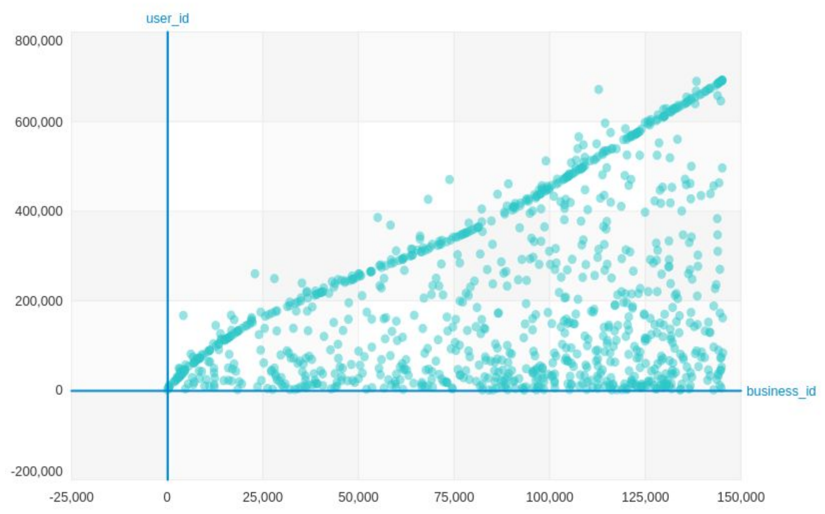
Data Mining Report - Recommendation System

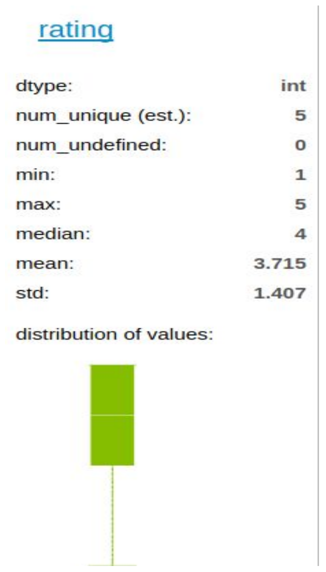
---

1. Keys Steps of Approach

1.1 Data Analysis

We started this problem with analyzing the data provided.The second figure show some stats for the ratings, as can be seen below if we just provide the mean the **RMSE** is expected to be around 1.407 **(std)** on Kaggle. The training data contains ratings for **77541 unique users** and **50017 unique business**. The total possible pairs of business-users are **3778334197** while the training data only contains **2038130,** hence the utility matrix is **99.94% sparse** which can be seen in the figure below.





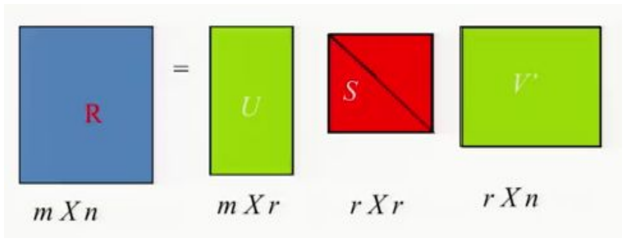
rating

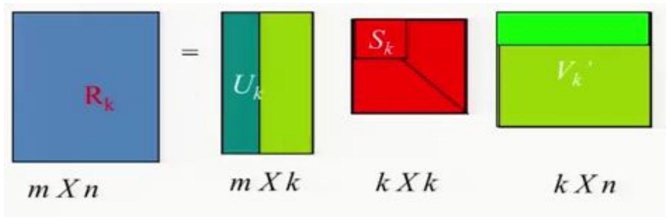
dtype:	int
num_unique (est.):	5
num_undefined:	0
min:	1
max:	5
median:	4
mean:	3.715
std:	1.407

distribution of values:

1.3 Algorithm for recommendations systems

In Latent Factors Algorithm we convert the **Utility Matrix R (m x n)** into two smaller matrix (m x k) and (k x n)**.** So we create a feature matrix S **(r x r)** which is just a triangular with each value in this matrix representing the importance of the feature.





In expressing the matrix R, more specifically, if we sort those values by decreasing absolute value and cut the matrix S off at some k dimensions. So we get a smaller matrix and in turn we cut off user and business matrix at k dimensions. Doing this we just compress all our matrix to just k dimensions. **Now with just dot product of U_k and V_k we predict the ratings for m users and n business.** This provide a major benefit of SVD resulting in small and fast model and more similar items moreover we switched from a sparse utility matrix to a dense user-feature and business-feature matrix, though we need to decide on **rank k** the number of latent features.

1.3.1 Singular Value Decomposition (SVD) using Surprise

We have used Surprise package SVD matrix factorization model. The term Surprise stands for “Simple python recommendation system engine”. This algorithm was popularized by Simon Funk during the Netflix Challenge.To estimate all the unknown, we minimize the following regularized squared error:

$$\sum_{r_{ui} \in R_{train}} (r_{ui} - \hat{r}_{ui})^2 + \lambda \left(b_i^2 + b_u^2 + ||q_i||^2 + ||p_u||^2 \right)$$

Now, the minimization is performed by SGD (stochastic gradient descent),

$$\begin{aligned} b_u &\leftarrow b_u + \gamma(e_{ui} - \lambda b_u) \\ b_i &\leftarrow b_i + \gamma(e_{ui} - \lambda b_i) \\ p_u &\leftarrow p_u + \gamma(e_{ui} \cdot q_i - \lambda p_u) \\ q_i &\leftarrow q_i + \gamma(e_{ui} \cdot p_u - \lambda q_i) \end{aligned}$$

The following parameters are tuned for achieving best results

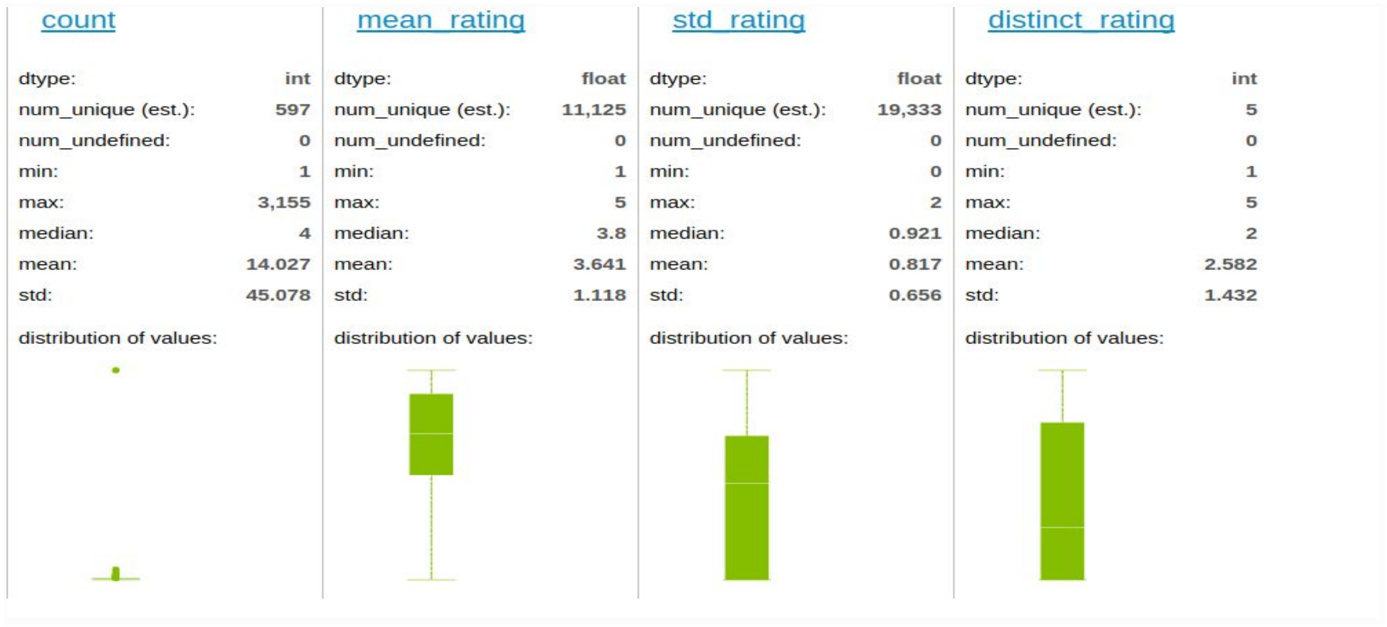
- **n_factors** – The number of factors. Default is 100.
- **n_epochs** – The number of iteration of the SGD procedure. Default is 20.
- **biased** (*bool*) – Whether to use baselines (or biases). Default is True.
- **init_std_dev** – The standard deviation of the normal distribution for factor vectors initialization. Default is 0.1
- **lr_all** – The learning rate for all parameters. Default is 0.005
- **reg_all** – The regularization term for all parameters. Default is 0.02
- **lr_bu** – The learning rate for bu. Default is None
- **lr_bi** – The learning rate for bi .Default is None
- **lr_pu** – The learning rate for pu. Default is None
- **lr_qi** – The learning rate for qi. Default is None
- **reg_bi** – The regularization term for bi. Default is None
- **reg_pu** – The regularization term for pu. Default is None
- **reg_qi** – The regularization term for qi. Default is None
- **verbose** – If True prints the current epoch. Default is False.

The best RMSE error we got with SVD is 1.29439.

1.3.2 Matrix Factorization using Graph labs

A Factorization Recommender learns latent factors for each user and item and uses them to make rating predictions. We chose this model from graph labs because **this model enables us to provide side features with our input data that is date converted in unix timestamp and year of review and** also it enables us to provide **user features and business features.** We chose this feature to ensure features impact on the predictions **so that if it works we can extract features for each business from reviews by creating TF.IDF profiles for each business.**

Some of the features extracted are shown below for businesses, similar features were also extracted for users



	count	mean_rating	std_rating	distinct_rating
dtype:	int	float	float	int
num_unique (est.):	597	11,125	19,333	5
num_undefined:	0	0	0	0
min:	1	1	0	1
max:	3,155	5	2	5
median:	4	3.8	0.921	2
mean:	14.027	3.641	0.817	2.582
std:	45.078	1.118	0.656	1.432
distribution of values:				

The predicted score for user *i* on item *j* is given by

$$score(i,j) \;=\; \mu \;+\; w_i \;+ w_j \;+ a^T x_i \;+ b^T y_j \;+ u_i{}^T v_j$$

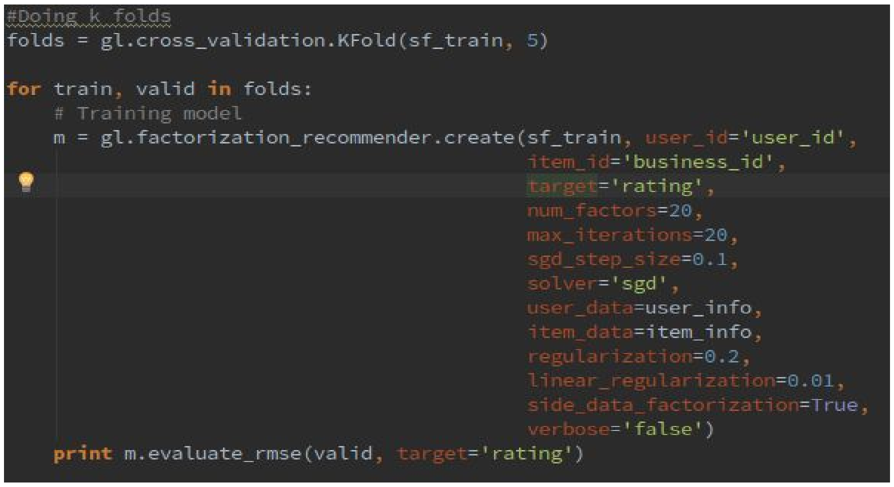
where μ is a global bias term, w_i is the weight term for user *i*, w_j is the weight term for item j, $\mathbf{x}_i$ and $\mathbf{y}_j$ are respectively the user and item side feature vectors, and **a** and **b** are respectively the weight vectors for those side features. The latent factors, which are vectors of length **num_factors**, are given by ui and vj. The model optimizes for the function given below

$$\min_{\mathbf{w},\mathbf{a},\mathbf{b},\mathbf{V},\mathbf{U}} \frac{1}{|\mathcal{D}|} \sum_{(i,j,r_{ij})\in\mathcal{D}} \mathcal{L}(\text{score}(i,j), r_{ij}) + \lambda_1(\|\mathbf{w}\|_2^2 + ||\mathbf{a}||_2^2 + ||\mathbf{b}||_2^2) + \lambda_2\left(\|\mathbf{U}\|_2^2 + \|\mathbf{V}\|_2^2\right)$$

where D is the observation dataset, *rij* is the rating that user *i* gave to item *j*, **U**=(**u**1,**u**2,...) denotes the user's latent factors and **V**=(**v**1,**v**2,...) denotes the item latent factors. *λ*1 denotes the linear_regularization parameter and *λ*2 the regularization parameter.

The choice we made for the solver to train our model is Stochastic Gradient Descent(sgd). The code that we use to create our model is given below. We create 5 folds for cross validation and create our model. The rmse for each testset **(valid)** during k-fold is around 0.91. Surprisingly on kaggle we get a score of around 1.41. We tuned the hyperparameters but were not able to improve our score which is a primary reason we moved to surprise.

The user_info and item_info are the features we created as discussed above for business and users. **With side data factorization True it include date in unix timestamp and year we provided in our training data and include it as features along with other linear terms.**



```
#Doing k folds
folds = gl.cross_validation.KFold(sf_train, 5)

for train, valid in folds:
    # Training model
    m = gl.factorization_recommender.create(sf_train, user_id='user_id',
                                            item_id='business_id',
                                            target='rating',
                                            num_factors=20,
                                            max_iterations=20,
                                            sgd_step_size=0.1,
                                            solver='sgd',
                                            user_data=user_info,
                                            item_data=item_info,
                                            regularization=0.2,
                                            linear_regularization=0.01,
                                            side_data_factorization=True,
                                            verbose='false')
    print m.evaluate_rmse(valid, target='rating')
```

In code shown num_factors are number of latent factors, **sgd_step_size** is the learning rate for sgd, **user_data** and **item_data** are the features we created as discussed above.

1.3.3 Spark MlLib ALS :

Our second approach is using Spark MLlib which implements a collaborative filtering algorithm called Alternating Least Squares (ALS). ALS works by trying to find the optimal representation of a user and a product matrix – which when combined, should accurately represent the original dataset. ALS is an iterative algorithm. In each iteration, the algorithm alternatively fixes one factor matrix and solves for the other, and this process continues until it converges.The unknown ratings can subsequently be computed by multiplying these factors. In this way, we can recommend products based on the predicted ratings. MLlib features a blocked implementation of the ALS algorithm that leverages Spark's efficient support for distributed, iterative computation.

We tuned the parameters and hyperparameters to minimize the RMSE error. We tried the following combinations to tune our model:

Parameter	**List of values**
Iterations	10, 15, 20
Regularization	1.0, 0.1, 0.01
Rank	12, 16, 20

The best model was achieved with below parameters :

Iterations	**Regularization**	**Rank**	**Train Error**
20	0.1	20	1.6551708468587

Conclusion : Below is the comparison of all the algorithms we used.

Library / Algorithms	**GraphLab Matrix Factorization**	**Surprise MlLib / ALS**	**Surprise SVD-PP**	**Surprise SVD**
RMSE	1.41	1.65	1.31	1.29