer.

Table 2: Sample Slices from Review Dataset after Cleaning

	overall	verified	reviewTime	reviewerID	asin	style	reviewText	summary	day_of_week
0	5	True	2013-12-22	A1ZCPG3D3HGRSS	0001388703	Audio CD	this is great cd full of worship favorites all time great keith green songs his best album by far	great worship cd	6
1	5	True	2013-09-11	AC2PL52NKPL29	0001388703	Audio CD	so creative love his music the words the message some of my favorite songs on this cd should have bought it years ago	gotta listen to this	2
2	5	True	2013-03-02	A1SUZXBDZSDQ3A	0001388703	Audio CD	keith green gone far to early in his carreer left us with these few golden alblums to bless us and let us see from more in sync world veiw or should say the language of the modern world had th...	great approach still gets the message out	5

Pre-processing Data

Music data set contains 74347 albums along with the attributes to the albums. Reviews data set contains 1584082 reviews along with the attributes and details of the reviews in the music department on Amazon. After preprocessing and cleaning, the Amazon review dataset on Music contains 182826 data samples and 14 useful features; I later train the models on 70% of the dataset and test the performance of the models on the remaining 30% of the dataset.

Table 3: Sample Slices from the Dataset after Merging

	reviewerID	overall	reviewText	asin	summary	day_of_week	verified	style	title	brand	rank	price	overall_mean	overall_median	overall_count
0	A1ZCPG3D3HGRSS	5	this is great cd full of worship favorites all time great keith green songs his best album by far	0001388703	great worship cd	6	True	Audio CD	So You Wanna Go Back to Egypt	keith green	203263.0	13.01	4.571429	5.0	28
1	AC2PL52NKPL29	5	so creative love his music the words the message some of my favorite songs on this cd should have bought it years ago	0001388703	gotta listen to this	2	True	Audio CD	So You Wanna Go Back to Egypt	keith green	203263.0	13.01	4.571429	5.0	28
2	A1SUZXBDZSDQ3A	5	keith green gone far to early in his carreer left us with these few golden alblums to bless us and let us see from more in sync world veiw or should say the language of the modern world had th...	0001388703	great approach still gets the message out	5	True	Audio CD	So You Wanna Go Back to Egypt	keith green	203263.0	13.01	4.571429	5.0	28
3	A3A0W7FZXM0IZW	5	keith green had his special comedy style of chirstian music with powerful message came quickly and packaged well would purchase from this vendor again	0001388703	great must have	6	True	Audio CD	So You Wanna Go Back to Egypt	keith green	203263.0	13.01	4.571429	5.0	28

```
Total No. of users that gave rating of 5.0 :  47698

Total No. of Single users that gave rating of 5.0 :  47698
```

Figure 1. **Most Popular 50 Albums**

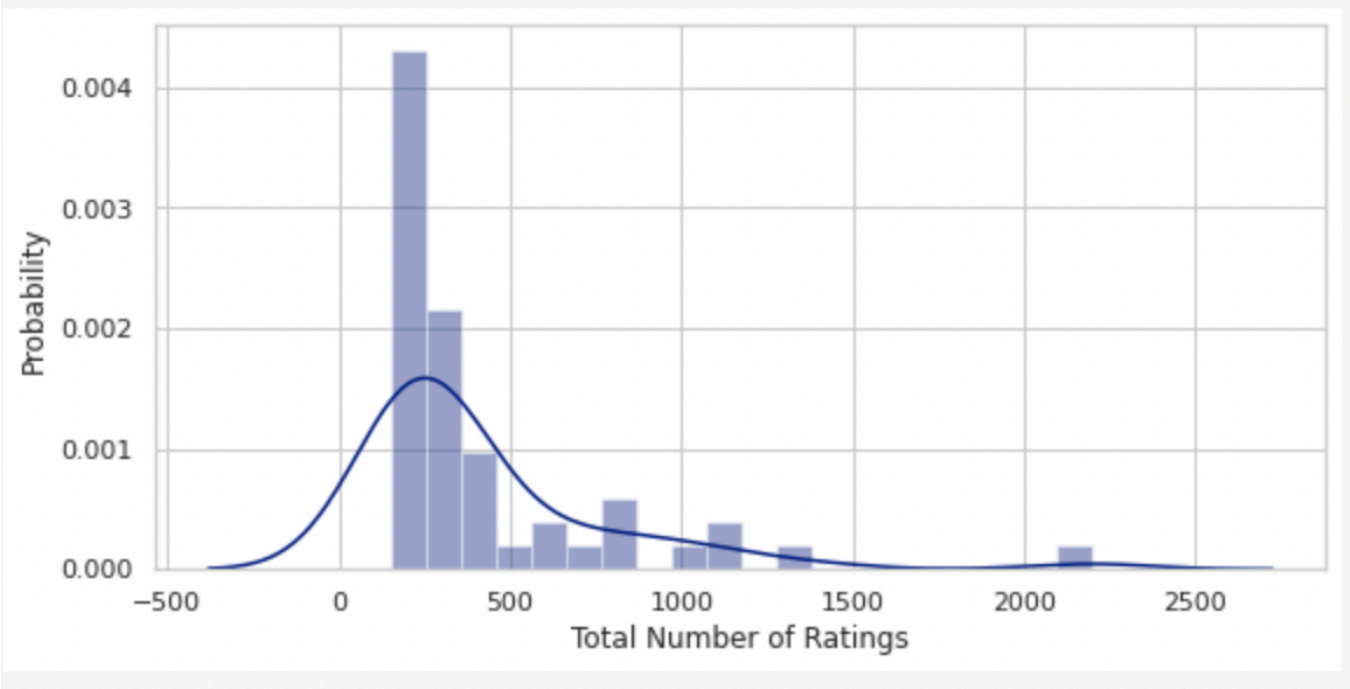


Figure 2. **Mean Rating Distribution of all Albums**

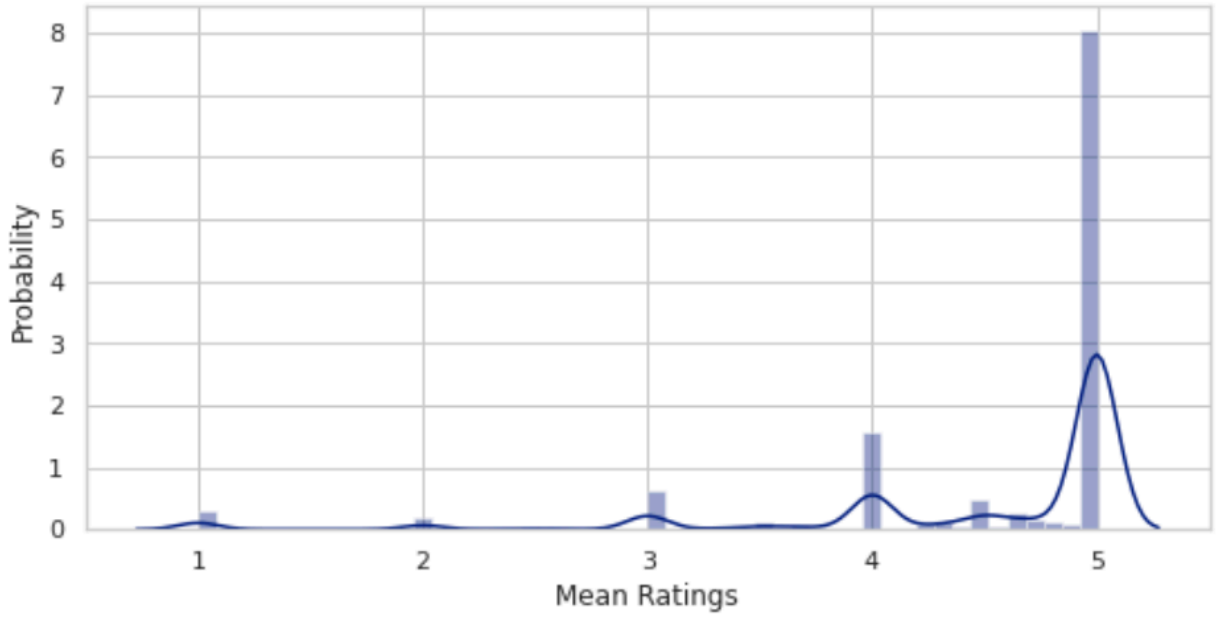


Figure 3. **Difference in Rating Distribution by Non/Verified Users**

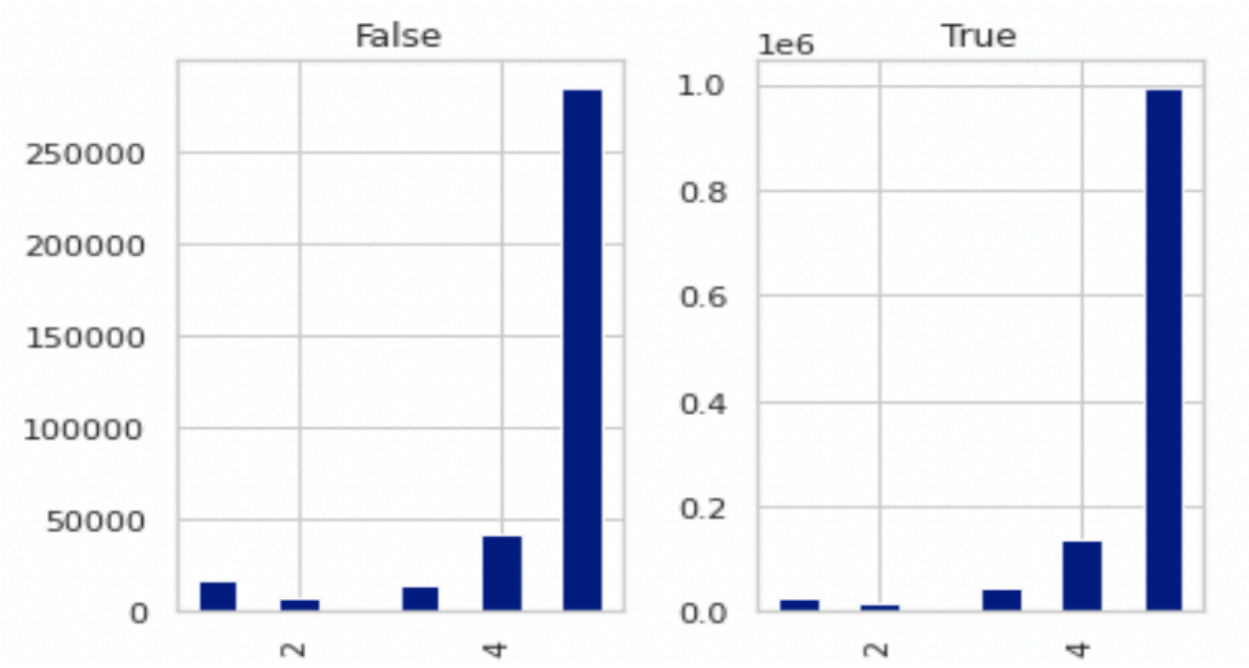
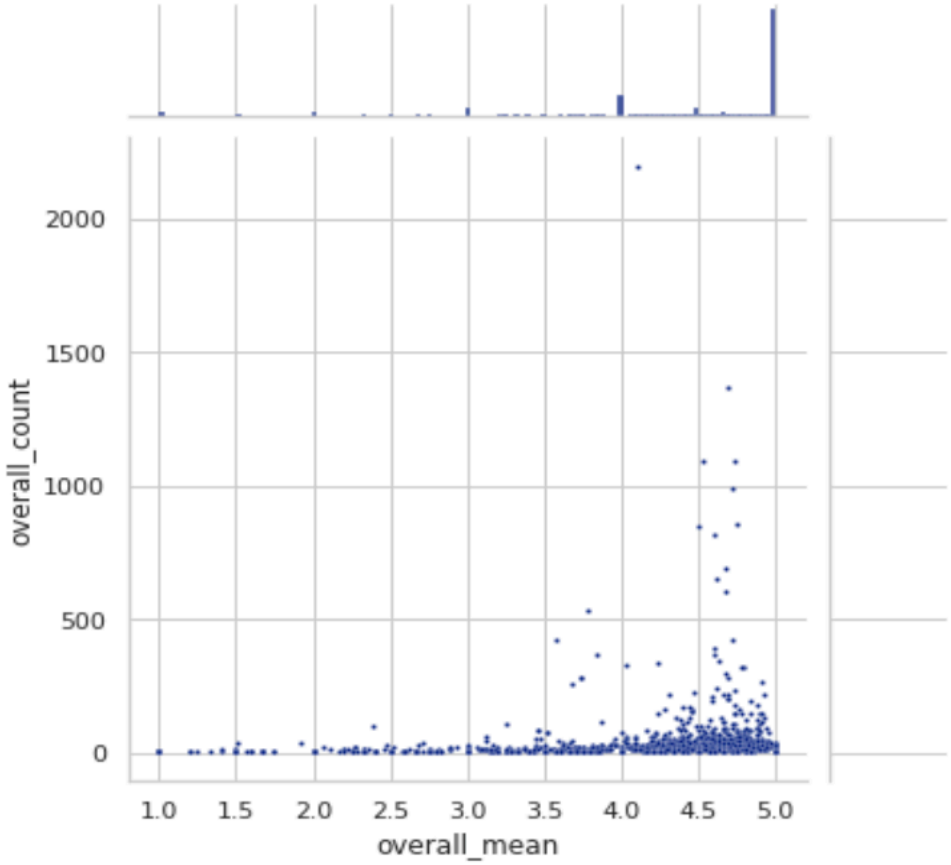


Figure 4. **Mean Rating VS No. Ratings Distributing of all Albums**



EDA – Matrix Recommender

Model without Deep Learning - Test our basic recommender system & Get the Results

```
i_corr = item_corr_matrix['555820690X'].sort_values(ascending=False)
i_corr.dropna(inplace=True)
album_similar_to_i = pd.DataFrame(data=i_corr.values, columns=['Correlation'], index = i_corr.index)
album_similar_to_i.head(10).merge(meta, on = 'asin')[['asin','title','Correlation','brand','overall_mean','overall_count']]
# album_similar_to_i.tail(10).merge(meta, on = 'asin')[['asin','title','Correlation','brand','overall_mean','overall_count']]
```

	asin	title	Correlation	brand	overall_mean	overall_count
0	555820690X	The Ultimate Collection	1.000000e+00	john michael talbot	4.700000	70
1	9714721180	<span class="a-size-medium a-color-secondary a-text-normal"	8.538006e-03	metallica	4.723404	987
2	0760135886	Don Francisco	3.761281e-03	don francisco	4.880000	25
3	0006935257	The Lord's Supper / Be Exalted	3.602278e-03	john michael talbot	4.853659	41
4	9490497045	Live On Earth	2.122866e-14	omnia	4.900000	20
5	B00025HD9E	Bubba Ho-Tep Signature Edition	1.807724e-14	brian tyler	4.833333	12
6	B00025HD9E	Bubba Ho-Tep Signature Edition	1.807724e-14	brian tyler	4.833333	12
7	B000056XHA	What's Up ? Audio 4 Non Blondes	1.353083e-14	4 non blondes	4.666667	12
8	B000056XHA	What's Up ? Audio 4 Non Blondes	1.353083e-14	4 non blondes	4.666667	12

Sentiment Analysis

Figure 5. **The Most Significant Words in 'reviewText'**

Figure 6. **The Most Significant Words in 'summary**

3. Modeling

Sentiment Analysis

The **Sentiment Analysis** model uses 1D convolution plus max pooling blocks followed by one fully connected layers. The output is a classification with a sigmoid activation.

The model during training shall learn the word embeddings from the input text. The total trainable params are 191,860. The output shape is 2 which corresponds to the positive (> 3 stars) and negative label (<= 3 stars) in the 5-star range.

After some tryouts, I embed the summary texts for the top 100 words. Then, I use a Convolutional Neural Network (CNN) as they have proven to be successful at document classification problems. A conservative CNN configuration is used with 128 filters (parallel fields for processing words) and a kernel size of 8 with a rectified linear ('relu') activation function. This is followed by a pooling layer that reduces the output of the convolutional layer by half.

Next, the 2D output from the CNN part of the model is flattened to one long 2D vector to represent the 'features' extracted by the CNN. The back-end of the model is a standard Multilayer Perceptron layers to interpret the CNN features. The output layer uses a sigmoid activation function to output a value between 0 and 1 for the negative and positive sentiment in the review.

```
from keras.layers.convolutional import Conv1D, MaxPooling1D
from keras.layers import Embedding, Dense

# define model
model2 = models.Sequential()

model2.add(Embedding(100, 150, input_length=X.shape[1]))
model2.add(Conv1D(filters=128, kernel_size=8, activation='relu'))
model2.add(MaxPooling1D(pool_size=2))
model2.add(layers.Flatten())
model2.add(Dense(30, activation='relu'))
model2.add(Dense(2, activation='sigmoid'))
print(model2.summary())
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 20, 150)           15000
_________________________________________________________________
conv1d (Conv1D)              (None, 13, 128)           153728
_________________________________________________________________
max_pooling1d (MaxPooling1D) (None, 6, 128)            0
_________________________________________________________________
flatten (Flatten)            (None, 768)               0
_________________________________________________________________
dense (Dense)                (None, 30)                23070
_________________________________________________________________
dense_1 (Dense)              (None, 2)                 62
=================================================================
Total params: 191,860
Trainable params: 191,860
Non-trainable params: 0
```

The 2D output from the CNN part of the model is flattened to one long 2D vector to represent the 'features' extracted by the CNN. The back-end of the model is a standard Multilayer Perceptron layers to interpret the CNN features. The output layer uses a sigmoid activation function to output a value between 0 and 1 for the negative and positive sentiment in the review.

Training the Model

binary_crossentropy and Adam is used to train the model for 10 epochs. See the following code and a screen shot from the training history and plots of loss and accuracy.

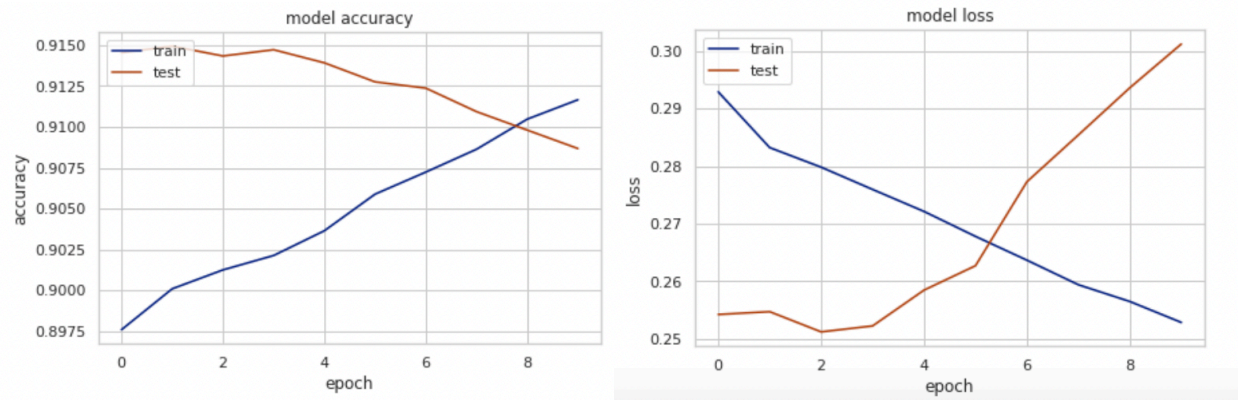
```
model2.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy','binary_crossentropy'])
```

```
Epoch 1/10
7999/7999 [==============================] - 66s 8ms/step - loss: 0.2844 - accuracy: 0.9010
Epoch 2/10
7999/7999 [==============================] - 66s 8ms/step - loss: 0.2735 - accuracy: 0.9053
Epoch 3/10
7999/7999 [==============================] - 66s 8ms/step - loss: 0.2701 - accuracy: 0.9059
Epoch 4/10
7999/7999 [==============================] - 67s 8ms/step - loss: 0.2670 - accuracy: 0.9069
Epoch 5/10
7999/7999 [==============================] - 66s 8ms/step - loss: 0.2633 - accuracy: 0.9081
Epoch 6/10
7999/7999 [==============================] - 65s 8ms/step - loss: 0.2593 - accuracy: 0.9097
Epoch 7/10
7999/7999 [==============================] - 65s 8ms/step - loss: 0.2554 - accuracy: 0.9110
Epoch 8/10
7999/7999 [==============================] - 66s 8ms/step - loss: 0.2522 - accuracy: 0.9125
Epoch 9/10
7999/7999 [==============================] - 67s 8ms/step - loss: 0.2497 - accuracy: 0.9132
Epoch 10/10
7999/7999 [==============================] - 69s 9ms/step - loss: 0.2466 - accuracy: 0.9145
<keras.callbacks.History at 0x7fb53b8d3090>
```

Loss Function and Performance Metrics

Binary crossentropy is used as the loss function to train the neural network. However, since the data is extremely imbananced by classes, additional performance metrics are needed to examine the model performance: precision, sensitivity (or recall), and F-score. The total numbers of observations that are misclassifed are also kept for further analysis by confusion matrix, see the following figure for an illustration followed by the definitions from Keras.

Figure 7. **The Model Accuracy and Model Loss**



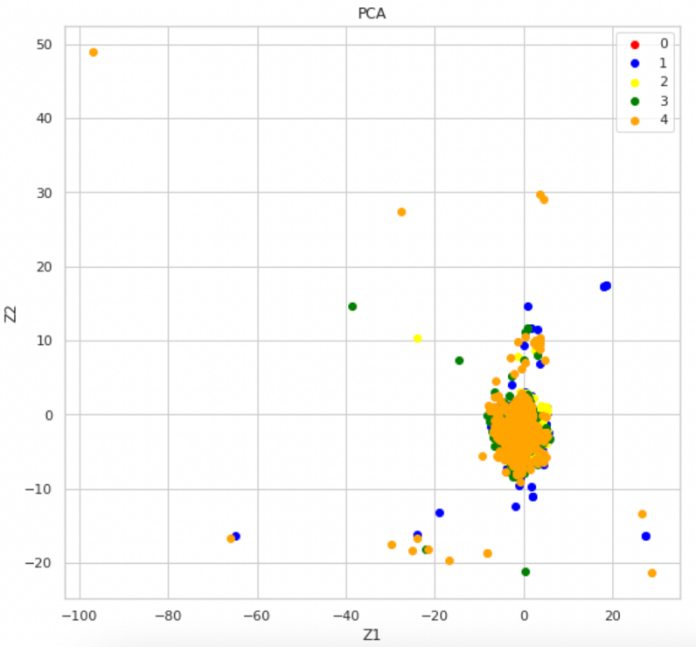
Autoencoder

The ratings are often very sparse in many applications, causing CF-based methods to degrade significantly in their recommendation performance. To address this sparsity problem, auxiliary information such as item content information may be utilized.

Build an autoencoder to reduce dimensionality and reconstruction X_Xtd for later Testing

- Preprocess:
  Encoding Categorical Variables
- Input:
  Standardized X_frame with 70 features

Figure 8. **PCA**



Training the Model

SGD is used to train the model for 10 epochs. See the following code and a screen shot from the training history and plots of loss and accuracy.

```
from tensorflow import keras
from tensorflow.keras import optimizers
from keras import models, layers
```

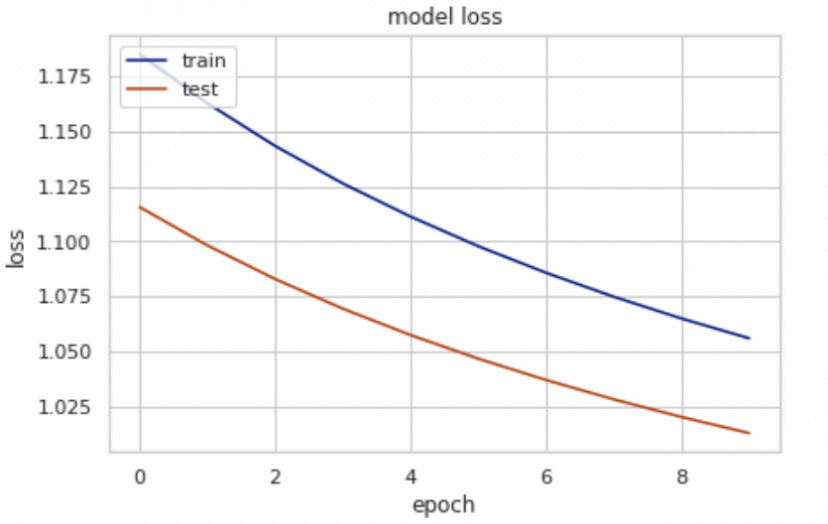
```
encoder = models.Sequential([layers.Dense(5, input_shape=[70,])])
decoder = models.Sequential([layers.Dense(70, input_shape=[5])])
autocoder = models.Sequential([encoder, decoder])
print(autocoder.summary())

autocoder.compile(loss = 'mse', optimizer = optimizers.SGD(lr = 0.0001))


history = autocoder.fit(X_train, X_train, epochs = 10,
                        validation_data=(X_test, X_test))
codings = encoder.predict(X_train)
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
sequential_27 (Sequential)   (None, 5)                 355
_________________________________________________________________
sequential_28 (Sequential)   (None, 70)                420
=================================================================
Total params: 775
Trainable params: 775
Non-trainable params: 0
_________________________________________________________________
None
Epoch 1/10
5714/5714 [==============================] - 9s 2ms/step - loss: 1.2110
Epoch 2/10
5714/5714 [==============================] - 9s 2ms/step - loss: 1.1726
Epoch 3/10
5714/5714 [==============================] - 9s 2ms/step - loss: 1.1418
Epoch 4/10
5714/5714 [==============================] - 10s 2ms/step - loss: 1.1164
Epoch 5/10
5714/5714 [==============================] - 10s 2ms/step - loss: 1.0952
Epoch 6/10
5714/5714 [==============================] - 10s 2ms/step - loss: 1.0771
Epoch 7/10
5714/5714 [==============================] - 9s 2ms/step - loss: 1.0615
Epoch 8/10
5714/5714 [==============================] - 9s 2ms/step - loss: 1.0478
Epoch 9/10
5714/5714 [==============================] - 9s 2ms/step - loss: 1.0358
Epoch 10/10
5714/5714 [==============================] - 9s 2ms/step - loss: 1.0251
```

Figure 9. **Model Loss**



Collaborative Filtering - Deep Learning

- Build a NN to predict the Ratings of each record
  - Preprocess:
    - Encoding Categorical Variables
  - Input:
    - Standardized X_frame with 70 features
  - Model: NN
  - Output: Score 1-5

Training the Model

SGD is used to train the model for 10 epochs. See the following code and a screen shot from the training history and plots of loss and accuracy.

```
model = models.Sequential()
model.add(Dense(256, activation='relu',
                input_dim=X_train.shape[1], activity_regularizer=regularizers.l1(0.01)))
model.add(BatchNormalization())
model.add(Dense(128, activation='relu'))
model.add(BatchNormalization())
model.add(Dense(64, activation='relu'))
model.add(BatchNormalization())
model.add(Dense(32, activation='relu'))
model.add(BatchNormalization())
model.add(Dense(8, activation='relu'))
model.add(BatchNormalization())
model.add(Dense(1))

model.compile(loss='mse', optimizer='adam', metrics=['mae'])
print(model.summary())
```

Mitigate Overfitting

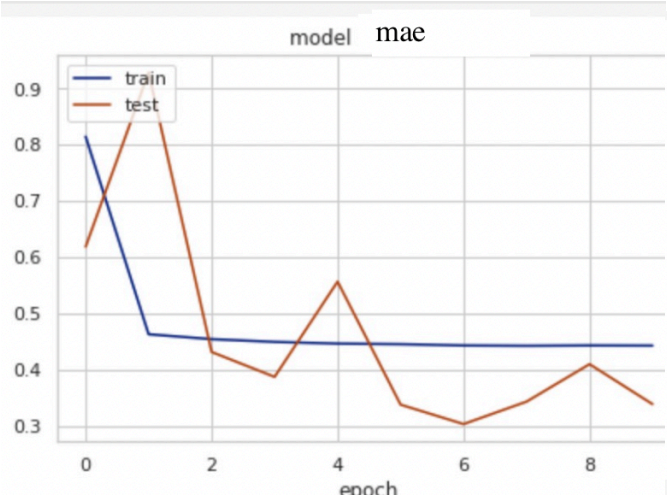
To fight against overfitting, we can try techniques such as Batch Normalization in the fully connected portion of network.

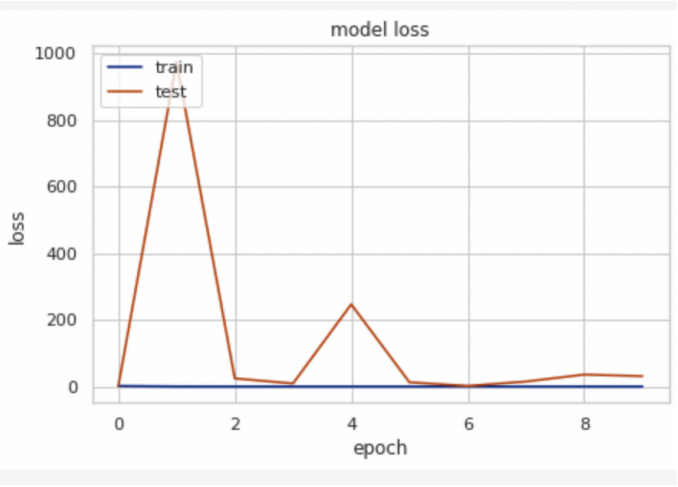
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_34 (Dense)             (None, 256)               18176
_________________________________________________________________
batch_normalization_10 (Batc (None, 256)               1024
_________________________________________________________________
dense_35 (Dense)             (None, 128)               32896
_________________________________________________________________
batch_normalization_11 (Batc (None, 128)               512
_________________________________________________________________
dense_36 (Dense)             (None, 64)                8256
_________________________________________________________________
batch_normalization_12 (Batc (None, 64)                256
_________________________________________________________________
dense_37 (Dense)             (None, 32)                2080
_________________________________________________________________
batch_normalization_13 (Batc (None, 32)                128
_________________________________________________________________
dense_38 (Dense)             (None, 8)                 264
_________________________________________________________________
batch_normalization_14 (Batc (None, 8)                 32
_________________________________________________________________
dense_39 (Dense)             (None, 1)                 9
=================================================================
Total params: 63,633
Trainable params: 62,657
Non-trainable params: 976
```

Training the Model

Adam is used to train the model for 10 epochs. See the following code and a screen shot from the training history and plots of loss and mae.

```
Epoch 1/10
4000/4000 [==============================] - 24s 5ms/step - loss: 2.0719 - mae: 0.8148 - val_loss: 1.2415 - val_mae: 0.6192
Epoch 2/10
4000/4000 [==============================] - 21s 5ms/step - loss: 0.6553 - mae: 0.4639 - val_loss: 975.3854 - val_mae: 0.928
Epoch 3/10
4000/4000 [==============================] - 22s 5ms/step - loss: 0.6283 - mae: 0.4553 - val_loss: 24.8035 - val_mae: 0.4325
Epoch 4/10
4000/4000 [==============================] - 21s 5ms/step - loss: 0.6147 - mae: 0.4504 - val_loss: 9.6386 - val_mae: 0.3883
Epoch 5/10
4000/4000 [==============================] - 22s 5ms/step - loss: 0.6086 - mae: 0.4474 - val_loss: 247.2344 - val_mae: 0.557
Epoch 6/10
4000/4000 [==============================] - 22s 5ms/step - loss: 0.6024 - mae: 0.4463 - val_loss: 13.2175 - val_mae: 0.3390
Epoch 7/10
4000/4000 [==============================] - 21s 5ms/step - loss: 0.5964 - mae: 0.4440 - val_loss: 2.3107 - val_mae: 0.3045
Epoch 8/10
4000/4000 [==============================] - 21s 5ms/step - loss: 0.5942 - mae: 0.4434 - val_loss: 15.5325 - val_mae: 0.3444
Epoch 9/10
4000/4000 [==============================] - 22s 5ms/step - loss: 0.5936 - mae: 0.4440 - val_loss: 36.3110 - val_mae: 0.4109
Epoch 10/10
4000/4000 [==============================] - 21s 5ms/step - loss: 0.5907 - mae: 0.4437 - val_loss: 31.6674 - val_mae: 0.3398
```





Result

Getting the Top10 Recommended Albums for the first review in our dataset

- Use the Reconstructed X_std and X_std, we get the Top 10 Recommended Albums Indices
- Then iloc these Indices from the Dataset

```
recommended_book_ids = (-predictions).argsort()[:5]
recommended_book_ids2 = (-predictions_auto).argsort()[:5]
```

```
print(recommended_book_ids)
print(recommended_book_ids2)
print(predictions[recommended_book_ids])
print(predictions[recommended_book_ids2])
```

```
[181283 178466 125849 134166 136761]
[131380  95673 151613 143370  92818]
[5.182131  5.182131  5.1768074 5.173042  5.168805 ]
[4.9613533 4.915155  4.9285164 4.8893046 4.201339 ]
```

+ Code + Markdown

```
res = dl_train.iloc[recommended_book_ids]['title'].tolist() + dl_train.iloc[recommended_book_ids2]['title'].tolist()
print(res)
```

```
["'Destroy &amp; Conquer Tasse' bLACK LABEL sOCIETY Official boxed Mug", 'PLANET HENDRIX', 'The Beatles John Lennon Help! Cord
/ Corduroy &amp; Moleskin Hat Cap (Medium, Black Moleskin),56646" />', "Emi - Elvis Presley Magnet Let's Face It", 'One Girl To
o Late', 'Excited', 'Lone Ranger', 'Just For Fun', 'Mozart : Symphony No.40 - Schubert: Symphony No.8 &quot;Unfinished&quot;',
'Somewhere In America']
```

4. Conclusion and Lessons Learned

This project illustrates how deep neural networks with convolution layers have the potential for sentiment analysis, and how auto encoder reduce dimensionality, and finally how deep neural networks contributed to the Recommendations System. Several neural network architectures are used to compare their performance. To mitigate the challenging issues such as imbalancing data and train set overfitting, techniques such as embedding, batch normalization, data augmentation and sampling are applied. The best classification for sentiment is above 91%. The Recommendations System's MAE below 0.4 and MSE below 40. However, since there is very few records of recurring buyer, it is barely possible for us to evaluate the accuracy of the Recommendations System.

Practical projects using deep neural networks require domain knowledge to get a good understanding of data. The sheer large size of the data might be challenging, where data pre-processing is crucial and may require tremendous efforts to get it ready for modeling. Working through a project is not a one way process and we need to move back and forth to make the process more efficient (say a better pipeline for feeding train set batches), reduce data dimensionality and training time, and improve model performance by using proper performance metrics.

References

1. Justifying recommendations using distantly-labeled reviews and fined-grained aspects (Ni, Li, McAuley, 2019)

Appendix

A. Hardware

Kaggle Server provides, 4 cores CPU, 1 GPU, 16GB RAM.

B. Jupyter Notebook Files

The following are all Jupyter Notebook files that work from pre-processing data until training deep neural networks step by step. Some operations are not coded inside Python, such as downloading raw files from LUNA16 challenge, moving files from local computer to AWS EC2 instance, etc.

- Step1 Read and Preprocesses Meta Dataset: drop irrelevant and null columns, perform texts cleaning, calculate useful numerical features
- Step2 Read and Preprocesses Reviews Dataset: drop irrelevant and null columns, perform texts cleaning, calculate useful numerical features, extract categorical features
- Step3 EDA: understand data and make a few visualizations
- Step4 Merge the Datasets: merge Meta file and Reviews file
- Step5 Understand Recommender: vectorizing our features, create item-item corr matix, get and test a basic Recommender System
- Step6 Train Sentiment Analysis CNN: tokenize texts, get X_matrix for text, create performance plots
- Step7 Train Autoencoder: perform PCA, reduce text dimensionality, get a reconstructed X_std for later use, create performance plots
- Step8 Train Recommender with DNN: train with original data set that is replaced with the text X_matrix, create performance plots
- Step9 Test for Recommender : use the reconstructed X_std and X_std, we get the top N recommended albums indices, then iloc these indices from the dataset

Step4 Slice, Step4 Slice Augment Positive Local, and Step4 Slice Sampling Negative: create 3D chunk for each candidate, augment chunks for nodules, generate balanced data set.

- Step5 Train, Step5 Train Aug: baseline models trained by the original train set and by the augmented balanced train set.
- Step6 Train Dropout ClassWeights, Step6 Train Dropout Aug: models with dropout layers, trained by the original train set and by the augmented balanced train set.
- Step7 Train Inception, Step7 Train Inception Aug: models with inception black, trained by original train set and by augmented balanced train set.
- Step8 Train RNNConv, Step8 Train RNNConv Aug: RNN model built upon convolution layers, trained by the original train set and by the augmented balanced train set.
- Step9 AnalysisPlots: create performance plots based on the training history of all eight models.
