COMP4434: Big Data Analytics

ZHANG Caiqi 18085481d

The Hong Kong Polytechnic University

1 Introduction

PolyTube is on online media platform which provides online teleplay services. In PolyTube's database, there are all the general information about each movie, such as name, type, genre, number of episodes and the numbers of people who have rated the teleplays. It also collected massive amounts of users' data, including their movie records and their rating.

In this project, we will complete two tasks. First, design a prediction model to predict the rating of recently published teleplays. Second, design a recommendation system to provide personalized recommendation services. In addition to regular work, this project also tried the following aspects and achieved good results.

- Many effective visualization methods are used to analyze the data.
- We used the word embedding methods to generate new features to predict the rating.
- Ten regression models have been applied to compare their performance.
- We applied the neural networks in both rating prediction and teleplays recommendation.

1.1 Presentation

The presentation link is https://youtu.be/b8kU4oHszlU.

2 Data analysis

The training set of Task 1 is named "Teleplay.csv". The file contains the history information of 10204 teleplays, including teleplay_id, name, genre, length/type, episodes, rating and members. By using the **pandas** [1]**, seaborn** [2]**, and matplotlib** [3] libraries in Python, we can get more detailed information about these features.

As to the training set of Task 2, which is "Rating.csv", it consists of the ratings of 73516 reviewers. If the user watched a teleplay but not rated it, the rating will be -1.

2.1 Outlier detection

We first detect null values for each columns. The results are shown in the following table. For the columns which contain null values, we will use different methods to deal with them in the next section.

teleplay id	False
name	False
genre	True
type	True
episodes	False
rating	True
members	False

2.2 Columns statistics

- teleplay_id: it is the unique id of type int to identify a teleplay. In some scenario, it can be used to link the teleplay with other information. There are totally 10204 unique ids.
- name: it is the origin name of the teleplay of type string. Some teleplays have very similar name.
- genre: it contains the categories that the teleplay may be related to. It totally contains 43 categories, such as Action, Adventure, Cars, Comedy, Dementia, Demons etc. Some teleplays do not belong to any one of the categories. From the following picture, we can find that Comedy, Action, and Adventure are the top 3 hot topics.

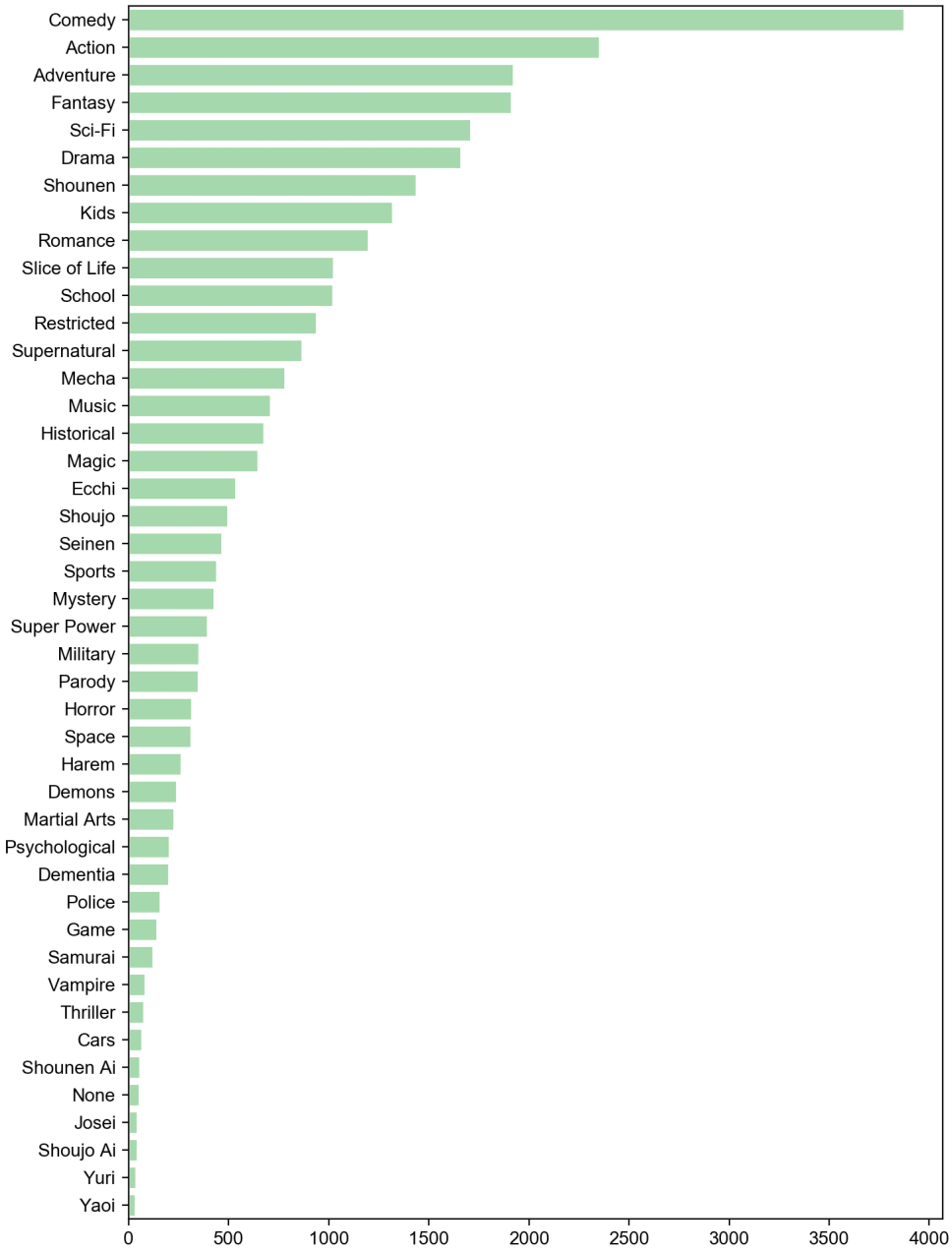

Figure 1: Genre counts

- length/type: it refers to the length of the teleplay. It totally has 6 classes which are short, medium, long, special, ONA and music. Some teleplays do not belong to any type or the type value is missing. From the Figure 2, we can find that 31% of the teleplays are in medium length and 27% are short. ONA and music are only in a very small percentages.

- episodes: it refers to the total number of episodes of a teleplay. Some teleplays have an "Unknown" value in this attributes. After eliminating the unknown values, we can get the following graph. We manually divided the episodes into several ranges and we

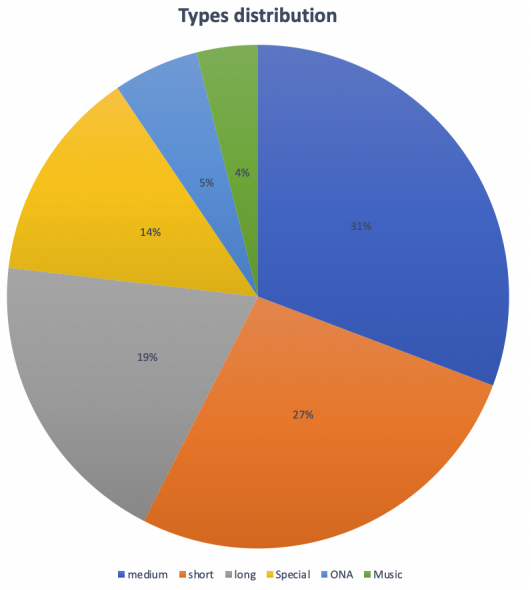

Figure 2: Types distribution

can find that the episodes of the teleplays vary a lot but almost concentrate on one to one hundred episodes. Details are in Figure 3.

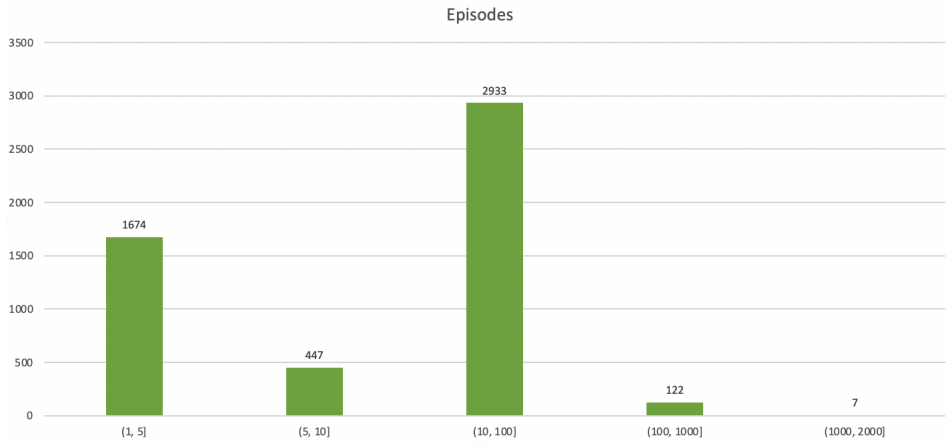

Figure 3: Episodes distribution

- rating: the score of a teleplay with maximum 10. Some teleplays in the data do not have rating. We can find that over 90% of the teleplays have the rating between 4 to 8. Only very few teleplays have extremely high or low scores. Figure 5 shows the top 10 teleplays with the highest ratings.

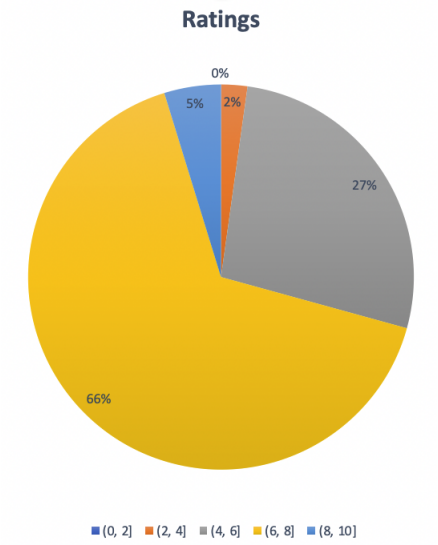

Figure 4: Ratings distribution

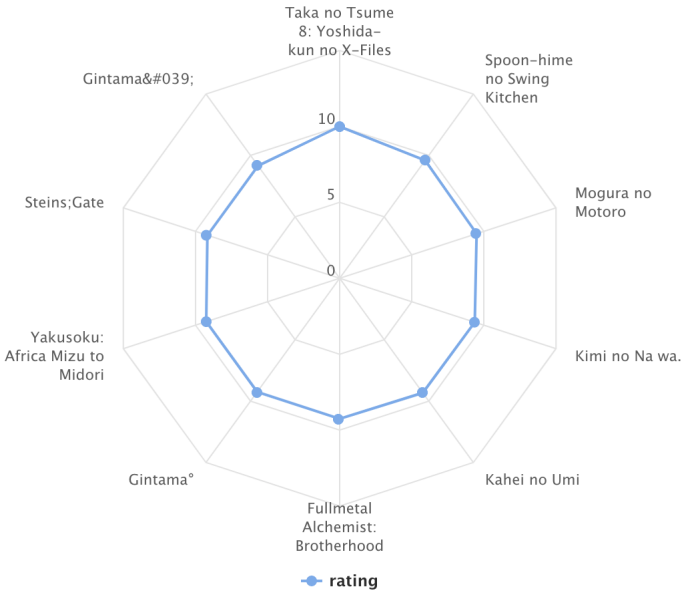

Figure 5: Top 10 teleplays

- members: the number of registered users that have rated the teleplay. The number of members can reflect the popularity of a teleplay. Figure 6 shows the members distribution of the teleplays. Some teleplays are very popular and have over 100,000 members, while on the other hand, some teleplays have very few members. In some sense, the popularity can reflect the rating of an teleplay, however, it is not absolute. We may also find many teleplays with around several hundreds of members have very high ratings.

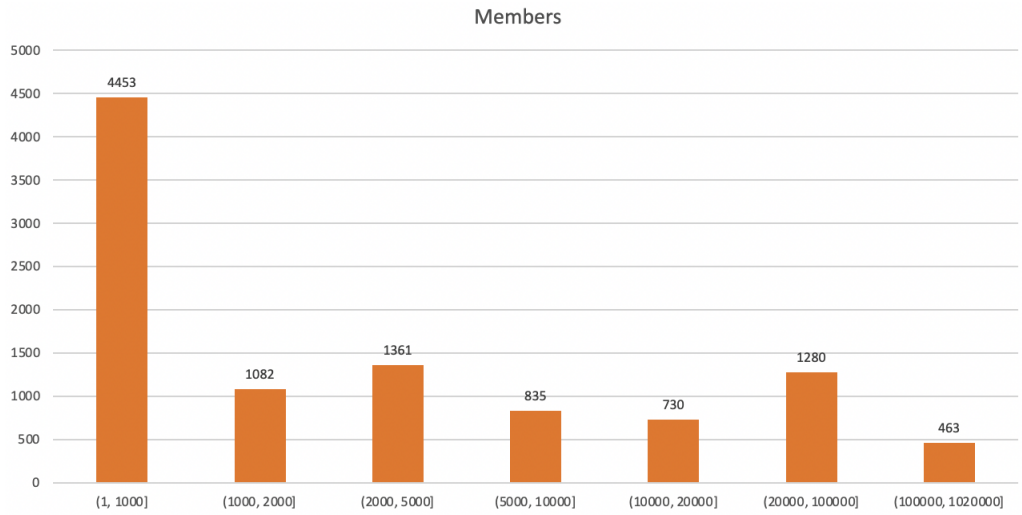

Figure 6: Members distribution

2.3 Users' ratings

In Task 2, we are going to use the users' ratings to give recommendations. Therefore, it is helpful to know more about the users' rating information. For every user, we made the following graph to show how many teleplays they rated. From the graph we can find that most people do not rate many teleplays. Most of the people rated less than 100 teleplays.

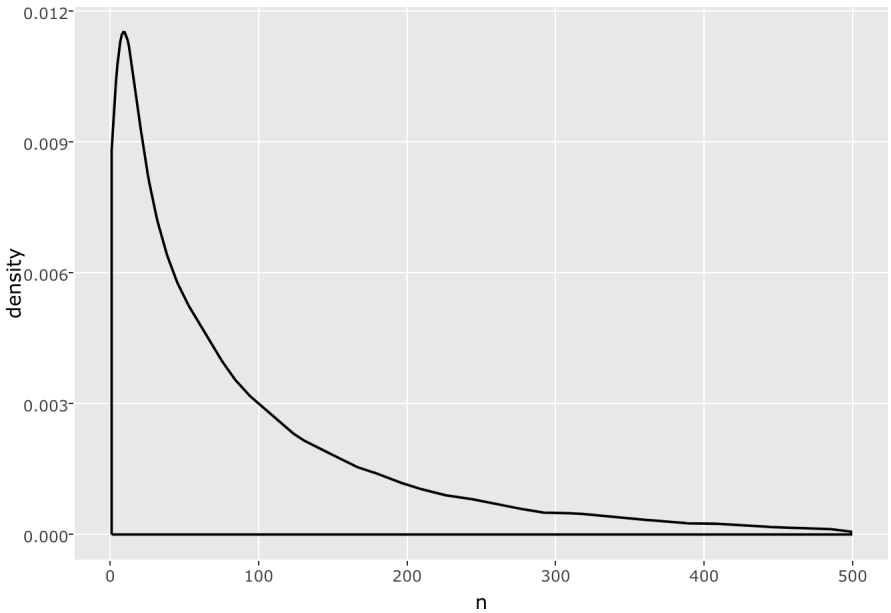

Figure 7: Users' ratings

2.4 The taste of user 53698

As we are going to design a model to recommend teleplays to the user 53698, it would be helpful if we can know more about his tastes of the teleplays. His preferences can also be used to evaluate the efficiency of the recommendation system. We selected all the teleplays which were rated as 10 by user 53698 and extracted the teleplays' genres.

To visualize the statistics, we made the following word cloud using the Python library **wordcloud** [4].

```
from wordcloud import WordCloud
```

The size of the words reflects the frequency of the words appearing in the genres of the user 53698's favorite teleplays. From the word cloud, we can find that user 53698 prefers the teleplays under the drama, romance and supernatural the most.

Figure 8: Word cloud of user 53698's preference

3 Data preprocessing

After analysing the raw data, the next step is to change the raw data into the data that can be trained by the machine learning models. The preprocessing work mainly contains three parts: removing null values, feature engineering and word embedding.

- Remove null values: If there is no rating of an teleplay, we will remove the entire row. If there is Unknown in the episodes, we will replace it as the average of the episodes. If there is no value in the genre, as it will not influence the training result, we will ignore it.

- Feature engineering: We should also transform some attribute into usable format for the future training. First, for the "type" feature, we will use 0, 1, 2, 3, 4, 5 to represent short, medium, long, special, ONA and music respectively. Second, for the "genre" feature, we will adopt the one hot coding, which replaces it by using each genre as a single feature, i.e., introduce the other 43 attributes. And we use 1 or 0 to indicate whether a specific contains this category.

- Word embedding: based on the finding that some teleplays which have very similar names also have very similar ratings, we surmise that the rating may also have some relationship with the teleplay names. It can also be explained as some popular IP may have many seasons, for example Gintama, Naruto, Conan and so on. Therefore, we will try to convert the teleplay names into word vectors.

For all the numerical values, we will also do the normalization using the formula:

$$X = (X - X.min())/(X.max() - X.min()). \tag{1}$$

We used the BERT [5], which stands for Bidirectional Encoder Representations from Transformers, to encode the teleplay names. Every sentence will be transformed to a vector with length 768.

After the preprocessing, the data will be as shown in Figure 9 and Figure 10:

teleplay_id	name	genre	type	episodes	rating	members	Comedy	Josei	Super Power	Yaoi	Fantasy
32281	Kimi no Na wa.	Drama, Romance, School,	2	1	9.37	200630	0	0	0	0	0
28977	Gintama°	Action, Comedy, Historical,	1	51	9.25	114262	1	0	0	0	0
9253	Steins;Gate	Sci-Fi, Thriller	1	24	9.17	673572	0	0	0	0	0
9969	Gintama'	Action, Comedy, Historical,	1	51	9.16	151266	1	0	0	0	0
32935	Haikyuu!!: Karasuno Koukou VS	Comedy, Drama, School,	1	10	9.15	93351	1	0	0	0	0
11061	Hunter x Hunter (2011)	Action, Adventure, Shounen,	1	148	9.13	425855	0	0	1	0	0
820	Ginga Eiyuu Densetsu	Drama, Military, Sci-Fi, Space	0	110	9.11	80679	0	0	0	0	0
15335	Gintama Movie: Kanketsu-he	Action, Comedy, Historical,	2	1	9.1	72534	1	0	0	0	0
15417	Gintama' : Enchousen	Action, Comedy, Historical,	1	13	9.11	81109	1	0	0	0	0
4181	Clannad: After Story	Drama, Fantasy, Romance,	1	24	9.06	456749	0	0	0	0	1

Figure 9: Training data

4 Data preprocessing by MapReduce

MapReduce is a programming model and an associated implementation for processing and generating big data sets with a parallel, distributed algorithm on a cluster. The MapReduce methods are with larger scalability, better cost-efficiency, better flexibility, and faster speed. In this project, to pre-process data in a parallel way, we designed the MapReduce algorithms for the tasks.

For Task 1, we used the MapReduce to deal with the null values, transfer the tags into one-hot code and also do some other data cleaning work. Because the procesure is simple,

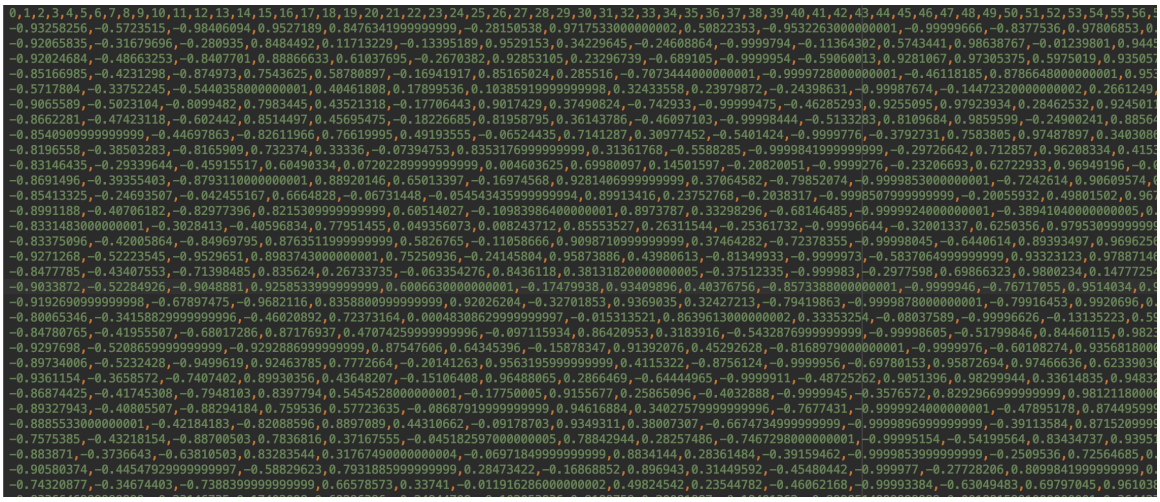

Figure 10: Word embeddings: each teleplay name is converted to a vector of length 768

we actually do not need reduce() step.

Algorithm 1: MapReduce Task 1

Input: Raw data of the teleplays
Output: Training data

```
Def Map():
    for each line do
        fill in null values;
        change type to integer;
        expand genre as one-hot code;
        other processing;
    end
    return line;

Def Reduce():
    return 1;
```

For Task 2, the average rating of each teleplay can be computed by MapReduce based on the rating of each user to each teleplay in rating.csv. The average rating computed from rating.csv can be attached as a new feature to the corresponding teleplay in Teleplay.csv.

To calculate an average, we need two values for each group: the sum of the values that we want to average and the number of values that went into the sum. These two values can be calculated on the reduce side very trivially, by iterating through each value in the set and adding to a running sum while keeping a count. After the iteration, simply divide the sum

by the count and output the average.

Algorithm 2: MapReduce Task 2

Input: Raw data of the users' ratings
Output: Training data

```
Def Map():
    for each line do
        if rating != -1 then
            emit(teleplay_id, rating);
        end
    end

Def Reduce():
    //key: teleplay_id
    //value: a list of ratings
    sum = 0;
    for each line do
        sum += rating;
    end
    emit(teleplay_id, sum/sizeof(values));
```

5 Model design and implementation of Task 1

5.1 Feature selection

In this dataset, we have very limited features so we need to make the full use of them. Firstly, the members are a key feature for rating prediction because that with more viewers, the teleplay is more likely to be of high score. Meanwhile, the type, episodes, and 43 genres may also related to the rating. About the teleplay name, as we discussed that there are some popular IPs such as Gintama, Naruto, Conan, it can also be used as a training feature.

5.2 Model selection

We will use the following models [6] to do the preliminary experiments. **The basic linear regression model is written from scratch.** For the other models, we used the *sklearn* package. Here I will briefly introduce each model.

- **Decision tree:** Decision trees are a non-parametric supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features. A tree can be seen as a piecewise constant approximation.

```
from sklearn import tree
model_DecisionTreeRegressor = tree.DecisionTreeRegressor()
```

- **Linear regression (L1)**: The linear regression with L1 regularization follows what we learned in the class.

- **Linear regression (L2)**: The linear regression with L2 regularization follows what we learned in the class.

- **Support vector machine (SVM):** The objective of the support vector machine algorithm is to find a hyperplane in an N-dimensional space(N — the number of features) that distinctly classifies the data points. It can be used in both classification and regression. The advantages of support vector machines is that it is effective in high dimensional spaces.

  ```
  from sklearn import svm
  model_SVR = svm.SVR()
  ```

- **K-nearest neighbors (KNN):** In pattern recognition, the k-nearest neighbors algorithm is a non-parametric method used for classification and regression. In both cases, the input consists of the k closest training examples in the feature space. In k-NN regression, the output is the property value for the object. This value is the average of the values of k nearest neighbors.

  ```
  from sklearn import neighbors
  model_KNeighborsRegressor = neighbors.KNeighborsRegressor()
  ```

- **Random Forest:** Random forest is a commonly-used machine learning algorithm, which combines the output of multiple decision trees to reach a single result. Its ease of use and flexibility have fueled its adoption, as it handles both classification and regression problems.

  ```
  from sklearn import ensemble
  model_RandomForestRegressor = ensemble.RandomForestRegressor(n_estimators=20)
  ```

- **AdaBoost:** AdaBoost is one of the first boosting algorithms to be adapted in solving practices. Adaboost helps to combine multiple "weak classifiers" into a single "strong classifier". The weak learners in AdaBoost are decision trees with a single split, called decision stumps. AdaBoost works by putting more weight on difficult to classify instances and less on those already handled well. It can be used for both classification and regression problem.

  ```
  from sklearn import ensemble
  model_AdaBoostRegressor = ensemble.AdaBoostRegressor(n_estimators=50)
  ```

- **Gradient Boosted Regression Trees (GBRT):** Gradient Boosting is similar to AdaBoost in that they both use an ensemble of decision trees to predict a target label. However, unlike AdaBoost, the Gradient Boost trees have a depth larger than 1.

```
from sklearn import ensemble
model_GradientBoostingRegressor = ensemble.GradientBoostingRegressor
(n_estimators=100)
```

- **Bagging:** Bootstrap aggregating, also called bagging, is a machine learning ensemble meta-algorithm designed to improve the stability and accuracy of machine learning algorithms used in statistical classification and regression. It also reduces variance and helps to avoid overfitting. Although it is usually applied to decision tree methods, it can be used with any type of method.

```
from sklearn.ensemble import BaggingRegressor
model_BaggingRegressor = BaggingRegressor()
```

- **ExtraTrees:** The Extra Trees algorithm works by creating a large number of unpruned decision trees from the training dataset. Predictions are made by averaging the prediction of the decision trees in the case of regression or using majority voting in the case of classification.

```
from sklearn.tree import ExtraTreeRegressor
model_ExtraTreeRegressor = ExtraTreeRegressor()
```

As the mean square error (MSE) will be used to evaluate the result, we will use MSE to design the loss function. We divide the teleplays in to train and test set in 4: 1. After 10000 iterations, the results in the test set are as follows:

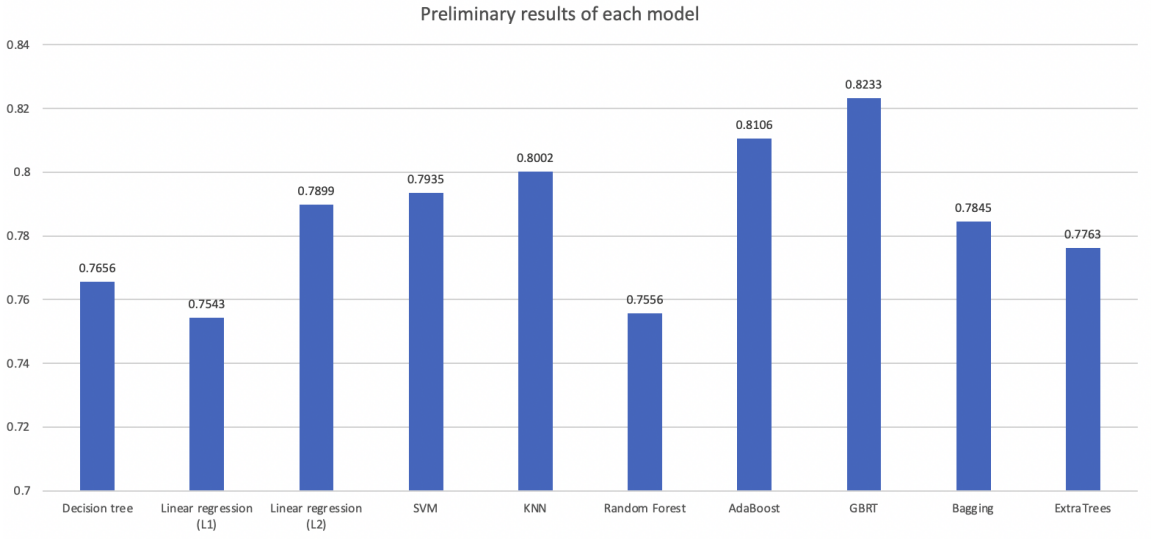

Figure 11: Preliminary experiments

From the Figure 8, we can see that using Linear Regression with L1 Regularization can achieve better performance. We will then do the further hyperparameters adjusting focusing on this one model.

5.2.1 Hyper-parameters of linear regression with L1 regularization

The main hyper-parameters in Linear Regression with L1 Regularization are the learning rate α and constant λ.

We first set λ to 0 and try to find a appropriate α. We tried several α and from the experiment results we adopt $\alpha = 0.01$.

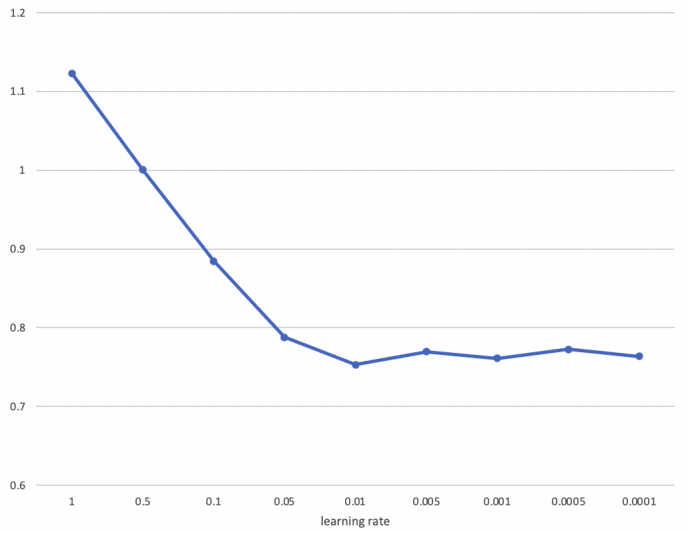

Figure 12: Learning rate experiments

After fixing the α, we test the value of λ.

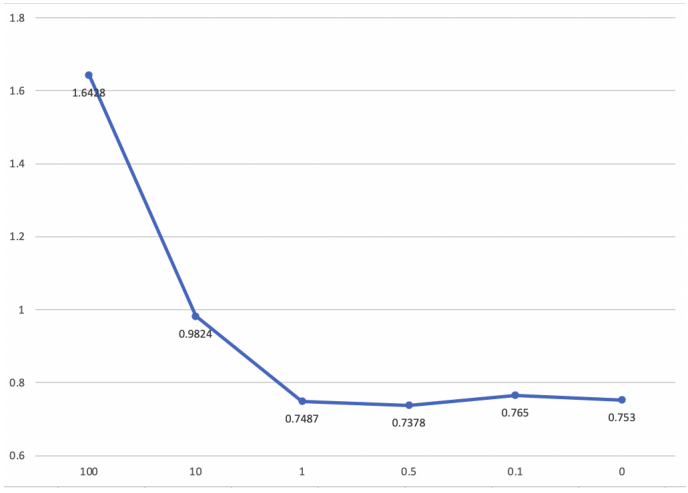

Figure 13: λ experiments

Finally, we choose $\alpha = 0.01$ and $\lambda = 0.5$ in Linear Regression with L1 Regularization. The MSE is around 0.7378.

5.3 Neural network

We also implemented the deep neural network (DNN) to predict the ratings. The incentive of using neural network is that the neural networks work better at predictive analytics because of the hidden layers. Linear regression models use only input and output nodes to make predictions. Neural network also use the hidden layer to make predictions more accurate. That's because it "learns" the way a human does.

Each neuron takes into consideration a set of input values. Each of them gets linked to a "weight", which is a numerical value that can be derived using either supervised or unsupervised training such as data clustering, and a value called "bias". The network chooses from the answer produced by a neuron based on its' weight and bias.

The model consists of 5 fully connected layers using ReLu as the activation function.

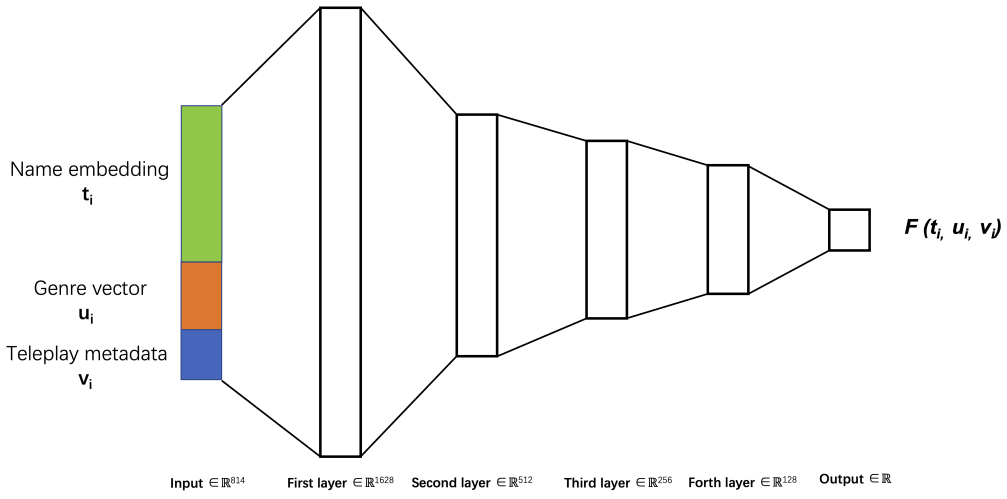

Figure 14: Network of Task1

To choose the best learning, we did the following experiments and finally chose the learning rate as 0.005. We may also find that the MSE is lower than using linear regression.

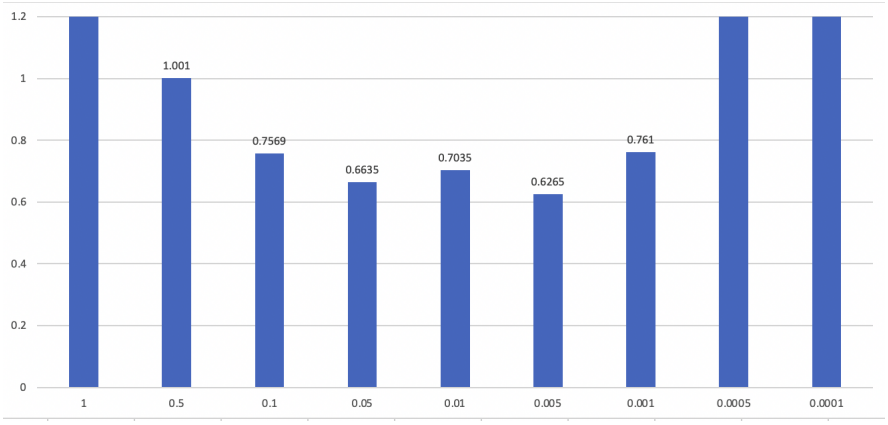

Figure 15: Prediction

6 Result of Task 1

As the DNN reached better performance, we will use it to fill do the Task 1. This time we do not split the training set but use all the data to train the model. It is because that previously we are trying to find the best model so we need to split the test set to test the mode. Finally, we can get a prediction of the "New_teleplay.csv". A screen shot is shown in Figure 13.

teleplay_id	name	genre	type	episodes	rating	members
9345	Gakkou no Kowai Uwasa Shin:	Comedy, Horror, School,	medium	33	5.5778513	341
1793	Kaze no Na wa Amnesia	Action, Dementia, Drama, Sci-Fi	long	1	5.6808367	8230
33909	Marine Dreamin'	Music	Music	1	4.332425	126
10585	Chuunen Punk	Music	Music	1	4.3324685	308
28993	Hand Soap	Dementia, Psychological	long	1	3.9756863	246
9074	Arakawa Under the Bridge x Bridge	Comedy, Romance, Seinen	medium	13	5.45722	80394
7290	Cyborg 009: Kaijuu Sensou	Action, Adventure, Sci-Fi	long	1	5.5730863	806
10824	Ranma ½: DoCo Music Video	Comedy, Martial Arts,	short	1	5.1347003	1775
4054	Norimono Oukoku BuBu ChaCha	Comedy, Slice of Life	medium	26	5.044998	315

Figure 16: Prediction

7 Models for Task 2

7.1 Contend-based recommendation system

We firstly try to build a contend-based (CB) recommendation system to give users recommendations. A content based recommender works with data that the user provides, either explicitly (rating) or implicitly (clicking on a link). Based on that data, a user profile is generated, which is then used to make suggestions to the user. In other words, it will give recommendations to a user based on items with "similar" content in user's profile and the recommendation is only dependent on particular user's historical data.

According to the theory of CB system, we will implement the system in the following two steps:

- Calculate the similarity between teleplays. In model-building stage, the system first find the similarity between all pairs of items. Then, it uses the most similar items to a user's already-rated items to generate a list of recommendations in recommendation stage. In this case, we use the cosine similarity to calculate the similarities.

```
m2m = cosine_similarity(df_movies_tf_idf_described)
```

- From existing dataset, find user 53698's favorite teleplay, and based on that, recommend similar teleplays for user 53698. In Figure 17, the screenshot lists several teleplays which are very similar to the favorate movies of user 53698.

```
(31251, 'Mobile Suit Gundam: Iron-Blooded Orphans')
(32281, 'Kimi no Na wa.')
(15391, 'Kagaku na Yatsura')
(4132, 'Wakakusa no Yon Shimai')
(1579, 'Kiniro no Corda: Primo Passo')
(21549, 'Hitotsuboshi-ke no Ultra Baasan')
(28227, 'White Album 2 Picture Drama')
(16458, 'Perrine Monogatari Movie')
(32845, 'Ishitsubutsu Toriatsukaijo')
(81, 'Mobile Suit Gundam: The 08th MS Team')
(83, 'Mobile Suit Gundam: The 08th MS Team - Miller&#039;s Report')
(6235, 'Immoral')
(17501, 'Abe George Kattobi Seishun Ki: Shibuya Honky Tonk')
(1633, 'Shintaisou: Kari')
(20079, 'Ijiwaru Baasan')
(20081, 'Ijiwaru Baasan (1996)')
(4722, 'Skip Beat!')
(29301, 'Kurage no Shokudou')
(31362, 'Osiris no Tenbin')
(9351, 'Geunyeoneun Yeppeotda')
(145, 'Kareshi Kanojo no Jijou')
(148, 'Kita e.: Diamond Dust Drops')
(20123, 'Kappamaki')
(3231, 'Gunslinger Girl: Il Teatrino')
(1701, 'Boku no Marie')
(29357, 'Eien')
(30385, 'Valkyrie Drive: Mermaid')
(32948, 'Fune wo Amu')
(26303, 'Cello Hiki no Gauche (OVA)')
(2760, 'Densetsu Kyojin Ideon: Sesshoku-hen')
(2761, 'Densetsu Kyojin Ideon: Hatsudou-hen')
(201, 'Video Girl Ai')
(31953, 'New Game!')
(28883, 'Hidan no Aria AA')
(30419, 'Wake Up, Girls! Beyond the Bottom')
(3802, 'Gakuen Nanafushigi')
```

Figure 17: CB recommend

7.2 Collaborative filtering-based recommendation system

We also implement the collaborative filtering-based (CF) recommendation system. In CF system, recommendations are based on the assumption that users having similar history are more likely to have similar tastes/needs. In other words, it will give recommendations to a user based on the preferences of "similar" users and recommendation is dependent on other users' historical data. Meanwhile, in CF, we only have user-item interactions (i.e., ratings) as the inputs.

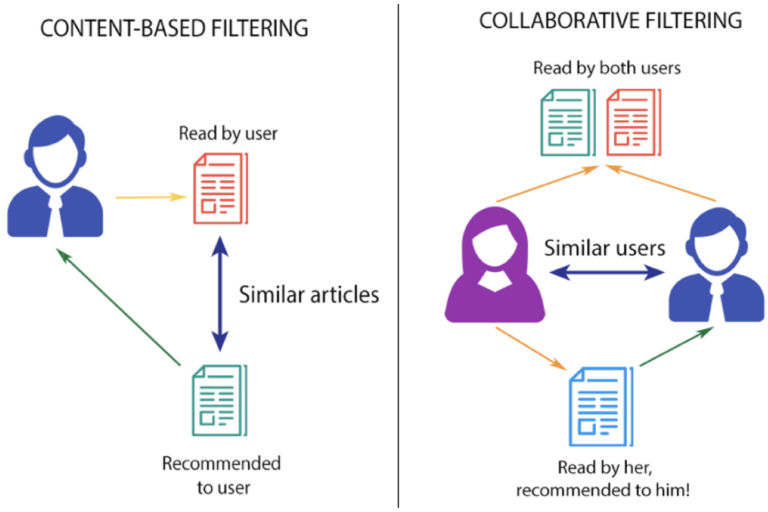

Figure 18: Contend-based and collaborative filtering-based recommendation system

According to the theory of CF system, the workflow can be summarized as the following three steps:

- A user expresses his or her preferences by rating items i.e., teleplays ratings, of the system. These ratings can be viewed as an approximate representation of the user's interest in the corresponding domain.
- The system matches this user's ratings against other users' and finds the people with most "similar" tastes.
- With similar users, the system recommends items that the similar users have rated highly but not yet being rated by this user (presumably the absence of rating is often considered as the unfamiliarity of an item)

The core of the CF algorithm is the matrix calculation. The function fit() is mainly responsible for the matrix calculation. During the iterations, P and Q will be updated and finally they can be used to predict the users' preferences.

```
def fit(self, X_train, X_val):
    m, n = X_train.shape
```

```
self.P = 3 * np.random.rand(self.n_latent_features, m)
self.Q = 3 * np.random.rand(self.n_latent_features, n)

self.train_error = []
self.val_error = []

users, items = X_train.nonzero()

for epoch in tqdm(range(self.n_epochs)):
    for u, i in zip(users, items):
        error = X_train[u, i] - self.predictions(self.P[:, u], self.Q[:, i])
        self.P[:, u] += self.learning_rate * (error * self.Q[:, i] -
        self.lmbda * self.P[:, u])
        self.Q[:, i] += self.learning_rate * (error * self.P[:, u] -
        self.lmbda * self.Q[:, i])

    train_rmse = rmse(self.predictions(self.P, self.Q), X_train)
    val_rmse = rmse(self.predictions(self.P, self.Q), X_val)
    print(train_rmse)
    print(val_rmse)
    self.train_error.append(train_rmse)
    self.val_error.append(val_rmse)
```

The system predicts the rating based on the user's rated teleplays and other similar user's ratings. Here is a screenshot of the prediction of user 53698. The detailed information is in "user_53698_prediction.csv". The system takes more than 10 hours to finish the training and the change of the loss value is shown in Figure 19.

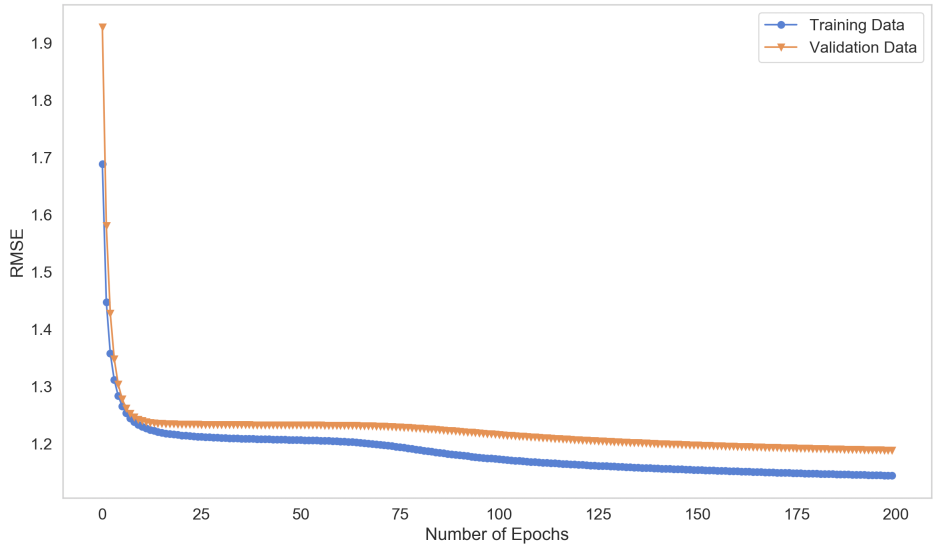

Figure 19: The loss function graph of each iteration

7.3 Hybrid recommendation system using neural network

After introducing the contend-based and collaborative filtering-based recommendation system, let's see the hybrid recommendation system using the neural network.

The incentive of using hybrid recommendation system is to address these shortcomings of the matrix factorization methods. The shortcomings of the traditional recommendation system are as follows:

- The matrix factorization also had the cold start problem due to the fact that it had no feature vector or embedding for the new items.
- Matrix factorization often tends to recommend popular items to everyone which does not always reflect the specific user interests mostly when Dot products are used.
- Matrix factorization works on the simple inner product of the User and item feature embeddings, it is often not enough to capture and represent the complex relations in the user and items.

Deep neural network (DNN) models can address these limitations of matrix factorization [7]. DNNs can easily incorporate query features and item features (due to the flexibility of the input layer of the network), which can help capture the specific interests of a user and improve the relevance of recommendations.

Meanwhile, DNN can model the non-linear interactions in the data with non-linear activations such as ReLU, Sigmoid, Tanh... This property makes it possible to capture the complex and intricate user-item interaction patterns.

The structure of the network is shown in Figure 20.

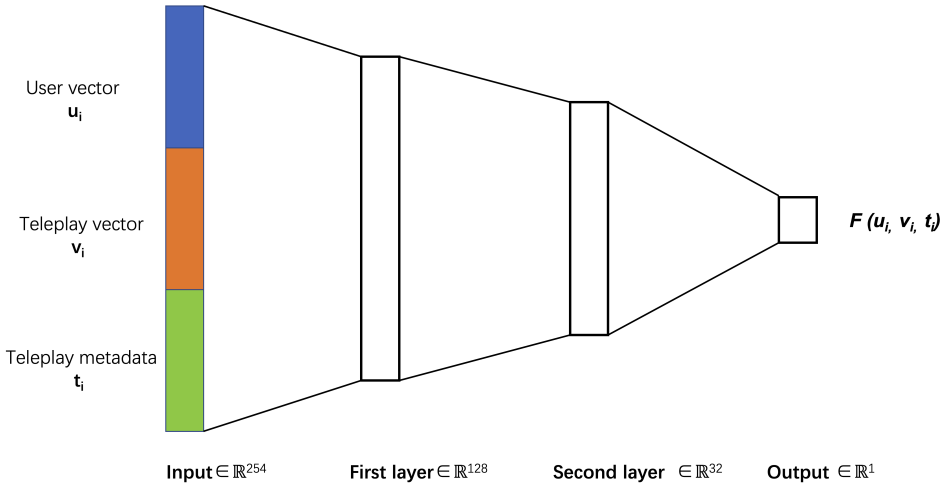

Figure 20: The structure of the DNN

We firstly embed the 73517 users into a vector of length 100, which is the User vector in the figure. We also embed the 10204 teleplays into a vector of length 100, which is the Teleplay vector in the figure. We also included some metadata of the teleplays, such as

episodes, members, genres and types. The dimension of the input layer is thus 100 + 100 + 54 = 254. The following two layers are all fully connected layers. Now with this model, we can predict how a specific user will rate a specific teleplay with some level of confidence. The codes of the model are shown here.

```
class RecommendationNet(nn.Module):
    def __init__(self):
        super(RecommendationNet, self).__init__()
        self.users = nn.Embedding(73517, 100)
        self.animes = nn.Embedding(10204, 100)
        self.linear1 = nn.Linear(100 + 100 + 54, 128)
        self.linear2 = nn.Linear(128, 32)
        self.linear3 = nn.Linear(32, 1)

    def forward(self, x):
        user = x[:, 0].long()
        anime = x[:, 1].long()
        otherfeatures = x[:, 2:]
        userVector = self.users(user)
        animeVector = self.animes(anime)
        layer1 = torch.cat((userVector, animeVector, otherfeatures), 1)
        layer2 = F.relu(self.linear1(layer1))
        layer3 = F.relu(self.linear2(layer2))
        out = torch.sigmoid(self.linear3(layer3))
        return out
```

The experiments show that the RMSE of the DNN model can reach 1.103. This can be explained that the DNN will combine the advantages of the CB and CF recommendation system. DNN will not only consider the user's tastes and preferences, but also try to find the other similar users. Another important feature is that it also contains the other metadata such as episodes, members and so on. Therefore, theoretically, it will have better performance.

8 Conclusion and future work

In this project, we finished two tasks. In task 1, we tested ten traditional regression models and DNN to find the best model to predict the ratings. In task 2, we implemented three kinds of recommendation systems to give personalized recommendation services. In the future, we can try more different DNN models with different hyper parameters in both task 1 and task 2. Also, we may use some other more scientific criteria to evaluate the three models in task 2 rather than only use RMSE.

References

[1] Pandas. [Online]. Available: https://pandas.pydata.org/

[2] Seaborn. [Online]. Available: https://seaborn.pydata.org/

[3] Matplotlib. [Online]. Available: https://matplotlib.org/

[4] Wordcloud. [Online]. Available: https://amueller.github.io/word_cloud/

[5] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," 2019.

[6] Regressions. [Online]. Available: https://github.com/angeliababy/houseprice_regression

[7] S. Zhang, L. Yao, A. Sun, and Y. Tay, "Deep learning based recommender system: A survey and new perspectives," *ACM Computing Surveys (CSUR)*, vol. 52, no. 1, pp. 1–38, 2019.