Movie Recommendation with Limited User Information

Tianchang Li
tli289@wisc.edu

Shixuan Song
ssong85@wisc.edu

Maoze Wang
mwang373@wisc.edu

Xuchen Xue
xxue8@wisc.edu

Abstract

The project aims to build a movie recommendation system by limited user information. In other words, the main work of this project is to find a relatively good performed model by implementing machine learning algorithms, and the selected model would be used in movie recommendation system. We proposed some supervised learning algorithms, including decision tree, logistic regression, K-nearest neighbors, and random forest, as well as limited unsupervised learning algorithms like clustering. In supervised learning algorithms we set the user rating as the target variable because our target is to find a model that could predict the user rating well. Steps of building these models are slightly different; however, they are parallel from a broad perspective. Steps are splitting the data sets into train and test sets, fitting the model by different algorithms, tuning the best parameters, and computing the average rating accuracy from all users. Among all models, the KNN model provided a highest overall accuracy of 90%, while others algorithms had a range of overall accuracy from 70% to 50%. Therefore, for the given data set, we select KNN model to predict the class of movie ratings.

1. Introduction

During the last few decades, the rise of internet and web services such as YouTube, Netflix, and many other big technology companies had widely used recommendation systems to increase sales and revenue. Moreover, recommendation systems also help people live a convenient life (Rocca, 2019)[11]. For example, when a potential customer is viewing the Amazon website, the recommendation systems employ proper algorithm(s) to recommend products to the customer.

Recommendation system is crucial for online film market. Many video streaming companies like Hulu, and Netflix are using recommendation systems to generate recommendations to their customers. The video streaming industry is growing and is expected to grow at approximately USD 82 Billion by 2023, at 17% of the compound annual growth rate (CAGR) between 2017 and 2023(Vikash,2019) [16]. For example, Netflix held a data challenge [1] which to find a recommendation system from the public that performs better than its system. The prize for the winner was 1 million dollars. This project aimed to build a movie recommendation mechanism within Netflix. The given datasets consisted views from over 17K movies and 500K+ customers. Several recommendation models offered from users in Kaggle were based on collaborative filtering and Pearson's R correlations [2].

Therefore, it would be practical to construct a movie recommendation system with machine learning. Techniques used in our project were mainly focusing on the supervised learning algorithms, like Logistic Regression, K- Nearest Neighbors, Decision Tree Regression, Decision Tree Classification, and Random Forest. Also, we tried unsupervised learning algorithms which were uncovered in our class, like clustering. These algorithms will be discussed in details in the section "Proposed Method." Our target on this project is to compare model performances produced by different algorithms, then choose a best performed model for generalization. Again, details of the project result will be discussed in the sections "Results and Discussion" and "Conclusion."

1.1. Motivation

The goal of this project is to create a movie recommendation model based on user rating rather than user personal information. Recommendation systems are supposed to rely less on personal information due to the rising concerns on the usage of personal data. We, as users of those streaming services, are not willing to provide streaming services with too much personal information. Moreover, some third party organization may also want to implement movie recommendation system to improve their user experiences. However, without access to user information like some streaming companies like Netflix, designing a recommender without using user information would be practical to offer better service to users.

[1] https://www.kaggle.com/netflix-inc/netflix-prize-data

[2] https://www.kaggle.com/laowingkin/netflix-movie-recommendation

2. Related Work

2.1. Recommendation System Using Logistic Regression and the Hashing Trick

For using logistic regression, they used the binary result for prediction with 1(will watch) and 0(will not watch). Based on a data set they used, there will be N users, M movies, and K features per movie (Nowling Geromel, 2016) [8]. Thus the vector of the feature will be MK entries. When the researchers try to study a pair of users and movies (u,m), the indices of the feature will contain the indices of [mK,(m+1)K]. Since the logistic regressions are based on each movie, there will be M regression models created in the research. As the researchers indicated in the report, "In our case, for each movie m, we will use the interactions with the M1 other movies as the features. Thus, K=M1 and our vector will have M(M1) entries" (Nowling Geromel, 2016) [8]. There will be a problem with memory usage since this will be computationally expensive to create a regression model for each movie. Therefore, there will be impossible to build a model for every movie.

For this reason, the hashing technique is used. The number of interactions will not be significant, which merely a small part of M-1 features in each movie. The researchers generate a small vector 2 to the B entries for each user and movie pairs, which B could be considered as a new parameter in the model. For every movie that is not equal to m that the user watched, researches can encode the ids m and s as a string. By operating this method, the researchers can locate the index in the feature vector idx = hash(s) % 2**B and increment the count at that index(Nowling Geromel, 2016) [8]. The researchers later used the L2(Ridge) penalty and log loss to reduce the risk of overfitting. Evaluated the accuracy, the researchers finally achieved an AUC with 96% at 2 to the 28 hashed features.

This work is similar to part of our project since we will also conduct the logistic regression models for building movie recommendation systems. Differently, we are going to mainly analyze the movies watched by the same user rather than creating an interaction matrix. Moreover, the penalty method we picked for logistic regression is L1(Lasso). Similarly, the binary output will still be 0, and 1 represented not recommend and recommend.

2.2. Content-Based Recommendation System

In this project, we are using content-based recommendation system to predict possible interests of users according to their previous rating history. Content-based recommendation system has been popular among some websites like Facebook and YouTube[12]. This algorithm is based on the user preferences for items and it would recommend items that are categorized in the similar genres or domains[12]. For example, a paper published by Szomszor et al. proposed a semantic web to predict user interest in different types of movie [10]. As shown in figure 3.2.1[9] below, the recommender would recommend items that share similar characteristics based on user preference.

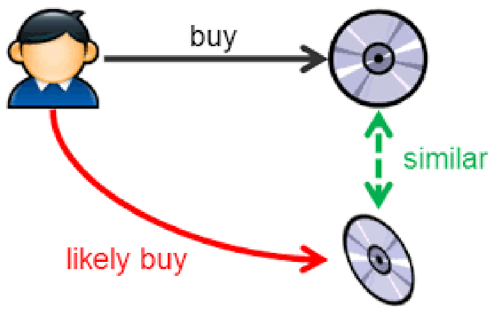

Figure 3.2.1 Recommendation System Picture

Moreover, Subramaniyaswamy and Logesh[15] introduced a method to enhance recommendation accuracy based on user contexts. To classify users and items based on those data, an adaptive KNN algorithm is implemented. Instead of starting with users with adequate information to build models, Subramaniyaswamy and Logesh[15] started their work with new users without sufficient user information and rating information. However, due to the fact of insufficient rating information, they were not able to generate recommendations based on ratings, but on user information that was collected when the user registered for an account. In the second part of the paper, Subramaniyaswamy and Logesh[15] decided to generate recommendations based on ratings for old users with long enough history.

In this report, we present our recommendation system based on content-based algorithm without using user information. Even though Subramaniyaswamy and Logesh's[15] method with new users could handle the situation without too much rating data exposed to the recommender, we did not build models with user information as we tried to avoid relying on personal information as the concerns of user privacy as we have mentioned. Instead, we build models based on each user's rating history, which is similar to the second approach in their paper. The following sections describe our proposed methodology and experiment processes to build models in detail.

3. Proposed Method

To find out the model with the "best performance" in the movie recommendation system, we divide the experiment into two parts. First, we build multiple machine learning models based on each user with corresponding movie ratings and other movie information. Then, select the model for the specific user by model accuracy and train the final model with the selected model. The machine models we build in this project are listed below.

3.1. Lasso Logistic Regression

The logistic regression is used in order to predict the probability of a class or an event will happen or not. The dependent variables are 0 and 1 or two level categories. The output of the logistic regression model is the probability which indicates that how possible of an event will happen. If the probability of an event is higher than 0.5, we can say that the result will be 1, and 0 when the probability is under 0.5. In logistic model, the log-odds of an event is a linear function with one or several independent variables that could be category or continuous, which can be shown in the coefficients. The log odds is converted by the Sigmoid function, which is

$$h_\theta(x) = \frac{1}{1 + e^{-}\theta^T x}$$

. The formula for logistic regression and Sigmoid function is shown in Figure 3.1.1[14].

Regression formula.png

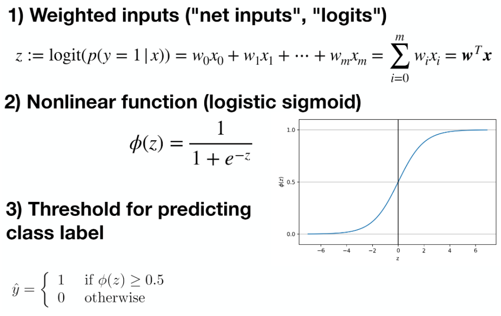
1) Weighted inputs ("net inputs", "logits")
$z := \text{logit}(p(y=1\mid x)) = w_0x_0 + w_1x_1 + \cdots + w_mx_m = \sum_{i=0}^{m} w_ix_i = \mathbf{w}^T\mathbf{x}$
2) Nonlinear function (logistic sigmoid)
$\phi(z) = \frac{1}{1+e^{-z}}$
3) Threshold for predicting class label
$\hat{y} = \begin{cases} 1 & \text{if } \phi(z) \geq 0.5 \\ 0 & \text{otherwise} \end{cases}$

Figure 3.1.1 Logistic Regression Model Sigmoid Function

The lasso penalty is a method to conduct feature selection, increase model predicting accuracy, and avoid the risk of overfitting. Since there is a bias and variance tradeoff which will cause the errors in prediction, Lasso is working based on penalized the absolute values of the coefficient, which will lead some coefficients to zero. Therefore, it is crucial for us to perform Lasso penalty on our logistic regression model. The Figure 3.1.2[13] shows the procedure of a logistic regression with lasso penalty.

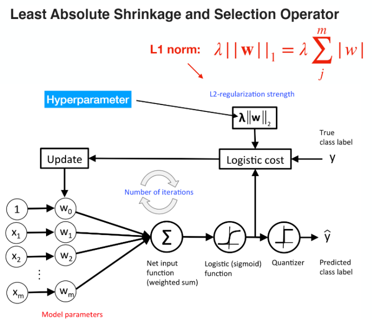
Least Absolute Shrinkage and Selection Operator
L1 norm: $\lambda||\mathbf{w}||_1 = \lambda\sum_j^m |w|$
L2-regularization strength
Hyperparameter
$\lambda||\mathbf{w}||_2$
True class label
Update
Logistic cost
y
Number of iterations
Net input function (weighted sum)
Logistic (sigmoid) function
Quantizer
Predicted class label
Model parameters

Figure 3.1.2 Logistic Lasso Regression

3.2. Decision Tree Classification and Regression

Decision Tree is a method that uses tree-like graph or combinations of decision rules to predict possible consequences. In other words, decision tree algorithm predict the classified labels by implementing several combinations of conditional statements. Rajesh BridIn(2018), in his teaching session, described that within a decision tree, each internal node was a test on an attribute, each branch represented the outcome of the test, and each leaf node represented a class label, and the classification rules are represented by the path from root to leaf. [5] The reason of why we used decision tree algorithm in our project is that it is one of the most popular unsupervised learning method. It is powerful on predicting the outcome with high accuracy.

3.3. KNN

The KNN is a lazy and supervised machine learning algorithm. [3] It does not train data, but stores all training data and finds the nearest k data point stored before around the query data, and conducts a ”plurality” vote to classify the query point.

3.4. Gradient Boosting

Gradient boosting is a machine learning method that combines weak models into an ensemble of strong prediction model in an iterative process.[4] This technique can be applied for both classification and regression problems. In this project, we use gradient boosting algorithm and expect it might provide us a well-performed model because all weak models contribute to the strong model.

3.5. Random Forest

After considering the noticeable difference between the training set accuracy and test set accuracy of decision tree classifiers, we implement random forest classifier to reduce further overfitting. The training process for random forest applies bootstrap to each tree classifier[4]. Suppose for a data frame with n training samples and k feature columns. For each tree, by repeatedly bootstrapping a subset of m training samples (m much smaller than n) with replacement, we have $y_1, y_2, \ldots, y_m$. By repeatedly bootstrapping a subset of t features (t much smaller than k) with replacement, we have $x_1, x_2, \ldots, x_t$ variables. After fitting each tree classifier with a subset of training samples with a subset of variables, a classification (prediction for regression mode) ŷ will be made. Suppose bootstrapping B times, the final classification will be determined with majority voting over $\hat{y}_1, \ldots, \hat{y}_B$ (averaging $\hat{y}_1, \ldots, \hat{y}_B$ for regression mode). In

[3]`https://github.com/rasbt/stat479-machine-learning-fs19/blob/master/02_knn/02-knn__notes.pdf`

[4]`https://en.wikipedia.org/wiki/Gradient_boosting`

this project we apply binary classification mode. By doing this, random forest algorithm adjusts for decision trees' tendency of overfitting to their training set.

3.6. Clustering

K-Means Clustering Despite regular supervised learning methods, we also explore relevant unsupervised learning algorithms. K-Means Clustering is one of the most straightforward clustering methods that partitions data points into K pre-determined groups[1]. K-means clustering tries to assign data points into groups as far from each other as possible by minimizing the sum of squared distance between each point and its cluster center J (this sum is also referred to as distortion).

$$J = \sum_{i=1}^{m} \sum_{k=1}^{K} w_{ik} \|x^i - \mu_k\|^2$$

Figure 3.6.1 Cluster Center Funcation

where wik=1 for data point xi if it belongs to cluster k; otherwise, wik=0. Also, k is the center of xi's cluster[6]. Data points that belong to the same cluster should be very similar to each other and thus should have similar classification.

Mean Shift Clustering The mean shift method is a non-parametric clustering algorithm which does not require knowledge of the optimal number of clusters in advance, and does not constrain the shape of the clusters[6]. Mean shift is a procedure for finding the maxima of a density function given data sampled from that function[3]. Gaussian kernel function is the most commonly used.

$$K(x_i - x) = e^{-c\|x_i - x\|^2}$$

Figure 3.6.2 Gaussian Kernel Function

The density function determines the weight of nearby points for re-estimation of the mean[3]. The procedure starts with an initial estimate x. The weighted mean of the density determined by K is

$$m(x) = \frac{\sum_{x_i \in N(x)} K(x_i - x) x_i}{\sum_{x_i \in N(x)} K(x_i - x)}.$$

Figure 3.6.3 Weighted Mean Density Function

where N(x) is the neighborhood of x, a set of points for which K(xi) 0[3]. The difference m(x) - x is called mean shift[7]. This method continuously shifts x to m(x) until m(x) converges.

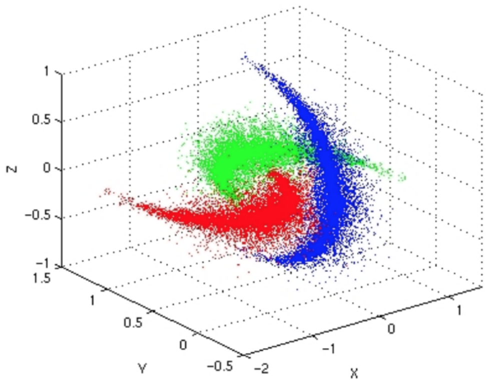

Figure 3.6.4 Means Shift Demonstration

DBSCAN DBSCAN groups points that are close to each other based on their distance (usually Euclidean distance) and a minimum number of points[2]. It also marks points that are far from the dense region as outliers (labeled as -1 in scikit learn). On top of the previous techniques, this feature helps focus on main clusters and allows a few irrelevant data points.

4. Experiments

4.1. Dataset

Dataframe overview and split data In this project, our raw data come from The Movie Database (TMDB) 5000 Movie Dataset `https://www.kaggle.com/tmdb/tmdb-movie-metadata` redistributed on Kaggle and MovieLens Datasets compiled by Harper and Konstan `https://grouplens.org/datasets/movielens/latest/`. To best utilize those data sets, we combine the movie information from TMDB and the rating data from MovieLens based on movie names. The final data set imported in our project contains 3065 movie features (including 2965 dummy variables indicating a specific genre of the movie) and 32080 movie ratings from 101 distinct users who rated more than 150 movies in our data, because generating recommendations without adequate ratings can be really tough, as we have discussed in the previous sections. For each model, we use the movies user 414 watched as example for early model constructing and tuning, as user 414 has watched the most (1281 movies). After that, all models are evaluated on each of the 101 users separately which gives 101 accuracies. At the end we average these accuracies as the final evaluation of each model. For features, we delete unuseable ones such as review text and time stamp. Since each movie might have multiple genres, production companies, and countries, we split each of the factor into a dummy variable, indicating it is one of the movies' features or no. For samples, we randomly split movies for each user into 80% training and 20% testing with stratification.

4.2. Procedure

Logistic Lasso Regression For conducting the logistic lasso regression, we first change the rating into 0 and 1. If the rating is higher than 3.75, denoted as 1, if the rating is lower 3.75, denoted as 0. Import datasets and linear_model from sklearn, and import train_test_split from sklearn.model_selection build a function with the input variable user's id. Inside the function, selected the movies that the inputted user id that has watched. Then split the data into training and test dataset with the ratio of 4:1 — next, import LogisticRegression from sklearn.linear_model and import accuracy_score from sklearn.metrics. Then create a function to build logistic lasso regression. Inside the function, fit the logistic regression with the parameters inside: penalty='l1', solver='liblinear'. Predict the result in for training data and calculate the accuracy. For checking the performance of logistic lasso regression, draw the confusion matrix and ROC curve with confusion matrix by using confusion_matrix and metrics from sklearn. By using for loops to go through each user Id, we can gain the accuracy of logistic lasso regression model on every user.

Decision Tree Classification and Regression Steps in decision tree classification and regression were similar and simple. In general, we created a function that fitted decision tree model for one specific user to get one movie rating accuracy. And then applied this function to all users to get an overall accuracy of this decision tree model. First step was to apply 'train test split' function to split the train and test sets. The ratio of splitting train and test size is 7:3. Then we applied grid search to find parameters, including max depth and criterion (gini and entropy), that could return highest accuracy. This step was only conducted in the decision tree classification. And then we fit the models by using train data points as the movie information and the movie ratings given by the users. Later we used the model to predict movie ratings. The last step is to compare the predicted movie ratings and the actual ratings and then computed the accuracy rate. Details about the result of our decision tree classification and regression models will be discussed in the following session.

Random Forest We apply classification approach for random forest in this project. So before fitting model, the "rating" was manually labeled as 0 and 1, where 0 means "not like" for training set, "not recommend" for testing outcome and 1 means "like" and "recommend". All movies rated equal to or higher than 4 are converted to label 1 and all movies rated lower than 4 are converted to label 0. Then we further split the training set into training and validation set where the hyperparameters "max_depth", "n_estimators", and "criterion" are tuned on. After finding the best model setting, we perform feature selection by removing the ones with extremely lowest importance (¡0.001). After that we fit the same model with selected features and evaluate it with 10-fold cross validation and test set accuracy.

Gradient Boosting To implement gradient boosting ensemble method, we label our dataset with 0 (ratings lower than 4.0) and 1 (ratings greater or equal to 4.0). The training and tesing data set ratio is also 7:3. To tune hyperparameters in gradient boosting, we use GridSearchCV from sklearn package in Python with 10-fold cross validation. For tuning the models for each user, we set learning rate as 0.01, 0.025, 0.05, 0.075, 0.1, 0.15, 0.2, 0.25, 0.5, 0.75 and 1. Then, we calculate the mean model accuracy and compare with the other models to find the one with the highest model accuracy.

K-means Clustering Training set is standardized before model fitting. Then we try to find the optimal number of clusters by looking at the distortion for each setting. Then, we split training data into this number of clusters and predict the rating for each movie with the average rating of the cluster it belongs to. For clusters with average rating higher than 3.75, the movies in them will be predicted as "recommend". The rest will be "not recommend". Then we obtain the accuracy by calculating the probability of misclassification.

Mean shift Clustering To avoid using a fixed number of cluster, mean shift is introduced. Training set is standardized before model fitting. Then we estimate the bandwidth with "estimated_bandwidth" function in sklearn.cluster. This is used as the only tuned hyperparameter. After fitting the model, similar to K-Means, all movies predicted with higher than 3.75 will be labeled as "recommend" and vice versa. Accuracy is obtained by calculating the probability of misclassification.

DBSCAN Clustering To further consider non-linear pattern and weakening the noise from outlier, DBSCAN is introduced. Training set is standardized before model fitting. The following procedure is identical to mean shift approach. In addition, confusion matrix is also shown.

K-Nearest Neighborhood To classify movies with kNN model, we first label dataset with 0 and 1 based on user ratings, which is the same as the previous algorithms mentioned above. The ratio of training and testing data set is

7:3. We implement grid search with 10-fold cross validation to tune the hyperparameter for each user. For the first stage of the experiment, we build kNN models for each user separately based on his or hers own rating history, we do not present the k values for all users here.

Final Model with K-Nearest Neighborhood After calculating all model accuracies we build in the project based on the experiment presented above, we find that kNN has the highest mean model accuracy, so we regard kNN has the best performance among all the models used in this project. Then, rather than train separate models for each user, we build and test the final model with kNN algorithm with the data set containing user ratings to predict whether this movie should be recommended. Similar as the previous kNN model building procedure, we use GridSearchCV from sklearn to tune the final model. We figure out when k=12, the model has the highest accuracy of 96.76% with all user rating data.

4.3. Software

Python 3.7
numpy 1.16.4
sklearn 0.21.2
Jupyter Notebook
Google Colab
The source code is available at our GitHub repository[5]

4.4. Hardware

Two Macbooks and two Dell laptops (8th Gen i3/8GB RAM/512GB SSD)

5. Results and Discussion

The average accuracy table is showed below:

Table 1. Average Accuracy Table

Model	Average Accuracy
KNN	90.14%
Gradient Boosting	73.12%
Logistic Lasso Regression	69.44%
Random Forest	68.88%
Decision Tree Regression	62.19%
k-Mean cluster	51.34%
Mean Shift cluster	55.32%
DBSCAN	53.57%

[5]`https://github.com/LTCrazy/Stat479_Movie_Recommender-`

5.1. Decision Tree Classification Regression

Decision Tree Regression The depth we used in decision tree regression was three and the criterion is 'mse', the average accuracy of decision tree regression for all users is 62.19 %. Here we used one user (Id = 414) to illustrate the tree plot of model, the figure is attached below:

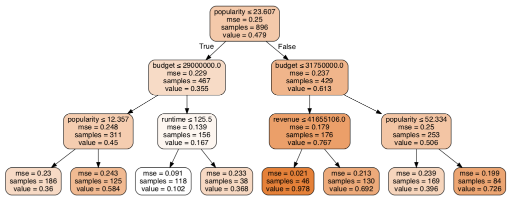

Figure 5.1.1 Tree Plot of Decision Tree Regression

In the tree plot of decision tree regression, the value close to 1 means recommend (the user is likely to have a rating greater or equal to 4), whereas the value close to 0 means don't recommend (likely to have a rating ¡4). For example, if a movie that has a popularity greater than 23 and the movie's budget smaller than or equal to 31.75 million dollars and if it has a revenue smaller than or equal to 41.66 million dollars, the user will give the movie a rating as recommend.

Decision Tree Classification The best parameters in decision tree classification are depth with five and criterion with 'gini', the average accuracy of decision tree classification for all users is 62.08 %. This average accuracy is pretty close to the average accuracy of decision tree regression. For consistency, here we also used user414 to show the tree plot. Since the plot with a depth five is too large to make interpretation, here we don't discuss the detailed interpretations of decision tree classification. The interpretation for each node is similar as it in the decision tree regression.

5.2. Logistic Lasso Regression

The average accuracy for logistic lasso regression of each individual user is 69.44%. In order to have better understanding, we performed a detailed performance analysis on user 414. The accuracy of logistic lasso regression on user 414 is 66.15 %. The precision is 0.68 and the recall is 0.54. The confusion matrix is below in Figure 5.2.1 and ROC curve in Figure 5.2.2.

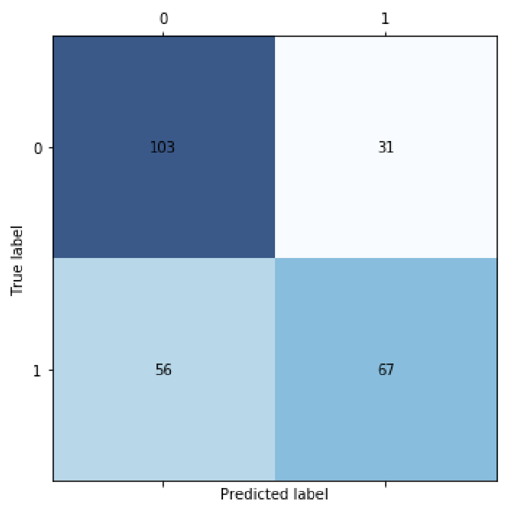

Figure 5.2.1 Confusion Matrix

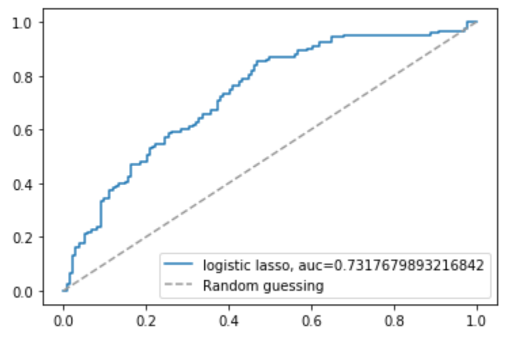

Figure 5.2.2 ROC Curve
Based on the ROC curve, we can see that the performance of logistic regression is better than random guessing and is nearly in the middle of random guessing and ideal situation.

5.3. Random Forest

The optimal "max_depth" and "n_estimators" for most users falls within range of 5 - 15 and 120 - 180. These hyperparameter are considered in our finalized model tuning. More informative features such as budgeting and year from now turn out to be more important than most of the dummy variables for production countries and companies. 32 out of 3000 features are used in the final model. Our final average accuracy among all users reaches 68.88% which is one of the most accurate methods in our project. Given the previous experience of decision tree method, average rating from public seems to dominate the prediction over all other feature which is eliminated in our final model.

Comparing to decision tree approach, random forest classifier further boosts the average accuracy. It also successfully reduces training set overfitting by reducing the gap between training accuracy and test accuracy (mostly within 5%). Feature selection by importance also further boosts the accuracy by 5 - 10%. However, for a few samples, accuracy is lower after feature selection. This might result from the small sample size for each users or the overfitting from fitting a model on the same samples twice.

5.4. Clustering

K-means Clustering We did not detect an obvious "elbow" for the optimal number of clusters for user 414:

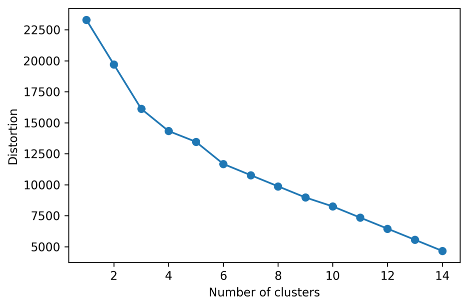

Figure 5.4.1 Scree Plot
As 4 is more likely here, we fit k-means model with 4 clusters. The accuracy only goes to 51.34%. Given the suspected distortion graph, we doubt that setting 4 clusters might not be an optimal decision. Then we explored mean shift.

Mean Shift Clustering This approach sufficiently increases the number of clusters. For instance, it splits the movie watched by user 414 into 25 clusters and reaches an accuracy of 55.91%. However, by looking at the distribution of movies in the clusters, we realize that most clusters only contain 1 or 2 movies whereas some clusters contain almost all the movies in the set. This model also has a huge gap of false positive rate between training and testing (about 60% for user 414). We conclude that this method tends to overfit on training set noise, which leads to our attempt on DBSCAN method.

DBSCAN This model splits over 800 movies into 20 clusters and gives us an accuracy of 53.57% for user 414. This accuracy seems lower than mean shift. However, this model gives us a surprisingly low false positive rate (4.55%). Here is the confusion matrix:

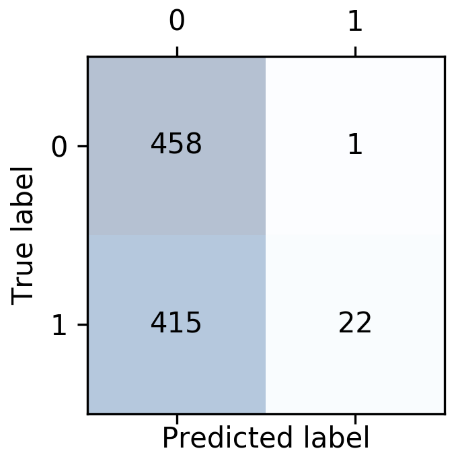

Figure 5.4.2 Confusion Matrix for DBSCAN

A low false positive rate means there is a high chance that the movies our model recommends will actually be liked by users. The downside of that is this model only recommends a small number of movies, 23 out of 800 movies for user 414 for example.

In conclusion, clustering does not perform as well as some other methods in the project. The sample size is not sufficient enough for models to capture the true clusters compared to the big number of features. A more sophisticated/delicate classifier is needed.

5.5. Shortcomings and Limitations

There are several limitations to our movie recommendation project. First, the sample size is too small in terms of creating an efficient movie recommendation system. Due to the shortage of memory, we are not able to analyze the large data set with more observations and features. Secondly, there is a class imbalance after we converted the users' ratings into 0 and 1. There are more 0s than 1s, which will cause our model biased. Thirdly, due to the hardship in data cleaning, we gave up on some features that we could not analyze. For instance, the text description could be an essential feature in our model, but we have limited skills in analyzing these descriptions.

6. Conclusions

Movie recommendation systems have been widely used to optimize resource distribution. However, concerns are raised on the usage of personal data and the limited access for some third-party websites. A recommendation systems with little usage of user information. In this project, we implement decision tree, random forest, clustering, lasso logistic regression, *k*Nearest Neighborhood and gradient boosting algorithms to build models with movie features and user ratings. Unlike previous works that based on study similarity between viewers such as the first research in related work section, our project focused on the taste of movie of a single user. We are able to build machine learning models based on the users' rating and movie features, rather than user personal information, which meets our primary motivation for doing this project.

We first implement machine learning algorithms on data sets that contains only one user's rating data and iterate the same procedure of all users. However, as we discuss above, clustering methods do not perform well due to the overwhelming feature size compared to sample size. We find that KNN has the highest mean accuracy(90.14%). Then, we build a final KNN model that can be generally used to classify whether a movie should be recommended by the system with the data set contains movie ratings from all users.

In the future, we may want to use GPU instead of our own computers to modeling on a significantly larger data set. It is also necessary to use more sophisticated methods for analyzing unbalanced rating labels (1 far fewer than 0). We might also consider to normalize each user's rating and customize a "liked" threshold for each one. Other important features should also be considered in order to further increase the accuracy. We will try to conduct techniques of text sentimental analysis such as Bayes Net to improve our recommendation models.

Our project is practical and might be an approach that recommendation systems would develop in the future. It can be used by websites to make recommendations with a better privacy protection to users, let along some third-party websites without access to user information.

7. Acknowledgements

We would like to show our gratitude to our great professor, Sebastian Raschka for providing us instruction in machine learning and offering insights to complete the final project. We also want to thank those authors who developed scikit-learn package, so that we could implement our methods easily in Python.

8. Contributions

Everyone in our group contributes evenly in this project. We work on data cleaning process together and each one implements several methods for the experiment part.

Tianchang Li contributes to decision tree classification, random forest, and clustering algorithms implementation. Shixuan Song builds models with KNN and gradient boosting. She is also in charge of modifying KNN as our finalized model. Maoze Wang mainly works on presentation, code with logistic regression and LASSO regression, and drafting the final report in Latex. Xuchen Xue builds decision tree regression model and helps with Latex editing.

References

[1] Towards data science.

[2] Towards data science.

[3] Wikipedia for mean shift.

[4] Wikipedia for random forest.

[5] R. S. Brid. Decision trees a simple way to visualize a decision, 2018.

[6] M. S. D. Comaniciu and P. Meer. Mean shift: A robust approach toward feature space analysis. 2002.

[7] K. L. D. H. Fukunaga. [the estimation of the gradient of a density function, with applications in pattern recognition. 1975.

[8] R. N. . R. R. Geromel. Recommendation system using logistic regression and the hashing trick, in nowling lab blog. *Nowling Lab Blog*, 2016.

[9] M. M. [figure]learning how recommendation system recommends. *Medium: Towards Data Science*, 2018.

[10] H. A. K. O. A. B. V. L. M. Szomszor, C. Cattuto and V. Servedio. the semantic web, and movie recommendation. *Bridging the Gap between Semantic Web and Web 2.0*, 2007.

[11] B. Rocca. Introduction to recommendation system,toward data science. 2019.

[12] D. S. Beginners guide to learn about content based recommender engines. *Nowling Lab Blog*, 2015.

[13] sebastian raschka. L1 regularization/lasso(emedded). 2019.

[14] sebastian raschka. Logistic regression. 2019.

[15] V. Subramaniyaswamy and R. Logesh. tadaptive knn based recommender system through mining of user preferences. *Wireless Pers Commun 97*, 2017.

[16] Vikash. Netflix inc, competitive position and analysis, medium. 2019.