Google Local Reviews - Rating Prediction

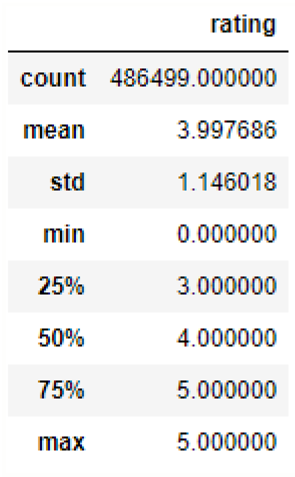
Thy Nguyen
UC San Diego
tcn002@ucsd.edu

1. Identify a Dataset to Study and Perform an Exploratory Analysis

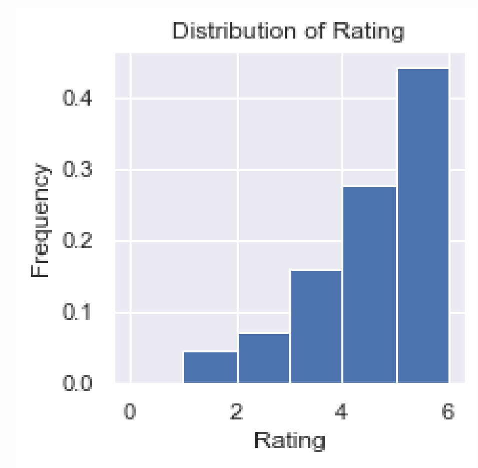
The dataset for this study is Google Local Reviews, which includes three separate sets of data, namely Places, User, and Review. In particular, there are 3,116,785 business and service locations in the Places dataset, 4,567,431 worldwide Google Plus users in the User dataset, and 11,453,845 reviews/ratings left by those users in the Review dataset. The first two datasets (Places and User) can be processed quickly by the Jupyter Notebook, but the huge Review file takes much more time to be loaded (around 15 minutes to extract only the first 2.15 million entries, which are much fewer than the provided dataset, but such amount of entries has satisfied the requirement of obtaining at least 50,000 rows of data). For this particular study, we'll use a subset of this dataset, which contains 486,499 entries of Google Plus users from all over the world.

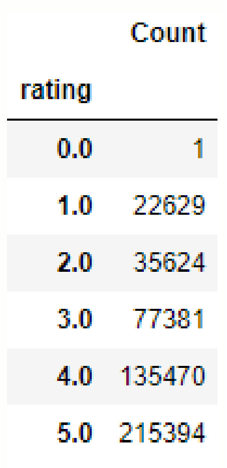
After loading those aforementioned datasets, we merge them by their distinct gPlusUserId and gPlusPlaceId, then consider this new dataset as "combined". Besides, as the project aims to perform a predictive task (specifically, predicting a business's rating), we split the "combined" dataset into three smaller subsets: a test set with the most recent interaction on Google Plus of each user, a validation set with the second latest interaction of each user, and a training set containing the rest of the interactions. Per this way of splitting data, we have to discard all users that have fewer than three actions on Google Plus. As a result, we are left with 60,530 unique users, 400,050 unique places, and 486,499 actions. Hence, there are roughly 8 actions per user and 1.22 actions per item.

Each training, validation, and test set contains 20 columns. Most of them are encoded as type "object", but some columns such as "rating" and "unixReviewTime" are of type "float64", and the "closed" column (whether the business is closing or not) is of type "bool" (e.g. True or False booleans). Additionally, some columns are mainly comprised of null data, namely the "price" column (with 74.26% missing data) or the column "previousPlaces" visited by users (with 73.40% missing data). Fortunately, our predictive task uses a model that does not need those columns. Instead, we will focus closely on the columns named "rating", "gPlusUserId, "gPlusPlaceId", and "unixReviewTime".



	rating
count	486499.000000
mean	3.997686
std	1.146018
min	0.000000
25%	3.000000
50%	4.000000
75%	5.000000
max	5.000000

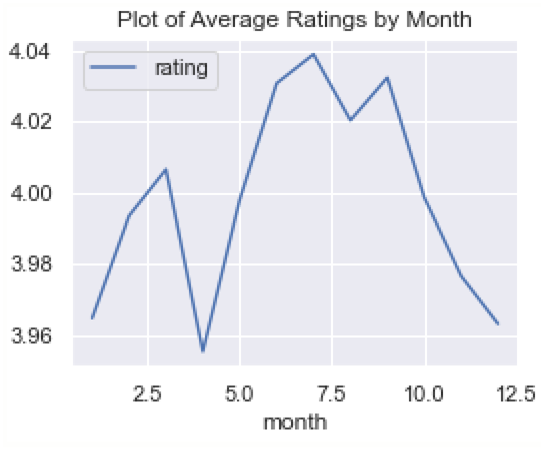


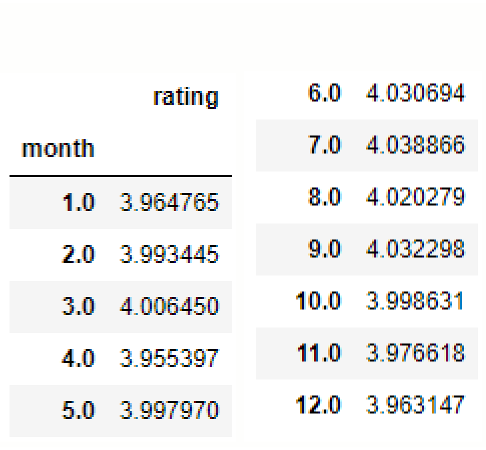


	Count
rating	
0.0	1
1.0	22629
2.0	35624
3.0	77381
4.0	135470
5.0	215394

Picture 1: Statistics of Rating

Specifically, Google Local uses star ratings to evaluate a business. The ratings are from 0.0 to 5.0, with an average rating of 3.998 and a standard deviation of 1.15. There is no null data in the “rating” column, and the distribution of ratings is left-skewed, with quite a long tail at the beginning of the histogram. Most ratings are five stars, which account for 44.27% of the data. Other ratings like 4.0 and 3.0 stars cover 27.85% and 15.91% of the data respectively. Interestingly, there is only one person rating one star in our subset. However, this rating appears to contradict the review text of that user. He or she expressed a strong interest in the restaurant’s food by saying the “Gamja fries are amazing.”, “Okonomiyaki is one of the best in the city“, and “[r]eally great bibimbap”, but the rating was 0-star. This signals us that the dataset contains noise in it, and this could be the case where the user forgot to star-rate their food, so the default settings for missing data set the rating to 0. Confronting this issue, we choose to exclude the 0.0 rating because this user mainly gave a rating from 3.0 to 5.0, and we want to prevent the noise (of the zero-rating) from affecting our model.

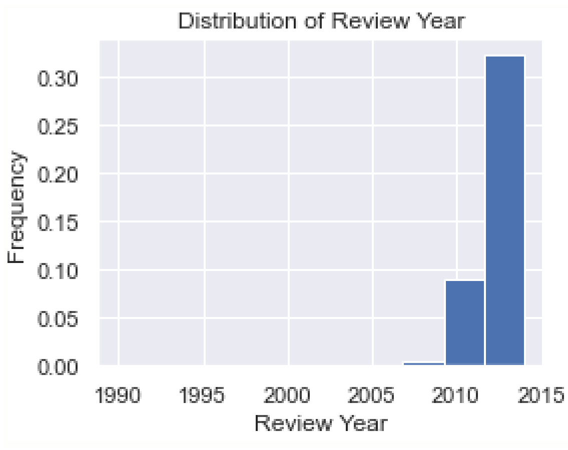


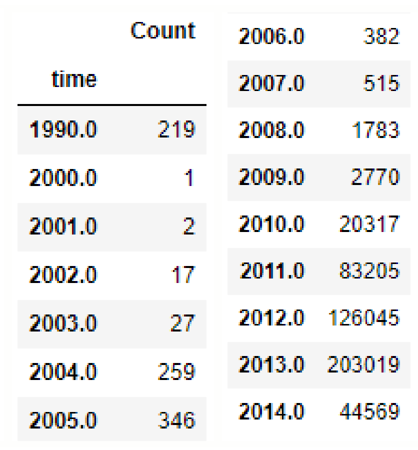


	rating
month	
1.0	3.964765
2.0	3.993445
3.0	4.006450
4.0	3.955397
5.0	3.997970
6.0	4.030694
7.0	4.038866
8.0	4.020279
9.0	4.032298
10.0	3.998631
11.0	3.976618
12.0	3.963147

Picture 2: Statistics of Ratings by Month

As shown in the above graph and tables, the monthly average ratings range from 3.95 and 4.04. July has the highest average rating of around 4.04, and the top three lowest average ratings belong to April (3.955), December (3.963), and January (3.964). It appears that people tend to give higher ratings during the Summer and the beginning of Autumn, with the average ratings being above 4.0. Therefore, we should consider the temporal feature for our predictive model in the later section.

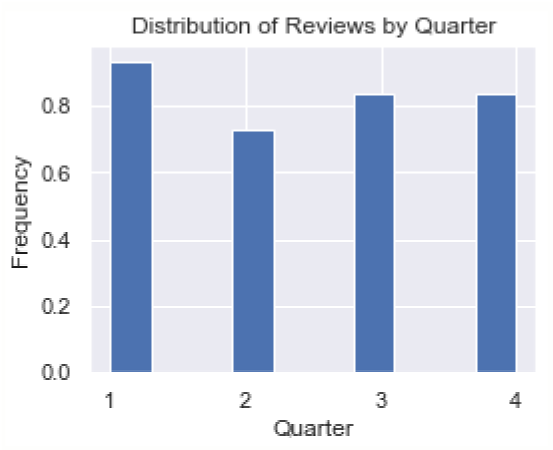


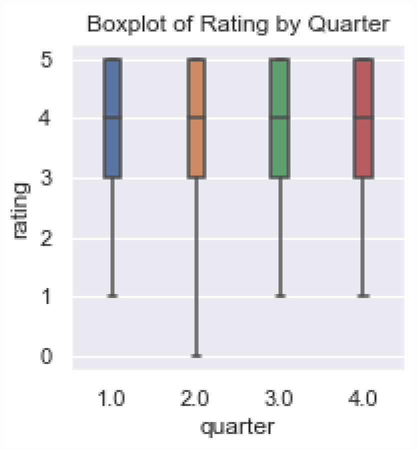


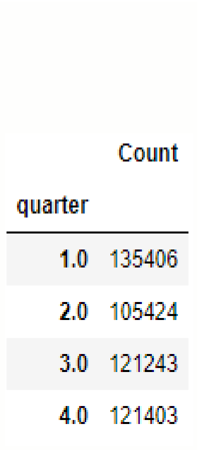
time	Count
1990.0	219
2000.0	1
2001.0	2
2002.0	17
2003.0	27
2004.0	259
2005.0	346
2006.0	382
2007.0	515
2008.0	1783
2009.0	2770
2010.0	20317
2011.0	83205
2012.0	126045
2013.0	203019
2014.0	44569

Picture 3: Statistics of Review Year

The review years in our sub-dataset are from 1990 to 2014. The distribution is also left-skewed as most interactions were captured in 2013. There are significant differences in the number of user interactions among the ’90s, the early 2000s and the 2010s years. These differences could be explained by the fact that more people have gained access to the Internet recently, and hence, the number of Google Plus reviews increased drastically.







quarter	Count
1.0	135406
2.0	105424
3.0	121243
4.0	121403

Picture 4: Statistics of Reviews by Quarter

Even though the reviews are given throughout a year, people seem to usually leave a review in the first quarter; there is not much of a difference between the number of reviews submitted in the third and fourth quarters. Meanwhile, the second quarter has the fewest number of reviews. Additionally, the above boxplot shows the similarities of the rating given among the quarters. Specifically, the 25^{th}, 50^{th}, and 75^{th} percentiles of star ratings per quarter are always equal to 3.0, 4.0, and 5.0 respectively. Except for the second quarter with a zero rating, other quarters have a minimum rating of 1.0. As a result, we expect the one-hot encoding of the quarters won't contribute much to the accuracy of our rating prediction.

Finally, we introduce the attributes gPlusPlaceId(s) and gPlusUserId(s), which are all the strings of length 21, starting with digit "1" followed by other digits from 0 to 9. With these distinct IDs, we can easily differentiate users and places that have the same names.

2. Identify a Predictive Task

As mentioned earlier, a suitable predictive task for this dataset is to predict the rating for a business/service based on a user's review time and their historical ratings. A model for this task will be built based on the *factorization machine* framework. This model is appropriate as it captures the interactions among users, review times, and the locations that the users have spent a visit. More details about this model will be included in the third section of the report. The model will use some input features from the dataset, namely "gPlusUserId", "gPlusPlaceId", and "unixReviewTime", whereas the output will be the "rating" column.

Before applying this model, we conduct some data pre-processing. Apart from splitting data in terms of the latest interactions, we have to process the missing values of the *unixReviewTime* column. Firstly, we drop all rows with missing *unixReviewTime* in the training data and keep a list of the remaining unique users. Secondly, in the validation and test sets, we delete the users that are no longer present in the training set. This step is done so as to guarantee that all users appearing in the validation/test sets have their older interactions properly trained. (In other words, we choose to skip the unseen portion of data). Thirdly, we impute the missing *unixReviewTime* in the test/validation sets by the time of the last interaction per user in the training data. This imputation step is done based on our common sense: the most recent reviews shouldn't be sooner than the latest review time available in the training data. For example, user A has three interactions in the training set, with the review times being 2021-08-20, 2021-08-25, and 2021-08-30 respectively. Then her latest (training) review time is 2021-08-30, and this will also be the imputed value for her missing (testing) review time. Additionally, after we handle missing data, we hope to separate the temporal features. In particular, we extract each component of a *datetime* object. For instance, given a *unixReviewTime*, we convert it into a normal *datetime* object, then get the corresponding years, months, and quarters from those times. This extraction step also concludes our data-processing duty.

Next, we'll establish the assessment criteria for our model. Since the output of our predictive task is continuous, we can use MSE to compute the mean squared error between the prediction and the ground-truth value of a rating. This is similar to finding an average squared distance between the predictions and the true ratings. Therefore, the smaller value of MSE we can obtain from the test set, the higher accuracy our model achieves. Indeed, to penalize large errors, the MSE assigns very large penalties. The reason is that MSE seems to assume large errors are uncommon when compared to the common small errors (McAuley, 2021), and the formula of MSE is

$$\mathrm{MSE}(y, f_\theta(X)) = \frac{1}{|y|}\sum_{i=1}^{|y|}(f_\theta(x_i) - y_i)^2 \tag{1}$$

where $f_\theta(X)$ is a prediction model, y is a set of ratings, y_i is a single rating in the set y, and x_i is a single input of the model $f_\theta(X)$. Along with MSE, we can calculate the coefficient of determination R^2, which is

$$R^2 = 1 - \frac{\mathrm{MSE}(y, f_\theta(X))}{\mathrm{var}(y)} \tag{2}$$

where $var(y)$ is the variance of ratings in our dataset. Even though MSE is a pretty good evaluation metric for our predictive task on a 5-point star-rating scale, MSE generally relies on the scale and variability of a dataset. On the other hand, R^2 coefficient shows the correlation between predictions and the true ratings. Hence, using R^2 in addition to MSE will strengthen our conclusion about the validity of our model's prediction. In particular, a perfect predictor has an R^2equal to 1, whereas a zero R^2 illustrates a trivial predictor.

In our predictive task, we plan to use the factorization machine framework to build our model. Details of this model will be covered in section 3 of the report. Based on such a model, we hope to introduce some relevant baselines that can be used for our comparison, namely
[1] predicting a new rating based on the median rating given by a user,
[2] predicting a new rating to be equivalent to the average rating given by a user,
[3] predicting a new rating by *fastFM* without temporal features (*)

The first two baselines are merely trivial predictors that illustrate whether our model is working or not. The subsequent baseline describes the "ablation" method by removing the temporal factors out of our model. This baseline will help show what components of our method are actually useful and working efficiently.

3. Describe the Model

To emphasize what we've mentioned earlier, we base our model on the implementation of the factorization machine framework. According to researcher Rendle (2010), *Factorization Machine* is a general-purpose approach that can combine several features into a model and capture their pairwise interactions. For instance, Factorization Machines can be used for various predictive tasks, namely Regression, Binary classification, and Ranking. Factorization Machine expands the ideas of the latent-factor models — models that primarily account for the pairwise interactions between users and items. Following this argument, we'll display some examples to compare the matrix representations of the Factorization Machine method and a usual Matrix Factorization.

User / Place	$place_1$ $(item_1)$	$place_2$ $(item_2)$	$place_3$ $(item_3)$

(*) The feature matrix of this method will be provided in section 3.

$user_1$	Rating = 4.0	Rating = 2.5	
$user_2$		Rating = 3.0	
$user_3$	Rating = 5.0	Rating = 3.5	Rating = 5.0

Picture 5: An Example of Matix Factorization (on Training Data)

	$user_1$	$user_2$	$user_3$	$place_1$	$place_2$	$place_3$	$other_1$	$other_2$	y
x_1	1	0	0	1	0	0	1.2	3.5	4.0
x_2	1	0	0	0	1	0	2.3	1.2	2.5
x_3	0	1	0	0	1	0	3.4	6.3	3.0
x_4	0	0	1	1	0	0	4.5	2.6	5.0
x_5	0	0	1	0	1	0	5.6	3.6	3.5
x_6	0	0	1	0	0	1	5.5	6.5	5.0

Picture 6: An Example of Factorization Machine matrix (on Training Data) Corresponding to the above Matrix Factorization

(Note: $other_1$ and $other_2$ are just some auxiliary features. These data are used to demonstrate the matrices, and they are not from Google Local Reviews dataset).

In our case, the factorization machine method suggests turning the input features into a feature matrix X, whose entries are one-hot-encoded identities of users, items (places), and the review times (years and months). For instance,

$$\begin{array}{cccc} \left[\begin{array}{cccc} 100000\ldots & 0100000\ldots & 000100000\ldots & 000100000000 \\ 000100\ldots & 0000100\ldots & 001000000\ldots & 000100000000 \\ \vdots & \vdots & \vdots & \vdots \\ 001000\ldots & 0010000\ldots & 000000100\ldots. & 000010000000 \end{array}\right] \\ \underbrace{\qquad\qquad}_{\text{user}} \quad \underbrace{\qquad\qquad}_{\text{item}} \quad \underbrace{\qquad\qquad}_{\text{year}} \quad \underbrace{\qquad\qquad}_{\text{month}} \end{array}$$

With a feature dimensionality F, the model equation is defined by the summation of *offset* terms, *bias* terms, and the *interactions* of all pairs of non-zero features.

$$f(x) = \underbrace{w_0 + \sum_{i=1}^{F} w_i x_i}_{\text{offset and bias terms}} + \underbrace{\sum_{i=1}^{F}\sum_{j=i+1}^{F} \langle \gamma_i, \gamma_j \rangle x_i x_j}_{\text{feature interactions}} \tag{3}$$

where $w_0 \in \mathbb{R}$, $\mathbf{w} \in \mathbb{R}^F$, $\gamma \in \mathbb{R}^{FxK}$, $< \gamma_i, \gamma_j >= \sum_{f=1}^{K} \gamma_{i,f} \cdot \gamma_{j,f}$ for some positive integer K.

More specifically, the model can be computed in some $O(KF)$ time if we re-write the feature interactions in equation (3) as the following equation:

$$\sum_{i=1}^{F}\sum_{j=i+1}^{F} \langle \gamma_i, \gamma_j \rangle x_i x_j = \frac{1}{2}\sum_{f=1}^{K}\left(\left(\sum_{i=1}^{F} \gamma_{i,f} x_i\right)^2 - \sum_{i=1}^{F} \gamma_{i,f}^2 x_i^2\right) \tag{4}$$

where K is the dimension of the latent factors.

As we propose this model, we want to explain why we choose it. Firstly, to predict ratings, we need to understand a user's previous rating behavior, which means we want to incorporate the temporal features into our model. A usual non-temporal latent-factor model can only capture the pairwise interactions of users and items. For instance,

$$r(u,i) = \alpha + \beta_u + \beta_i + \gamma_u \cdot \gamma_i \tag{5}$$

where γ_u, γ_i are latent vectors describing a user u or an item i;

β_u and β_i denote the extent to which user u's rating trend or item i's rating tends to be higher or lower than an offset α. Hence, using Factorization Machine makes it feasible to combine auxiliary features such as review times. Secondly, temporal dynamics might play a crucial role in determining a user's rating tendency. For instance, ratings could vary on some months of the year or some days of the week. Specifically, from our EDA steps, we observe a higher average rating during four months (from June to September), and we observe the lowest average rating in April. Additionally, temporal dynamics have proved their advantages on multiple occasions, namely, the Netflix Prize introduced in Chapter 5 lecture (McAuley, 2021). Thirdly, Factorization Machine works well in problems with huge sparsity. In particular, when we display Google Local's user-to-item interactions as a matrix, there exist many empty entries in this matrix factorization. The reason is that even though there are 400,050 unique places in our sub-dataset, each place only captures 1.22 actions, and each user interacts with 8 places on average. Hence, the user-item interactions are sparsely spread throughout the matrix, and Factorization Machine will assist us in solving the issues related to sparsity.

Next, we would like to discuss how we optimize our model. First, we start with a non-temporal latent-factor model built by the *Surprise* package. This model is used to set up our learning pipeline and help find some good initial values of latent factors and regularizers. After that, we try to incorporate the available temporal features in the dataset. We choose to one-hot encode the review times, especially the years and months of each interaction. Then, we'll predict the ratings on the validation sets and find the corresponding MSE and R^2 coefficient. This step helps us pick the optimal hyperparameters that yield some reasonably good results of MSE and R^2 without overfitting on the validation set. In practice, the model parameters of the Factorization Machine can be learned by Gradient Descent (GD) or Stochastic Gradient Descent (SGD), and according to researcher Rendle (2010), the gradient of this model is:

$$\frac{\partial f(x)}{\partial \theta} = \begin{cases} 1, & \theta = w_0 \\ x_i, & \theta = w_i \\ x_i \sum_{j=1}^{F} \gamma_{j,f} x_j - \gamma_{i,f} x_i^2, & \theta = \gamma_{i,f} \end{cases} \tag{6}$$

Lastly, when we feel satisfied with our hyperparameters as well as MSE and the R^2 on our validation set, we move to the next stage where we'll predict ratings on the test set and report the corresponding evaluation metrics.

During our modeling process, we confront the issues of missing *unixReviewTime* data. However, we have mentioned how to address this in the second part of this report. In short, we choose to replace the missing time with the latest interaction's time of a user.

Now, let us introduce some models that we consider for comparison.

[1] $FM_{no\ time}$: Factorization Machine model without temporal features (i.e.we consider only the users and items in a feature matrix representation).

$$X = \begin{bmatrix} 100000... & 0100000 \\ 000100... & 0000100 \\ \vdots & \vdots \\ 001000... & 0010000 \end{bmatrix}$$
$$\qquad\quad \underbrace{\qquad\quad}_{\text{user}} \quad \underbrace{\qquad\quad}_{\text{item}}$$

[2] $FM_{year\ only}$: Factorization Machine model with one of the temporal features left out (for instance, we'll leave out the month's one-hot encoding).

$$X = \begin{bmatrix} 100000... & 0100000... & 000100000 \\ 000100... & 0000100... & 001000000 \\ \vdots & \vdots & \vdots \\ 001000... & 0010000... & 000000100 \end{bmatrix}$$
$$\qquad\quad \underbrace{\qquad\quad}_{\text{user}} \quad \underbrace{\qquad\quad}_{\text{item}} \quad \underbrace{\qquad\quad}_{\text{year}}$$

[3] $FM_{with\ quarter}$: Factorization Machine model with one-hot encodings of “year” and “quarter”

$$X = \begin{bmatrix} 100000\ldots & 0100000\ldots & 000100000 & \ldots & 1000 \\ 000100\ldots & 0000100\ldots & 001000000 & \ldots & 0100 \\ \vdots & \vdots & \vdots & & \vdots \\ 001000\ldots & 0010000\ldots & 000000100 & \cdots & 0010 \end{bmatrix}$$
$$\underbrace{\qquad\qquad}_{\text{user}} \quad \underbrace{\qquad\qquad}_{\text{item}} \quad \underbrace{\qquad\qquad}_{\text{year}} \quad \underbrace{\qquad\qquad}_{\text{quarter}}$$

[4] SVD: Matrix Factorization-based algorithm without temporal features (built by package *Surprise’s SVD*). SVD stands for the *Singular Value Decomposition*, a method of decomposing the matrix M into

$$M = U\Sigma V^T \tag{7}$$

where U is the eigenvector of MM^T (the left singular value of M), V is the right singular value of M (eigenvector of M^TM), and Σ is a diagonal matrix of eigenvectors of MM^T (or M^TM).
Then, the prediction is defined by the following equation:

$$\hat{r}_{ui} = \mu + b_u + b_i + q_i^T p_u \tag{8}$$

where b_u and b_i are the bias terms for users and items; p_u and q_i are the user and item factors, respectively (Surprise documentation, 2021).

When attempting different models, the biggest challenge we face is to choose suitable hyperparameters. We have to adjust the values of latent factors, regularizers as well as the number of epochs in *Stochastic Gradient Descent* to make the model effectively learn the given inputs. This step takes a lot of time, approximately 12 minutes every time we try to tune hyperparameters for a model that needs 1,000 iterations. However, the validation MSE sometimes gets worse, and we have to reset those hyperparameters. Another unsuccessful attempt is when we set too much computation in a model’s training process, making the Google Colab run out of RAM. This mistake teaches us to be more mindful and careful when we use a public resource to train our models.

Finally, we will discuss some strengths and weaknesses of the models we use for comparison. The first model is an implementation of the Factorization Machine method with temporal features; let’s call it $FM_{temporal}$. This model captures the available temporal dynamics of the dataset, creating a huge matrix of feature representation. With Factorization Machine, the problems of sparse data or huge data can quickly be solved. Additionally, since $FM_{temporal}$ can work with any real values, we can either use the real values of a feature (e.g. price of a product) or one-hot encode the features to apply them to our model. Aside from its flexibility, $FM_{temporal}$ runs in some linear time, and it’s optimized in the primal (Rendle, 2010). The weaknesses of

$FM_{temporal}$ are that it takes a long time to train data on a high-rank matrix representation; it requires us to establish the one-hot encoding of the temporal components when we try to build a feature matrix X, and finding the optimal hyperparameters for $FM_{temporal}$ can be troublesome. Secondly, we'll talk about SVD method. This method runs quickly and gives a pretty good result on the validation set. The only downside is that it does not utilize the available auxiliary features like "month" and "year". Other models, namely $FM_{no\ time}$, $FM_{year\ only}$, and $FM_{with\ quarter}$ are just variants of $FM_{temporal}$, so their advantages and disadvantages mostly align with $FM_{temporal}$.

4. Describe Related Literature

We base our study on two research papers of Professor McAuley and his graduate students. In our project, we use three existing datasets that originally come from Google Local Reviews, a website where business owners can introduce their service, and customers can submit their reviews or ratings. The datasets were collected and processed by the aforementioned Professor and students, who conducted two research papers "Translation-based Recommendation: A Scalable Method for Modeling Sequential Behavior" in 2017 and "Translation-based factorization machines for sequential recommendation" in 2018.

In the former paper, researchers focus on exploring the complexity of user-to-item and item-to-item interactions, which are responsible for designing a great recommender system that predicts users' sequential behavior (i.e. predicts what users will consume next). They also successfully model the interactions among users, users' previously-viewed items, and the next items that users will purchase, and as mentioned in their paper, this model outperforms the contemporary ones. Specifically, in lieu of using the existing methods of decomposing high-order interactions, they came up with a new "unified" method called TransRec, where users are transformed into translation vectors while items are instilled in a "transition space".

On the other hand, the latter paper tries to predict an individual's purchasing activities based on his/her historical interactions. This paper suggests a method called TransFM, which stands for Translation and Factorization Machines, along with some metric-based approaches. In response to TransRec's method, the TransFM's writers also agree on the strength of using third-order interactions (between users, previously viewed items, and next items) on creating an excellent recommender system. TransFM, on the other hand, continues the success of TransRec and works well in multiple aspects. For example, it takes care of the strength of interactions among different features by using Euclidean distance, incorporates all observed features for a better prediction result, and finally, it runs in a linear time and outperforms the contemporary methods.

For simplicity, from this point of our project, the 2017 and 2018 research papers will be referred to as "TransRec" and "TransFM" respectively.

Compared to our predictive tasks, TransRec and TransFM methods are used for different purposes, specifically, predicting sequential purchasing behaviors of users. However, we still learn a lot from those two papers. For instance, we've learned how to split the data reasonably when it comes to getting the validation and test sets. We learn to adapt the Factorization Machine method to build our model and learn to also consider temporal features when building matrix representations.

Now, we will discover what similar datasets have been studied in the past. For the TransRec paper, aside from the Google Local dataset, TransRec utilizes datasets from Amazon (reviews and the corresponding timestamps), Epinions (aggregated customer reviews on the Internet), Foursquare (check-in locations of the users), and Flixster (online movie ratings). In general, each of these datasets contains users and items that have at least 5 associated actions, and the “rating” column will be counted as users’ positive feedback (if such a column exists in a dataset). Using multiple datasets in the TransRec paper, researchers are able to compare the strengths and weaknesses of each prediction methodology, compute the scores of two evaluation metrics (AUC and Hit@50) as well as figure out how well the new method improves from the current state-of-the-art methods. Similar to TransRec, the TransFM paper uses Amazon and Google Local datasets to evaluate its method, combining with the data from MovieLens (data about movie ratings) to evaluate the performance of a wide range of recommendation algorithms. Additionally, TransFM researchers specify they use AUC as an evaluation metric, where the higher AUC value signifies a better prediction result. The formulas of AUC and Hit@50 are shown below:

$$AUC = \frac{1}{|\mathcal{U}|} \sum_{u \in \mathcal{U}} \frac{1}{|\mathcal{I} \setminus \mathcal{S}^u|} \sum_{j' \in \mathcal{I} \setminus \mathcal{S}^u} \mathbf{1}(R_{u,g_u} < R_{u,j'}) \tag{9}$$

$$Hit@50 = \frac{1}{|\mathcal{U}|} \sum_{u \in \mathcal{U}} \mathbf{1}(R_{u,g_u} \leq 50), \tag{10}$$

where $|\mathcal{U}|$ is the number of users; $|\mathcal{I} \setminus \mathcal{S}^u|$ is the number of items that have no interaction with users u; $R_{u,k}$ is the rank of item k given by user u; **1** is an indicator function that returns 1 or 0, and g_u is the “ground-truth” item that user u interacted with at the latest time (He et al., 2017).

Again, we would like to emphasize that TransRec and TransFM are both excellent methods, but they pertain to predicting the next item purchased by a user rather than predicting a rating of an item. Hence, when we talk about the state-of-the-art methods that are currently employed to study the rating data of Google Local Reviews, we should also present a general-purpose approach called *recurrent neural networks* (RNN). This method can capture complex dynamics around a user’s interaction sequences, learn complex semantics, and outperform other traditional sequential models (McAuley, 2021).

Finally, we will visit the main conclusions of TransRec and TransFM papers as well as the methods from recurrent networks. In short, TransRec researchers conclude their paper by emphasizing the advantages of using this new method in solving the sequential prediction task as well as other general-purpose recommendation issues. Particularly, the TransRec method is excellent at modeling the third-order interactions among features given in the datasets. Meanwhile, TransFM researchers clarify the importance of incorporating the content features in enhancing the general-purpose models and describe the future directions of their research. On the other hand, with the help of RNN, problems related to time-series data can be easily forecasted. Even though we agree with those researchers on the importance of adding additional features to our model, our results (specified in section 5) do not align with these perspectives. Indeed, when adding temporal features into the model, validation MSE gets bigger, while the validation R^2 coefficient is smaller. The results suggest the time component does not fit with our methodology, and thus, we should exclude them if we want to gain higher validation and testing results.

5. Describe Results and Conclusions

- Note: Except for the parameters that are self-explained, we'll include the interpretation of our model's parameters inside each summary table below.

a). Model SVD: Matrix Factorization-based algorithm without temporal features (by *Surprise*)

Model = SVD()	**MSE_val**	R^2	**Comments**

Default:
n_factors = 100,
n_epochs (for SGD) = 20,
biased = True,
init_mean = 0,
init_std_dev = 0.1,
lr_all (learning rate for all parameters) = 0.005,
reg_all (regularization term for all parameters) = 0.02
	1.3139	0.0859	*Interpretation*:
			n_factors = The number of factors,
			n_epochs = The number of iteration of the SGD procedure,
			biased = True (if we choose to use a baseline),
			init_mean / init_std_dev = The normal distribution's mean / STD for factor vectors initialization.
			
			The chosen model somehow fits the given input, but R^2 is still small and needs to be optimized.

Increase the default n_factors: n_factors = 120	1.3163	0.0843	The chosen model fits worse than the previous model as MSE_val is bigger, and R^2 is smaller.

Decrease the default n_factors:
n_factors = 80
n_factors = 50
n_factors = 10
n_factors = 1
	1.3123
	1.3098
	1.3066
	1.3056
		0.0870
		0.0888
		0.0910
		0.0914
			Decreasing the number of factors betters the MSE_val and R^2**.**

Decrease the default reg_all:
reg_all = 0.01
reg_all = 0.005
reg_all = 0.0025
reg_all = 0.001
	1.3059
	1.3058
	1.3058
	1.3058
		0.0915
		0.0915
		0.0915
		0.0915
			Decreasing the regularization term for all parameters, combined with 1 (latent) factor, improves MSE_val and R^2**.**

Decrease the default lr_all:			Decreasing the learning rate does

lr_all = 0.001	1.3895	0.0333	not help improve the model's performance on the validation set.

Increase the default lr_all:
lr_all = 0.01
lr_all = 0.015
lr_all = 0.0175
lr_all = 0.02
	1.2765
	1.2729
	1.2751
	1.2787
		0.1120
		0.1145
		0.1129
		0.1104
			Increasing the learning rate for all parameters (to 0.015) reduces the MSE_val and betters the R^2 coefficient.

Decrease the default n_epochs
n_epochs = 18
n_epochs = 15
n_epochs = 10
	1.2725
	1.2739
	1.2854
		0.1148
		0.1138
		0.1057
			Decreasing the default number of epochs from 20 to 18 gives the best R^2 up to now

Increase the default n_epochs n_epochs = 25	1.2768	0.1117	Increasing the default n_epochs doesn't yield better values of R^2 and MSE_val for the validation set (R^2**:** 0.1117 < 0.1138); (MSE_val: 1.2768 > 1.2739).

Conclusion 1: The SVD model works quite well. With n_factors = 1, reg_all = 0.0025, lr_all = 0.015, n_epochs = 18, its MSE_test is 1.3313, R^2_{test} = 0.1027.

b). $FM_{no\ time}$: Factorization Machine method without temporal features (by *fastFM* package)

Model = $FM_{no\ time}$ (baseline #3)	**MSE_val**	R^2	**Comments**
n_iter = 100, init_stdev = 0.1, rank (of the factorization, used for 2nd- order interactions) = 8, l2_reg_w (L2 penalty weight for linear coefficients) = 0.1, l2_reg_V (L2 penalty weight for pairwise coefficients) = 0.1	1.3110	0.0879	This starting point gives a better MSE_val and R^2, compared to SVD method.

Increase rank
rank = 15
rank = 30
rank = 45
	1.3095
	1.3087
	1.3085
		0.0890
		0.0895
		0.0897
			Increasing the rank improves the performance of the model.

Increase l2_reg_w
l2_reg_w = 0.125
l2_reg_w = 0.15
	1.3022
	1.3041
		0.0941
		0.0928
			Increasing L2 penalty weight for linear coefficients to some extent will reduce MSE_val and increase R^2**.** Meanwhile, changing l2_reg_V

			(either increasing or decreasing it) won't enhance the model's performance. For example, when l2_reg_V = 0.08, MSE_val = 1.306, R^2 = 0.0913
Increase n_iter n_iter = 1000	1.3011	0.0948	Increasing the number of iterations betters the model's performance on the validation set.

<u>Conclusion 2</u>: Compared to SVD, $FM_{no\ time}$ achieves a better MSE_{val} and R^2 at the beginning, but simply changing the model parameters won't help us obtain one MSE_{val} as low as the SVD model. For $FM_{no\ time}$ where n_iter = 1000, init_stdev = 0.1, rank = 45, random_state = 123, l2_reg_w = 0.125, l2_reg_V = 0.1, the MSE_{test} = 1.3594, and R^2= 0.0838.

c). Model $FM_{year\ only}$: Factorization Machine method with "year" as a temporal feature (by *fastFM* package)

Model = $FM_{year\ only}$	**MSE_val**	R^2	**Comments**

n_iter = 20,
init_stdev = 0.1,
rank = 5,
l2_reg_w = 0.125,
l2_reg_V = 0.1
	1.3265	0.0772	The chosen model somehow fits the given input, but R^2 is still small and needs to be optimized.

Increase l2_reg_w:
l2_reg_w = 0.15
l2_reg_w = 0.19
l2_reg_w = 0.2
	1.3208
	1.3176
	1.3182
		0.0811
		0.0834
		0.0829
			Increasing l2_reg_w from 0.125 to 0.19 helps increase R^2 and reduce MSE_val.

Decrease l2_reg_V
l2_reg_V = 0.09
l2_reg_V = 0.085
l2_reg_V = 0.08
	1.3167
	1.3168
	1.3176
		0.083939
		0.083931
		0.083365
			Decreasing l2_reg_V from 0.1 to 0.09 achieves the best MSE_val and R^2 for $FM_{year\ only}$ model up to now.

Change random_state, decrease n_iter Old random_state = 123 New random_state = 456 n_iter = 15 n_iter = 10	1.30437 1.30997	0.0925 0.0887	Changing the random state and decreasing n_iter from 20 to 15 enhance the model performance, compared to the previous step.

<u>Conclusion 3:</u> Adding the "year" feature does not benefit the model's performance. In fact, the current best combination of hyperparameters (based on their performance on the validation set) can only give an MSE_test of 1.3635 and R^2_{test} of 0.08105 (Setting: n_iter = 15, init_stdev = 0.1, rank = 5, random_state = 456, l2_reg_w = 0.19, l2_reg_V = 0.09).

d). $FM_{with\ quarter}$: Factorization Machine method with “quarter” as another temporal feature, along with “year” (by *fastFM* package)

Model = $FM_{with\ quarter}$	**MSE_val**	R^2	**Comments**

n_iter = 20,
init_stdev = 0.1,
rank = 5,
l2_reg_w = 0.125,
l2_reg_V = 0.1
	1.3309	0.0741	The chosen model somehow fits the given input, but R^2 is still small and needs to be optimized.

Decrease l2_reg_V:
l2_reg_V = 0.09
l2_reg_V = 0.08
l2_reg_V = 0.05
	1.3272
	1.3231
	1.3196
		0.0766
		0.0795
		0.0820
			Decreasing l2_reg_V helps increase R^2 and reduce MSE_val.

<u>Conclusion 4:</u> Other ways of tuning hyperparameters do not give better values of MSE_val and R^2_{val} . Therefore, we stop examining this $FM_{with\ quarter}$ model here. With the setting of n_iter = 20, init_stdev = 0.1, rank = 5, random_state = 456, l2_reg_w = 0.125, l2_reg_V = 0.05, we get MSE_{test} = 1.3797 and R^2_{test}= 0.0702.

e). $FM_{temporal}$: Factorization Machine method with “year” and “month” as temporal components (by *fastFM* package)

Model = $FM_{temporal}$	**MSE_val**	R^2	**Comments**

n_iter = 20,
init_stdev = 0.1,
rank = 5,
l2_reg_w = 0.1,
l2_reg_V = 0.05
	1.3203	0.0815	The initial phase of this model works better than the initial phase of $FM_{with\ quarter}$.

Increase l2_reg_w:
l2_reg_w = 0.11
l2_reg_w = 0.12
	1.3182
	1.3193
		0.0830
		0.0822
			Increasing l2_reg_w to 0.11 gives better MSE_val and R^2.

Increase rank: rank = 10	1.3132 1.3121	0.0864 0.0871	Increasing the rank from 5 to

rank = 15
rank = 16
	1.3115	0.0876	16 enhances the values of evaluation metrics.

<u>Conclusion 5</u>: Overall, the model $FM_{temporal}$ beats $FM_{with\ quarter}$.

MSE_{test} = 1.3779, R^2_{test} = 0.0714 with n_iter = 17, init_stdev = 0.1, rank = 16, random_state = 456, l2_reg_w = 0.11, l2_reg_V = 0.05.

f). Baseline #1: Always use the median rating per user to predict a new rating.
MSE_val = 1.7901, R^2_{val} = -0.2454 ($R^2 < 0$: worse than fitting a horizontal line).
MSE_test = 1.8755, R^2_{test} = -0.2640

g). Baseline #2: Always use the average rating per user to predict a new rating.
MSE_val = 1.6733, R^2_{val} = -0.1641
MSE_test = 1.7536, R^2_{test} = -0.1818

Overall, our model $FM_{temporal}$ works better than the baselines #1 and #2 as it has smaller values of MSE_val and MSE_test as well as positive values of R^2_{test} and R^2_{val}. Among the models that we consider for our comparison, $FM_{temporal}$ beats $FM_{with\ quarter}$, but it does not work well as $FM_{year\ only}$, $FM_{no\ time}$, and SVD. From these comparisons, we realize that the temporal features do not contribute to enhancing the model's performance in this Google Local sub-dataset. In fact, the model becomes worse whenever we add some types of temporal components. For instance, adding the "year" component makes $FM_{year\ only}$ worse than $FM_{no\ time}$, and adding "quarter" along with "year" makes $FM_{with\ quarter}$ worse than $FM_{year\ only}$. As a result, we come to the conclusion that the "year" feature works fine; the "quarter" does not fit the predictive task of this dataset; the "month" works better than the "quarter", but it still does not give the desired result. In short, the temporal dynamics do not work well in our case, and from our observation, we understand that sometimes, a simpler model like SVD can give the best overall result. This reminds us of the case where a basic popularity-based recommender system can beat other complex similarity metrics when we tried to predict if a user has cooked a recipe or not. Therefore, we learn to flexibly adjust our model and always consider both non-temporal and temporal models to get the best possible result for our future predictive task.

Finally, we hope to express some thoughts about why our proposed model ($FM_{temporal}$) does not work as expected. We think the biggest challenge from this dataset is that it has quite a lot of null data in the "unixReviewTime" column. Specifically, we have to drop around 1000 rows of the training data due to the null values of unixReviewTime. Even though the remaining number of training feature vectors is well above 50,000, we fail to capture as much information as possible. Besides, during data pre-processing, we choose to impute the missing “unixReviewTime” of the validation and test sets. We used to think that the distance of two review times per user won’t go beyond three months, which means their corresponding review “year” and “quarter” would highly be the same. Hence, we “safely” assumed that we could use data imputation. However, the results of the model tell a different story: the temporal features we used did add some more noise to our data, and consequently, a simpler non-temporal model like SVD yields better performance.

Future directions of our project include (1) continuing optimizing the hyperparameters for SVD, (2) incorporating geographical features (i.e. “gps”) to our FM model, and (3) determining the influence of new features that will be used on our model and the Google Local sub-dataset.

6. References:

Chapter 5 Workbook
Julian McAuley
CSE 158 Lecture, 2021
website

Factorization Machines
Steffen Rendle
2010
pdf

Matrix Factorization-based algorithms — Surprise 1 documentation
Surprise, 2021
website

Personalized Machine Learning
Julian McAuley
CSE 158 Textbook, 2021
pdf

Translation-based factorization machines for sequential recommendation
Rajiv Pasricha, Julian McAuley
RecSys, 2018
pdf

Translation-based recommendation
Ruining He, Wang-Cheng Kang, Julian McAuley
RecSys, 2017
pdf