MOVIE RECOMMENDATION USING SINGULAR VALUE DECOMPOSITION

BRIAN, DARSH, MAGGIE, RICHA, SHUBHAM

Abstract. The aim of this project is to compare two methods for a recommendation system that predicts user movie ratings based on existing user data. The dataset used was very sparse, so this paper focuses on comparing two different methods that eliminate the sparsity and evaluate their accuracy of predicting movie ratings. Method 1 is to replace null values with average movie ratings and performing Singular Value Decomposition (SVD) on a subset of the large overall dataset. Method 2 is to utilize Stochastic Gradient Descent (SGD) to replace null values and then perform SVD on the entire dataset. We selected a group of ten ratings to replace with null and use as a set of data to compare each methods' accuracy. Our objective is to reduce sparsity and find the best method for prediction. We found that Method 1 predictions were farther from the actual ratings, but not too far off. Method 2 predictions were closer to the actual ratings more often, but when they were wrong, they were wrong by a lot.

1. Introduction.

We want to recommend a movie to a user, or know whether a user will like a certain movie, based on how they have rated previous movies. In our data set, (Movie, user) ratings can be represented by points in a matrix where the rows are the users, columns are the movies and ratings are the values. However, much of this matrix is sparse, because not all users have rated every movie. Therefore, we need to find way to predict values of missing data points, to predict how a specific user might rate that specific movie.

To do this prediction, we will use Singular Value Decomposition(SVD). SVD is a common method of Matrix factorization for recommendation systems. The model finds an association between the users and movies based on past ratings. Then it predicts the rating for the movie in which the user may be interested.

Since we are using the same procedure, we will be comparing the different ways of reducing the matrix to a dense form in order to perform SVD to find the predicted values. Once such way to deal with this is with Stochastic Gradient Descent.

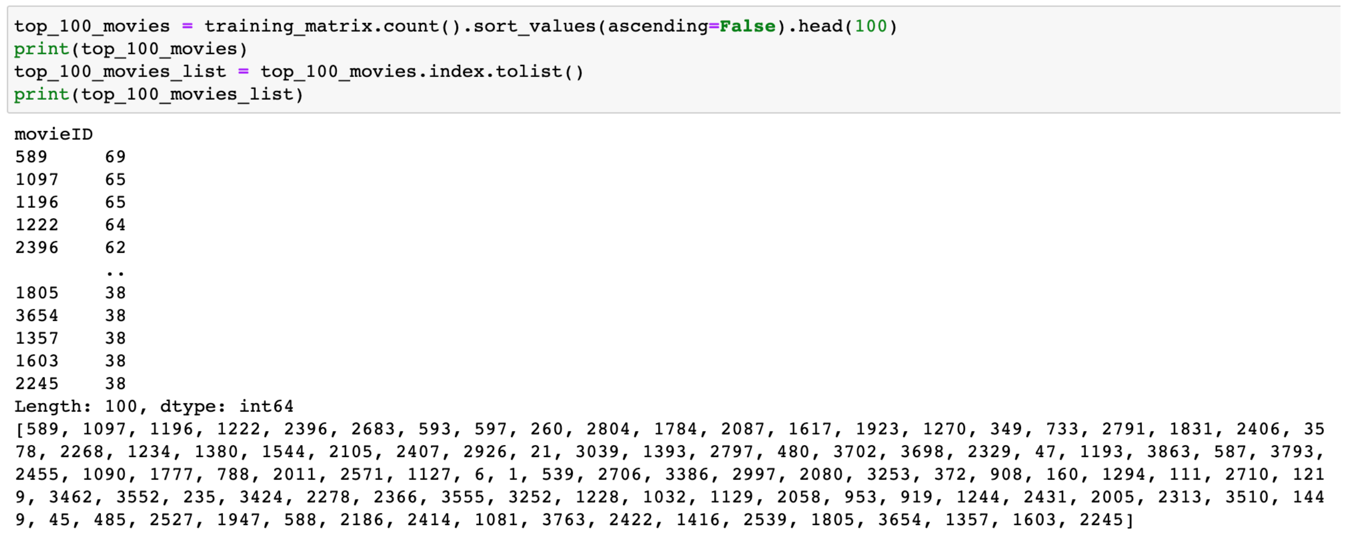
Stochastic Gradient Descent is an iterative method that optimizes learning parameters, sparse matrix, in order to find the best fit between the prediction and the actual data. In other words, SGD is used to convert numeric observations into predictive features which is why it is a very popular method used in any and all kinds of machine learning models.

Hypothesis. Performing Singular Value Decomposition (SVD) on a full dataset is beneficial for the accuracy of ratings of users on movies, in comparison to replacing null values with the movie's average rating and then performing SVD on a subset of movies.

2. Methods.

As our aim is to compare the accuracy of our methods to predict ratings, we will need actual ratings to verify accuracy. To properly evaluate our methods, we will remove 10 points from the matrix, and replace them with null values. In the context of this problem, we are pretending that 10 users each did not rate one of the movies they did rate. Then we will use each separate method to replace the null values, perform SVD on each matrix, and compare which method predicted these 10 ratings more accurately, comparing the predictions to the actual ratings we had previously extracted.

2.1. Method 1. In the first method, we began by decreasing our sample size. We chose to reduce the matrix of ratings to only include the 100 movies with the most ratings, not necessarily the best ratings.

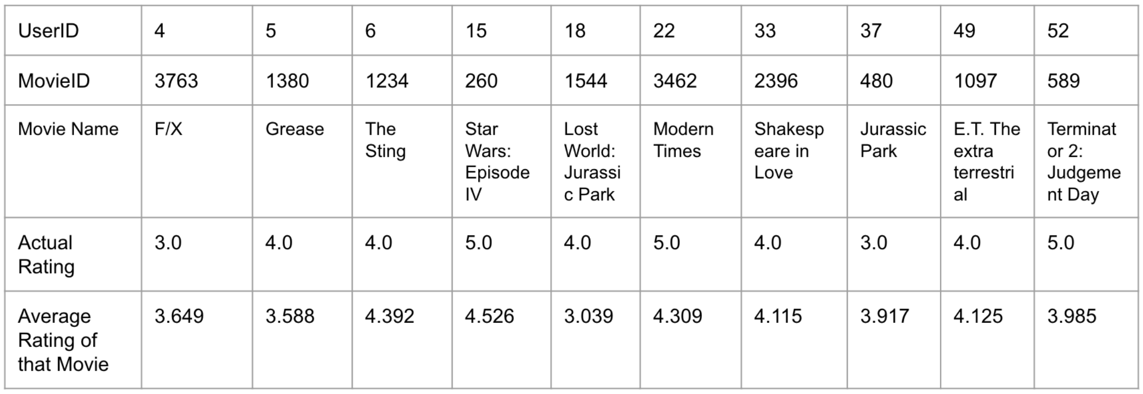


```
top_100_movies = training_matrix.count().sort_values(ascending=False).head(100)
print(top_100_movies)
top_100_movies_list = top_100_movies.index.tolist()
print(top_100_movies_list)
```

```
movieID
589     69
1097    65
1196    65
1222    64
2396    62
        ..
1805    38
3654    38
1357    38
1603    38
2245    38
Length: 100, dtype: int64
[589, 1097, 1196, 1222, 2396, 2683, 593, 597, 260, 2804, 1784, 2087, 1617, 1923, 1270, 349, 733, 2791, 1831, 2406, 35
78, 2268, 1234, 1380, 1544, 2105, 2407, 2926, 21, 3039, 1393, 2797, 480, 3702, 3698, 2329, 47, 1193, 3863, 587, 3793,
2455, 1090, 1777, 788, 2011, 2571, 1127, 6, 1, 539, 2706, 3386, 2997, 2080, 3253, 372, 908, 160, 1294, 111, 2710, 121
9, 3462, 3552, 235, 3424, 2278, 2366, 3555, 3252, 1228, 1032, 1129, 2058, 953, 919, 1244, 2431, 2005, 2313, 3510, 144
9, 45, 485, 2527, 1947, 588, 2186, 2414, 1081, 3763, 2422, 1416, 2539, 1805, 3654, 1357, 1603, 2245]
```

Fig. 1: The 10 extracted (User, Movie) Ratings

Next, we removed the following 10 ratings from the matrix, and replaced them with NaN. We took these points out so that we could later compare them to what the formula would predict for them.



UserID	4	5	6	15	18	22	33	37	49	52
MovieID	3763	1380	1234	260	1544	3462	2396	480	1097	589
Movie Name	F/X	Grease	The Sting	Star Wars: Episode IV	Lost World: Jurassi c Park	Modern Times	Shakesp eare in Love	Jurassic Park	E.T. The extra terrestri al	Terminat or 2: Judgeme nt Day
Actual Rating	3.0	4.0	4.0	5.0	4.0	5.0	4.0	3.0	4.0	5.0
Average Rating of that Movie	3.649	3.588	4.392	4.526	3.039	4.309	4.115	3.917	4.125	3.985

Fig. 2: Selecting 100 most rated movies

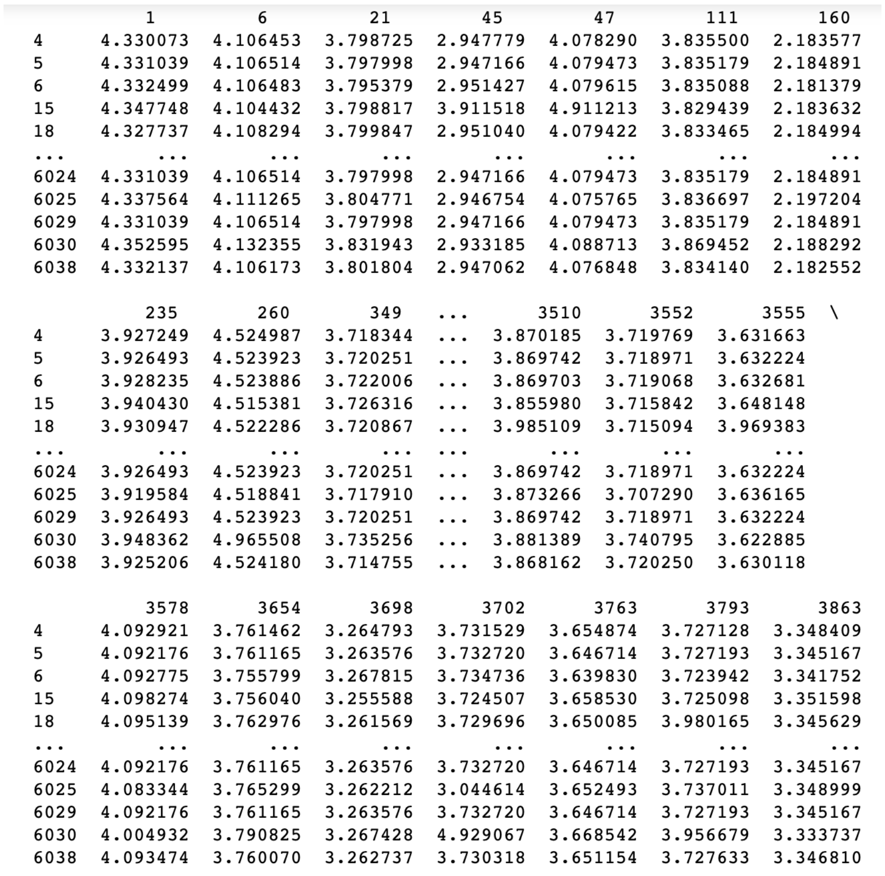
Lastly, to prepare the matrix for SVD, we replaced all null values in the matrix with the average of that movie's existing ratings. We did this because we assumed that a movie's likability is probably somewhat consistent. If a movie is bad, most users will rate it poorly, and if a movie is generally good, most users will rate it well. However, we acknowledge that this is a naive approach to filling in null values, because some users may not like a really good movie, and predicting the movie's high average rating for that user may not be an accurate approach.

After preparing our matrix by removing our 10 focus points and eliminating all null values, we performed Singular Value Decomposition on the matrix.

Singular Value Decomposition is a "popular method in linear algebra for matrix factorization in machine learning" and is used to reduce the space dimension of a large matrix[1]. It uses the principle of collaborative filtering. In collaborative filtering, the model assumes that movies that have been liked in the past will continue to be liked by others, and that users who liked certain movies in the past will like similar movies in the future. "The model finds an association between the users and the items", and then predicts how a user may rate a similar item[4].

Singular Value Decomposition decomposes the matrix, A of ratings into 3 matrices, U, S, and V^T, where $A = USV^T$. U is an "mxn matrix of the orthonormal eigenvectors of AA^T"[2], and it represents the relationship between users and latent factors[4]. S is a "diagonal matrix of the singular values which are the square roots of the eigenvalues of A^TA[2], and it represents the strength of each latent factor[4]. V^T is the "transpose of a nxn matrix containing the orthonormal eigenvectors of A^TA"[2], which represents the similarity between items and latent factors[4].

With this decomposition, we can compute the dot product of the matrices, using only the diagonal values from S, to compute a reconstructed prediction matrix.



```
            1         6        21        45        47       111       160
4    4.330073  4.106453  3.798725  2.947779  4.078290  3.835500  2.183577
5    4.331039  4.106514  3.797998  2.947166  4.079473  3.835179  2.184891
6    4.332499  4.106483  3.795379  2.951427  4.079615  3.835088  2.181379
15   4.347748  4.104432  3.798817  3.911518  4.911213  3.829439  2.183632
18   4.327737  4.108294  3.799847  2.951040  4.079422  3.833465  2.184994
...       ...       ...       ...       ...       ...       ...       ...
6024 4.331039  4.106514  3.797998  2.947166  4.079473  3.835179  2.184891
6025 4.337564  4.111265  3.804771  2.946754  4.075765  3.836697  2.197204
6029 4.331039  4.106514  3.797998  2.947166  4.079473  3.835179  2.184891
6030 4.352595  4.132355  3.831943  2.933185  4.088713  3.869452  2.188292
6038 4.332137  4.106173  3.801804  2.947062  4.076848  3.834140  2.182552

          235       260       349  ...      3510      3552      3555  \
4    3.927249  4.524987  3.718344  ...  3.870185  3.719769  3.631663
5    3.926493  4.523923  3.720251  ...  3.869742  3.718971  3.632224
6    3.928235  4.523886  3.722006  ...  3.869703  3.719068  3.632681
15   3.940430  4.515381  3.726316  ...  3.855980  3.715842  3.648148
18   3.930947  4.522286  3.720867  ...  3.985109  3.715094  3.969383
...       ...       ...       ...  ...       ...       ...       ...
6024 3.926493  4.523923  3.720251  ...  3.869742  3.718971  3.632224
6025 3.919584  4.518841  3.717910  ...  3.873266  3.707290  3.636165
6029 3.926493  4.523923  3.720251  ...  3.869742  3.718971  3.632224
6030 3.948362  4.965508  3.735256  ...  3.881389  3.740795  3.622885
6038 3.925206  4.524180  3.714755  ...  3.868162  3.720250  3.630118

         3578      3654      3698      3702      3763      3793      3863
4    4.092921  3.761462  3.264793  3.731529  3.654874  3.727128  3.348409
5    4.092176  3.761165  3.263576  3.732720  3.646714  3.727193  3.345167
6    4.092775  3.755799  3.267815  3.734736  3.639830  3.723942  3.341752
15   4.098274  3.756040  3.255588  3.724507  3.658530  3.725098  3.351598
18   4.095139  3.762976  3.261569  3.729696  3.650085  3.980165  3.345629
...       ...       ...       ...       ...       ...       ...       ...
6024 4.092176  3.761165  3.263576  3.732720  3.646714  3.727193  3.345167
6025 4.083344  3.765299  3.262212  3.044614  3.652493  3.737011  3.348999
6029 4.092176  3.761165  3.263576  3.732720  3.646714  3.727193  3.345167
6030 4.004932  3.790825  3.267428  4.929067  3.668542  3.956679  3.333737
6038 4.093474  3.760070  3.262737  3.730318  3.651154  3.727633  3.346810
```

Fig. 3: The recomposed matrix from SVD

2.2. Method 2. Utilize Stochastic Gradient Descent (SGD) to replace the null values and then SVD on the full matrix.

For the second method, we applied Stochastic Gradient Descent (SGD) to the entire data set in order to retrieve the predicted values for the user item matrix. SGD is an iterative method to optimize a loss function[3]. It approximates the true gradient of the function with a randomized sub sample of the gradients. In our problem, the gradients that we are estimating are the vectors U and V^T. Like SVD, we are first finding the dot product of the U matrix and the V^T, but the main difference is that the latent factors U and V^T matrices are randomly initialized, instead of them being in the S matrix in regular SVD. The error of this dot product is then minimized using Stochastic gradient descent and the vectors are updated. The gradient descent is done by selecting randomized points in each iteration to ensure faster computational complexity at the cost of a lower convergence rate.

With each iteration of SGD the error between the prediction and the actual matrix is reduced resulting in a step towards the optimal solution of the prediction. The optimal solution of prediction is when the initial matrix values have little or no difference between the prediction matrix values, only considering the non-zero values in the initial matrix. The formula for SGD is as follows[3]:

$$W_{i+1} = W_i + lr * \frac{\partial Loss}{\partial W}$$

$$W_{i+1} = Current\,pred, W_i = Previous\,pred, Loss = RMSE(Input\,matrix, Pred), lr = 0.001$$

This equation is linearly reducing the error with every iteration. 'lr' is the learning rate i.e. the size of step on the x-axis. RMSE is the root mean squared error. As SGD progresses the error exponentially reduces until it is very close to 0, which is good enough to make a valid prediction.

2.2.1. Implementation.

We use Pandas pivot method to create index/column data frame and fill the missing play counts with 0.

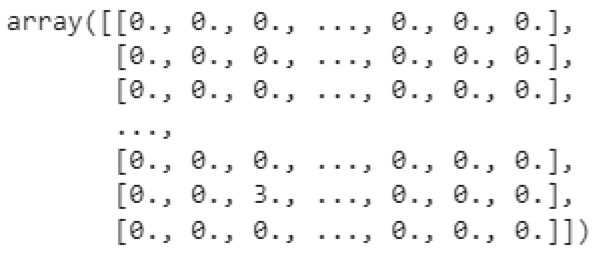

```
array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 3., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]])
```

Fig. 4: Dataset filled with 0 and converted to np.array

Defining train test split: We have used a custom train test split function to get the training and validation set to evaluate our model. We remove the existing movie ratings from the dataset by replacing them with zero and adding the same to our validation set. The *MIN_USER_RATINGS* variable is used to select the users we will be considering to split our data on, and the *DELETE_USER_RATINGS* variable is used to select the size of validation values we will be extracting from each row(user) selected.

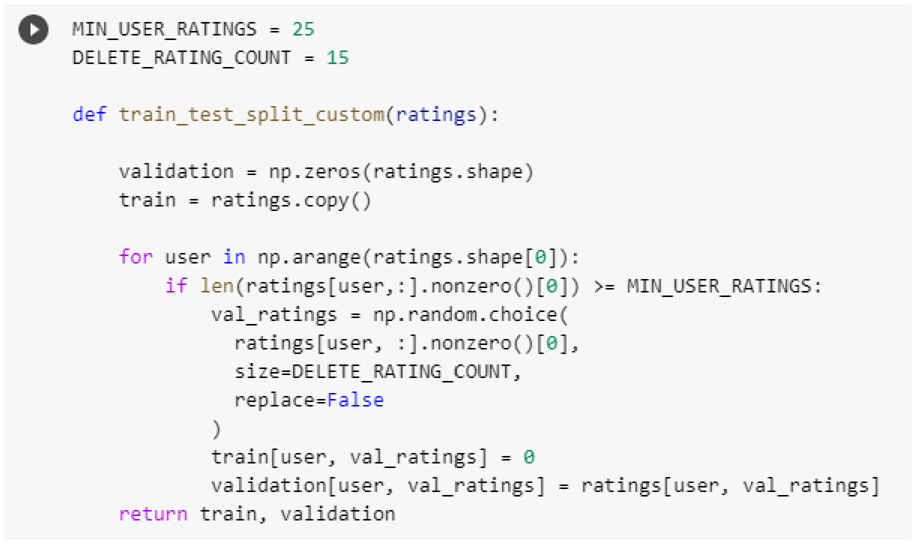
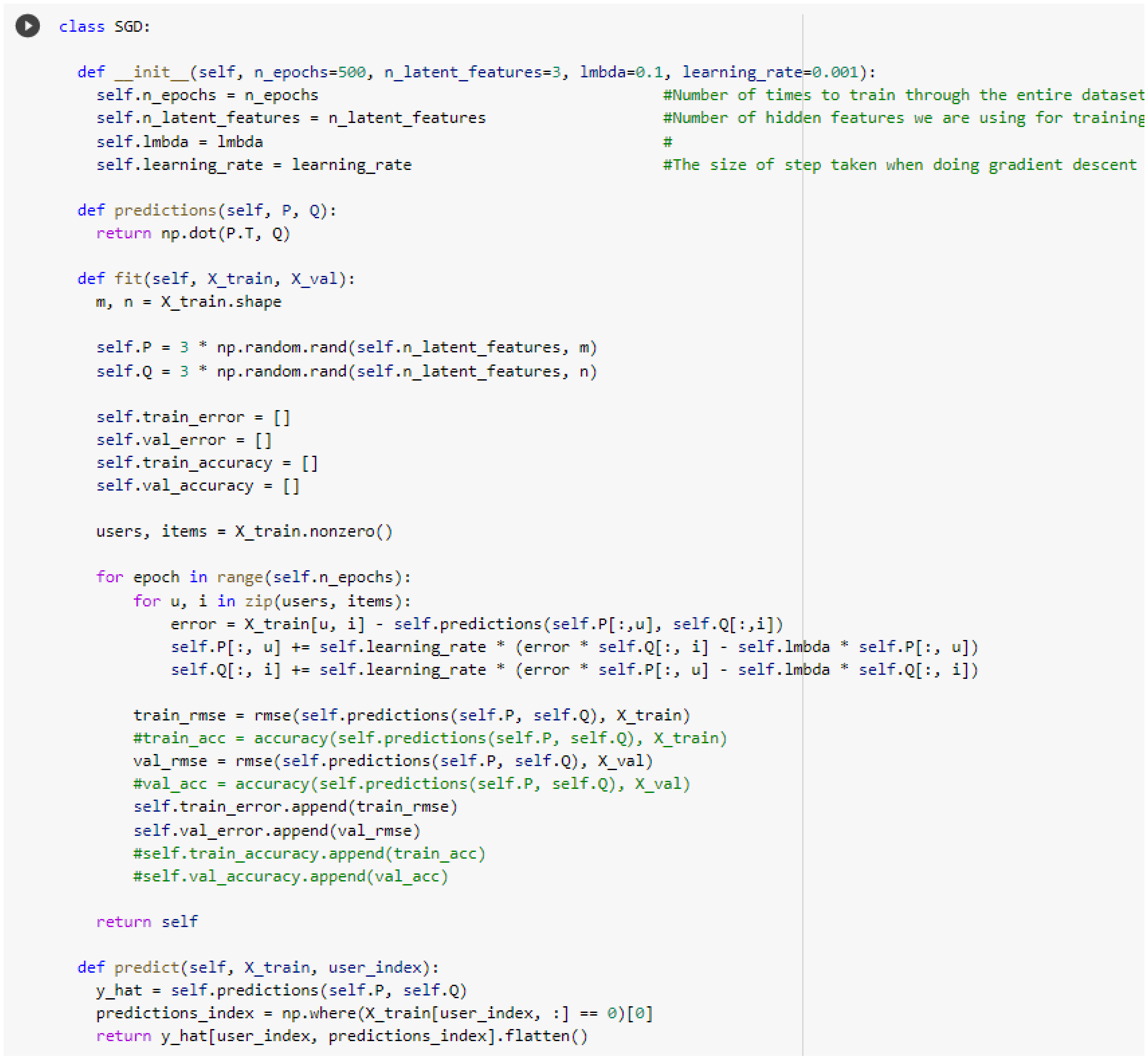

```
MIN_USER_RATINGS = 25
DELETE_RATING_COUNT = 15

def train_test_split_custom(ratings):

    validation = np.zeros(ratings.shape)
    train = ratings.copy()

    for user in np.arange(ratings.shape[0]):
        if len(ratings[user,:].nonzero()[0]) >= MIN_USER_RATINGS:
            val_ratings = np.random.choice(
              ratings[user, :].nonzero()[0],
              size=DELETE_RATING_COUNT,
              replace=False
            )
            train[user, val_ratings] = 0
            validation[user, val_ratings] = ratings[user, val_ratings]
    return train, validation
```

Fig. 5: Train Test Split Function

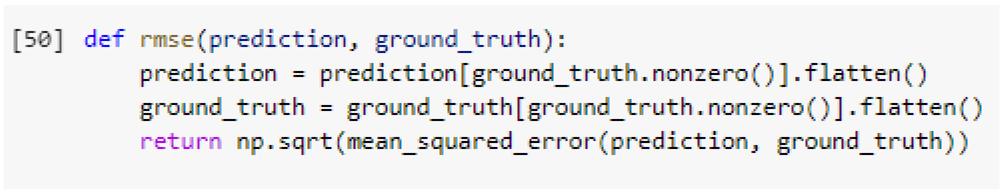
Training We start by creating 2 matrices for the latent features of the users and ratings. For each user, item pair, we calculate the error (note we use a simple difference of the existing and predicted ratings). We then update P and Q using Gradient Descent. As we discussed earlier, we obtain predictions by taking the dot product of the transposed P matrix with Q. After each training epoch, we calculate the training and validation errors and store their values to analyze later.



```
class SGD:

  def __init__(self, n_epochs=500, n_latent_features=3, lmbda=0.1, learning_rate=0.001):
    self.n_epochs = n_epochs                                  #Number of times to train through the entire dataset
    self.n_latent_features = n_latent_features                #Number of hidden features we are using for training
    self.lmbda = lmbda                                        #
    self.learning_rate = learning_rate                        #The size of step taken when doing gradient descent

  def predictions(self, P, Q):
    return np.dot(P.T, Q)

  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 = []
    self.train_accuracy = []
    self.val_accuracy = []

    users, items = X_train.nonzero()

    for epoch in 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)
        #train_acc = accuracy(self.predictions(self.P, self.Q), X_train)
        val_rmse = rmse(self.predictions(self.P, self.Q), X_val)
        #val_acc = accuracy(self.predictions(self.P, self.Q), X_val)
        self.train_error.append(train_rmse)
        self.val_error.append(val_rmse)
        #self.train_accuracy.append(train_acc)
        #self.val_accuracy.append(val_acc)

    return self

  def predict(self, X_train, user_index):
    y_hat = self.predictions(self.P, self.Q)
    predictions_index = np.where(X_train[user_index, :] == 0)[0]
    return y_hat[user_index, predictions_index].flatten()
```

Fig. 6: SGD Implementation

Evaluation: As, we are predicting the value for multiple movies, it becomes a multi-class evaluation and Root Mean Square error is the easiest way to evaluate our results.



```
[50] def rmse(prediction, ground_truth):
         prediction = prediction[ground_truth.nonzero()].flatten()
         ground_truth = ground_truth[ground_truth.nonzero()].flatten()
         return np.sqrt(mean_squared_error(prediction, ground_truth))
```

Fig. 7: Evaluation Function using RMSE

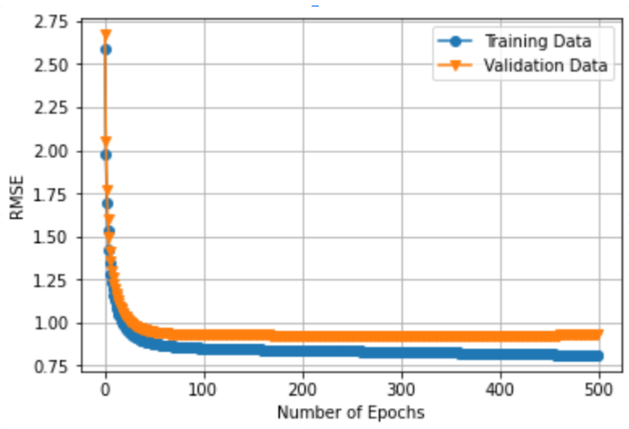


Fig. 8: Evaluation of our Model using test and validation

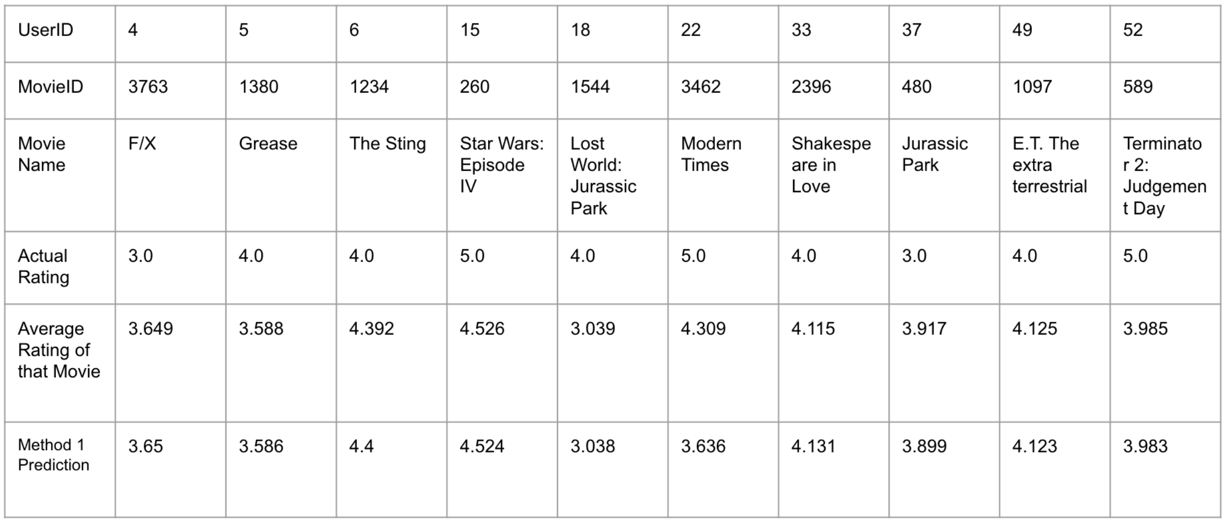
Observations: The main problem with SGD and the gradient descent process in general is the learning rate parameter. Setting the learning rate too high can result in over fitting or having the algorithm diverge. Setting it too low can slow down the convergence rate[5]. A solution to this is to add a decayed learning rate function, where the algorithm is trained with a large learning rate in the first iterations and the later ones will just slightly optimize the learning rate. During our implementation, we observed that a polynomial learning rate is more efficient and reaches the minima sooner than an exponential one (200 epochs), which didn't reach the minima even after 1000 epochs.

In addition to that, there are certain limitations observed while trying to find the best loss function which cannot be forgotten about while implementing SGD. To perform SGD, a loss function, in this case RMSE, needs to be differentiable. If it is not differentiable, SGD will not be able progress toward the optimal solution. At the point of non-linearity, the derivative of loss would evaluate to infinity or 0, which would break SGD.

3. Results.

3.1. Method 1.

Let us take a look at the 10 predicted ratings using method 1 to see how close the predictions are:

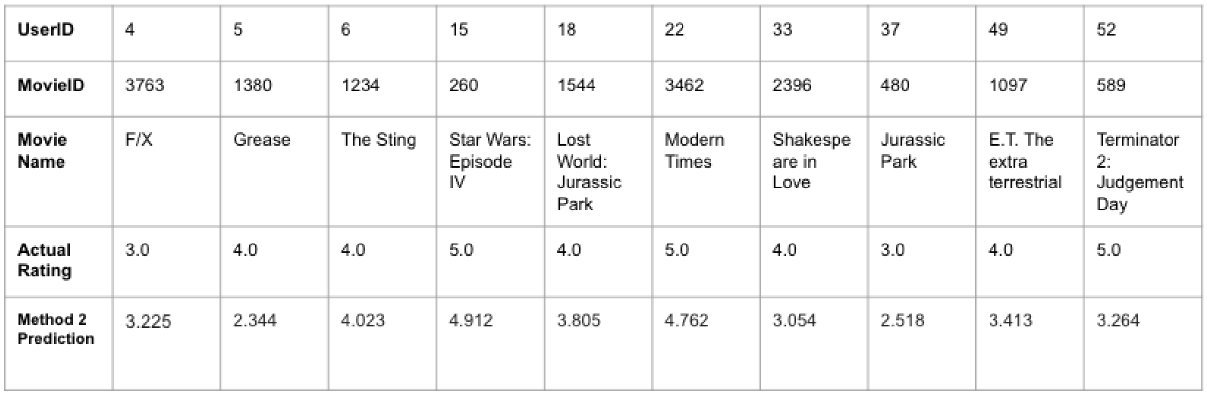


UserID	4	5	6	15	18	22	33	37	49	52
MovieID	3763	1380	1234	260	1544	3462	2396	480	1097	589
Movie Name	F/X	Grease	The Sting	Star Wars: Episode IV	Lost World: Jurassic Park	Modern Times	Shakespeare in Love	Jurassic Park	E.T. The extra terrestrial	Terminator 2: Judgement Day
Actual Rating	3.0	4.0	4.0	5.0	4.0	5.0	4.0	3.0	4.0	5.0
Average Rating of that Movie	3.649	3.588	4.392	4.526	3.039	4.309	4.115	3.917	4.125	3.985
Method 1 Prediction	3.65	3.586	4.4	4.524	3.038	3.636	4.131	3.899	4.123	3.983

Fig. 9: The 10 Predicted Ratings

After looking at these predictions, we can notice a few things. First, we can see that the predicted rating is not too far off from the actual rating. In fact, the Root Mean-Squared Error is 0.75108, which is not very large. Second of all, however, we can see that the predicted ratings are extremely similar to the average ratings of their respective movies. This leads us to believe that our naive approach in replacing nulls with movie averages may have caused a large bias in the predictions from our model. The model's predictions appear to be heavily influenced by the averages, which as we discussed before, are not always an accurate reflection of how a specific user might rate that movie.

3.2. Method 2. The results for method 2 were quite promising. After training through the entire data for several hundred epochs, we were able to get the error between prediction and actual values down to ± 0.85. This error could have been smaller had we used a polynomial learning rate which would dynamically check multiple local minimas and reach a global minima or a get the least error of them all. However, this was not possible as it requires a lot more processing power to perform SGD with varying learning rate.



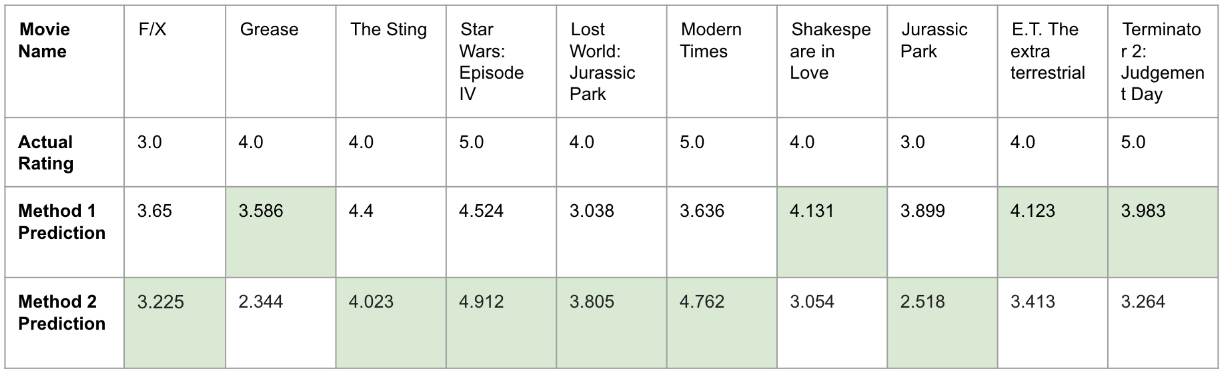
UserID	4	5	6	15	18	22	33	37	49	52
MovieID	3763	1380	1234	260	1544	3462	2396	480	1097	589
Movie Name	F/X	Grease	The Sting	Star Wars: Episode IV	Lost World: Jurassic Park	Modern Times	Shakespeare in Love	Jurassic Park	E.T. The extra terrestrial	Terminator 2: Judgement Day
Actual Rating	3.0	4.0	4.0	5.0	4.0	5.0	4.0	3.0	4.0	5.0
Method 2 Prediction	3.225	2.344	4.023	4.912	3.805	4.762	3.054	2.518	3.413	3.264

Fig. 10: The 10 Predicted Ratings after SGD

According to figure 10, we observe that even though the predicted values are not exactly the same they are very similar to the actual values with ±0.6 deviation for most of them. When the values are deviating more than the ±0.6, they are far off by a lot. This makes them the outliers. Since 30% of the outputs are such points in the 10 ratings we are evaluating, we can say that the local minimum or the optimal solution was not reached. However, we got pretty close to the optimal solution where all predicted values would have been very close to the actual ratings of the movies.

The results reveal that the hyperparameters such as the random guess, whether learning rate is so large that it misses the local minima or so small that it never reaches the local minima, play a huge role in finding the local minima of the loss function. In this case, SGD successfully produced a prediction with low error by learned about what users will like based on each users' past ratings but was still further away from a lot of the actual ratings.

3.3. Comparison. Now, let's compare the predictions from each method on our 10 focus points. In the figure below, green represents the prediction that was closer to the actual rating.



Movie Name	F/X	Grease	The Sting	Star Wars: Episode IV	Lost World: Jurassic Park	Modern Times	Shakespeare in Love	Jurassic Park	E.T. The extra terrestrial	Terminator 2: Judgement Day
Actual Rating	3.0	4.0	4.0	5.0	4.0	5.0	4.0	3.0	4.0	5.0
Method 1 Prediction	3.65	🟩 3.586	4.4	4.524	3.038	3.636	🟩 4.131	3.899	🟩 4.123	🟩 3.983
Method 2 Prediction	🟩 3.225	2.344	🟩 4.023	🟩 4.912	🟩 3.805	🟩 4.762	3.054	🟩 2.518	3.413	3.264

Fig. 11: Predicted Ratings from Both Methods

Method 1 was closer 4 out of 10 times, and had an RMSE of 0.75108. Method 2 was closer 6 out of 10 times, and had an RMSE of 0.85915. While Method 2 predicted closer to the actual rating more of the time, Method 1 has a smaller RMSE. Here, we can conclude that although Method 2 was farther less of the time, but when it is wrong, it is very wrong. On the other hand, when method 1 was farther from the actual rating than method 2, it was not too far off.

4. Summary. After attempting both methods on the matrix, we find it important to first note that method 1 only used the 100 most rated movies, whereas method 2 looked at the entire dataset. With that being said, through this comparison, we when we tried *Method 1: Replacing Nulls with Average Ratings*, the predicted ratings were heavily influenced by the average ratings, but were not too far off from the actual ratings.

When we tried *Method 2: Using SGD to Replace Nulls*, we found that this method has predictions closer to the actual values more of the time; however, when it is wrong, it is wrong by a lot. The added value of an entire dataset and the intricacy of Stochastic Gradient Descent seems to minimal and, in fact, leads to poorer generalization overall. There are two possible explanations for this: either the size of the dataset is too large or the method of using SGD to replace null values is not a proper method.

This leads us to ask an important question: do we really need massive data analysis methods or is it better to just pick a subset of movies to look at? There is a lot of exploration that can be done further to understand these two methods; for example, we could look at more than ten points to evaluate accuracy, we can attempt SGD on a subset of a large matrix, and we can look at other statistical techniques.

REFERENCES

[1] *Singular value decomposition (svd)*. https://towardsdatascience.com/recommender-system-singular-value-decomposition-svd-truncated-svd-97096338f361. Accessed: 2022-05-02.

[2] *Singular value decomposition (svd)*. https://www.geeksforgeeks.org/singular-value-decomposition-svd/. Accessed: 2022-05-02.

[3] N. Hug, *Understanding matrix factorization for recommendation (part 3) - svd for recommendation*. http://nicolas-hug.com/blog/matrix_facto_3. Accessed: 2022-05-03.

[4] D. V. Kumar, *Singular value decomposition (svd) its application in recommender system*. https://analyticsindiamag.com/singular-value-decomposition-svd-application-recommender-system/. Accessed: 2022-05-02.

[5] J. Zhang, *Stochastic gradient descent momentum explanation*. https://towardsdatascience.com/stochastic-gradient-descent-momentum-explanation-8548a1cd264e. Accessed: 2022-05-03.