Movie Recommendation System

Andrew Fix afix@wisc.edu
Chaiyeen Oh coh26@wisc.edu
Samuel Peters sjpeters3@wisc.edu

Abstract

Movies are one of the best forms of entertainment, but they are much more enjoyable when you find ones you like. This leaves the issue of spending too much time searching through a seemingly limitless library of movies on streaming services without much guidance on which movies you will like. To equip users with a better system of finding movies, this report uses machine learning algorithms aimed to develop predictions on what movies a given user will enjoy. The algorithms used were K-Nearest Neighbor, Decision Tree, Random Forest, and XG Boost, each one giving a unique interpretation on the problem at hand. The data provided user and movie data which included features such as genre and rating. To improve performance of the models and have a more intuitive approach to the data set, the movie ratings were changed from a numerical metric into a binary metric and movies with fewer than 25 ratings were removed. While each model exceeded the initial benchmark metric, XG Boost and Random Forest models produced the highest accuracy when used on test data. While there are still more aspects of this dilemma that can be explored, the models described in this report give users an effective way to find movies that they will enjoy.

1. Introduction

Before the invention of the internet, people had to buy newspapers, books, and magazines to get informed and be entertained. Nowadays, we just need to go to our laptops, type in a word and get thousands of results.Although information was never easier to access than it is today, people often get lost in an endless sea of information and spend more time searching rather than actual reading.This information overload not only happens in research, but also in the internet television industry.

Major internet television companies make their revenue through users' subscriptions, which allow them to watch an unlimited number of movies and TV shows on their platform. However, many users find it difficult to choose between too many movies to watch. According to Netflix customer research, "a typical Netflix member loses interest after perhaps 60 to 90 seconds of choosing [movies] ..."[2]. To lower risk of un-subscription, Netflix applied a personalized recommendation system on their platform. And as a result, the recommendation system influenced 80% of hours of streamed and maximized the company's revenue [2].

2. Related Works

Like Netflix's recommendation system, our goal for this project is to train one or more machine learning models that learn from the past movie preferences of a user and predict their rating of new movies. And as a reference, we will be looking at related works as described below.

2.1. K-Means Clustering

"Movie Recommendation With K-Means Clustering and Self-Organizing Map Methods" details how a K-means clustering algorithm can be used to generate a recommendation system for movies [7]. As shown in Figure 1, users can be grouped using K-means clustering according to how similar their ratings of movies are. This collaborative filtering method allows us to differentiate user groups. The recommendation system can then provide different recommendations for each group based upon its features. For example, if a new user were to rate a movie, they would be assigned to the closest user group. Then, they would be recommended the movies with the highest average rating among that group.

2.2. Genre Correlation

In 'An algorithm for movie classification and recommendation using genre correlation', authors were using interesting way of pre-processing data. They tried to calculate correlation using Pearson correlation coefficient to find similarities between movie genres and then made preference prediction model to predict the ratings. Mathematical formulas are shown in Figure 2 [4].

3. Proposed Method

3.1. Evaluation

Due to the nature of our data, we are unable to simply predict one variable for each data point. Rather, we must

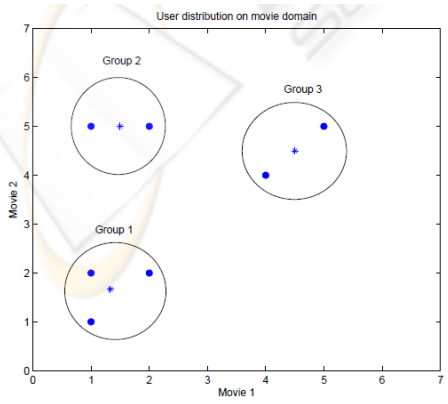

Figure 1. Users plotted based upon their ratings for 2 movies. A K-means algorithm groups the users into 3 distinct groups.

Item similarity computation

$$sim(i,j) = \frac{\sum_{u \in U_i \bigcap U_j} \left(r_{u,i} - \overline{r}_i\right)\left(r_{u,j} - \overline{r}_j\right)}{\sqrt{\sum_{u \in U_i \bigcap U_j} \left(r_{u,i} - \overline{r}_i\right)^2} \sqrt{\sum_{u \in U_i \bigcap U_j} \left(r_{u,j} - \overline{r}_j\right)^2}}$$

Preference prediction

$$P_{u,i} = \frac{\sum_{j \in I_u} sim(i,j) \times r_{u,j}}{\sum_{j \in I_u} |sim(i,j)|}$$

Figure 2. Iu and Iv are items rated by users u and v. And Nu contains K nearest neighbors of user u

remove a certain number of examples for each user in the validation and test sets, and use those ratings to predict. After splitting the data into an 80%-20% training-testing split, we further split the training data into another 80%-20% training-validation split. When splitting, we split along users to ensure that all of a user's ratings were exclusively in one data set. From there, we followed the example of the Netflix Prize [5], where we removed 10 ratings from each user in the validation and test sets to use for prediction. We decided to remove the most recent 10 ratings for such users instead of 10 random ones, as a recommendation system in practice would be making predictions about movies a user has yet to watch. Next, we combine the non-removed ratings of the validation and test sets individually with the training set. Since users are exclusively in the training, validation, or test sets, this assures that there is training data for each user that we predict.

Ratings, which we are trying to predict, are on a 0.5 star - 5.0 star scale in half-star increments. Rather than try to predict one of these 10 classes, or use regression, we decided to reduce this data into two classes: Highly Rated and Lowly Rated. We decided to classify all ratings of 3.0 stars and more as Highly Rated, and the rest as Lowly Rated. Then, using these two classes, we will use classification accuracy to assess the efficacy of our models.

3.2. KNN

KNN, short for K-Nearest Neighbors, is a supervised machine learning algorithm that can be used to predict the labels of a query point. The algorithm's philosophy is based on the idea that each query point should be labeled based on the labels of points close to it. A predetermined value, K, determines how many nearby data points will be considered in making each label. The label can be chosen in two different ways: majority voting or plurality voting. When making binary predictions, majority voting is used predicting the label based on what more than half of the K nearby points are labeled as. In multi-label cases, plurality voting is used to classify the query point based on which nearby label is most frequently used in the K nearest points.

KNN is a useful form of supervised learning because of how easy the concept is. K is the only hyper parameter used and is more multi-class friendly than most other classifier algorithms. One downside to using KNN is how long it takes to run the algorithm. Since it takes a memory-based approach on labeling data, it takes time to make each classification. Another problem with the KNN algorithm is determining which value of K works best with the data. This can sometimes be a challenge and greatly impacts how points are labeled.

3.3. Decision Tree

Decision Tree is one of the supervised machine learning methods used for classification and regression. It usually creates a treelike shaped map of the possible outcomes from different combination of choices. And its goal is to make predictions from how a previous set of questions were answered by greedily searching a tree [1]. For instance, for N binary features, there can be at most two to N power splits of tree.

Many developers use decision tree model because it is easy to understand and interpret from the tree visualization, runs in logarithmic time complexity in predicting the data, and does not require data normalization, dummy variables, or blank values to be removed. However, it also has a downside. The disadvantages of decision tree are that it can generate completely different tree from small variations in the data, predictions are not smooth nor continuous, and there are concepts in decision trees that are hard to learn such as XOR expression [6]. Because decision tree model is easy to understand from tree shaped diagram, it was one of the go-to models to start the project.

3.4. Random Forest

Random Forest is a supervised learning algorithm featuring a form of bagging on decision trees. The idea is that

a bagging method is used on decision trees, however, the feature subsets are generated randomly. Bootstrap samples are pulled for each decision tree and the features to be used are randomized.

Due to this randomization of features, there is much less tuning that needs to be done on the hyper parameters which is one reason why random forest is attractive. The random nature does provide some downsides because you have little control over what the model is actually doing.

3.5. XGBoost

XGBoost is a gradient boosting ensemble method that attempts to combine numerous weak learners into a strong learner. XGBoost and other gradient boosting algorithms are sequential and as a result are often expensive to train.

Gradient boosting optimizes a differentiable loss function of a weak learner through consecutive rounds of boosting. The result is an additive model of multiple weak learners, as opposed to majority voting which is often used in other ensemble methods.

XGBoost is a gradient boosting ensemble of decision trees where individual trees can be fit on a subset of the data and new trees minimize the errors of the trees already in the model. Put simply, XGBoost works by constructing a base tree, building a subsequent tree based upon the errors of the previous tree, combining the two trees, and then repeating.

4. Experiments

4.1. Dataset

We retrieved our data from GroupLens, a research group in the Department of Computer Science and Engineering at the University of Minnesota. GroupLens Research operates a movie recommender based on collaborative filtering, MovieLens, which is the source of these data [3]. We used two dataset, one which has userId, movieId, rating, and timestamp, and another which has movieId, and genres. User Ids are randomly selected MovieLens users with anonymized Ids, included users have at least 20 ratings. Movie Ids are movies from MovieLens with at least one rating. Ratings are on a 0.5 star - 5.0 star scale in half-star increments. Timestamps represents seconds since midnight Coordinated Universal Time (UTC) of January 1, 1970.

4.2. Data Pre-processing

4.2.1 Filtering

Figure 3 shows that many movies have very few ratings. Since movies in this dataset only need to have at least one rating, there are many with very few ratings. In fact, the 25^{th} percentile and median are 1 and 3 ratings, respectively. Meanwhile, the highest rated movie has 329 ratings, which is more than half of the 610 users.

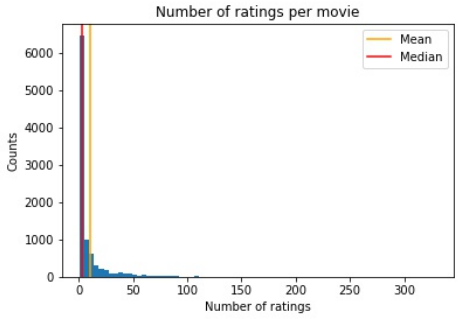

Figure 3. Histogram of the number of ratings each movie has.

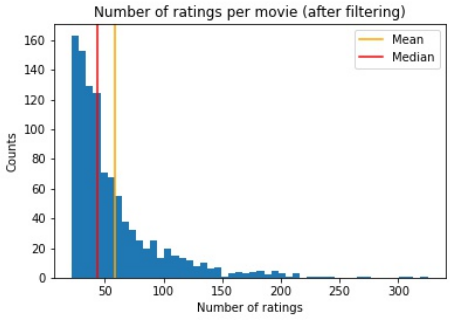

Figure 4. Histogram of the number of ratings each movie has after 8674 movies with fewer than 25 ratings have been removed.

Movies with very few ratings will be hard to predict since they won't have many training examples available. Furthermore, said movies are likely not very popular due to their low number of ratings, and are likely not to be highly recommended in the first place. For these reasons, we decided to remove all movies with fewer than 25 ratings. Figure 4 shows the new distribution of movie ratings after the 8674 movies with fewer than 25 ratings were removed. In order to maintain the 20 ratings per user, we also had to remove 61 users.

4.2.2 Generating Rating Averages

While trying to predict the rating a user will give to a movie, it is important to remember that some movies will tend to be rated higher than others, regardless of the user rating it. Furthermore, some users will tend to either be more critical or lax in their ratings, regardless of the movie they are reviewing. In order to best capture this element, we decided to encode each user's average rating as well as each movie's average rating. This should allow a better baseline for each user's rating of a movie, and are in fact the metrics we use to generate our baselines.

As shown in figure 5, users' average ratings appear to follow a uni-modal normal distribution, with a few users

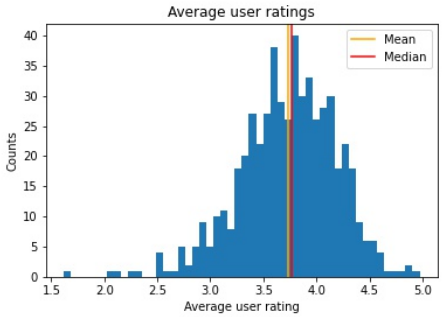

Figure 5. Histogram of movies' average ratings.

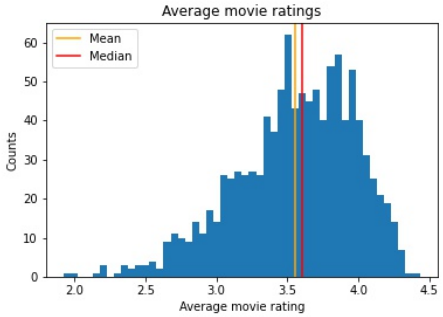

Figure 6. Histogram of users' average ratings.

who are either very critical or lax in their ratings. Figure 6 shows us that the distribution of movies' average ratings is left-skewed.

4.2.3 Generating Rating Counts

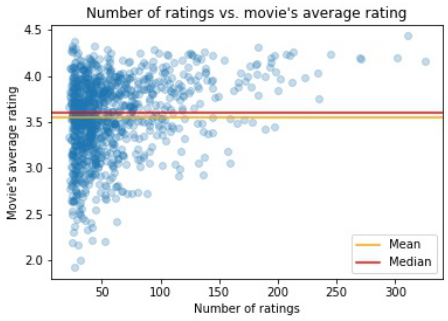

Figure 7. Scatter plot of a movie's average rating against its number of ratings.

Upon exploring the relationship between the number of ratings a movie has and its average rating, as shown in figure 7, we noticed that as the number of ratings a movie has increases the likelihood that it has a low average decreases. This discovery led us to include the number of ratings a movie has received in our dataset.

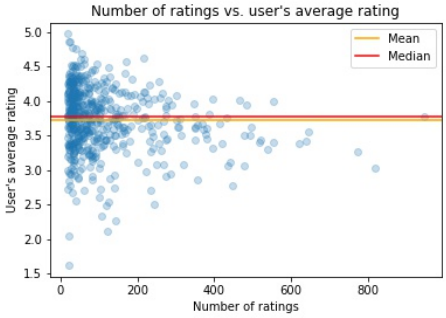

Figure 8. Scatter plot of a user's average rating against their number of ratings.

While the number of ratings a user has submitted is not as clearly related to their average rating, figure 8 suggests that the more ratings a user has, the more likely they are to be closer the the mean (which they more heavily influence), and be farther away from the extremes of rating too critically or forgivingly.

4.3. KNN

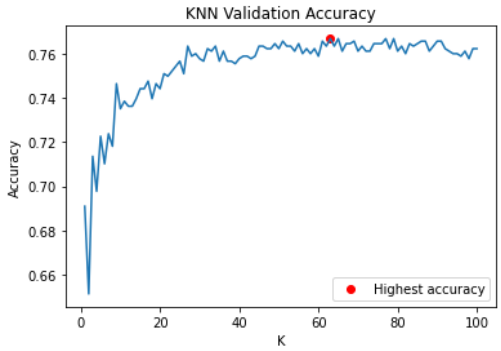

Figure 9. Validation accuracy for different values of K.

Considering that the only hyper parameter to tune in a K-Nearest Neighbors algorithm is K, the goal was to optimize this value and determine how many neighboring values should be considered in labeling query points. To find the best value for K, a K-Nearest Neighbors algorithm was repeatedly ran using different K values with the intentions of finding which K gives the best training accuracy. After trying K values from 1 to 100, the value with the best training accuracy was 77.

Higher K values generally make for smoother decision boundaries and a more refined idea of how to classify these query points. If the K value is too small, the model fails to generalize well and can be too sensitive to noisy data.

It makes sense that the K value is relatively high in this model. Looking at figure 8, we can see that as the value of K increases, so does validation accuracy.

It is worth mentioning that KNN is the only model where we did some form of feature selection. For our KNN model, we received very poor performance the more features we used. Through trial and error, and by looking at what features were important in the other models, we decided to only use the user and movie average features.

4.4. Decision Tree

To find the best parameters of decision tree, we used Decision Tree Classifier and GridSearchCV from sklearn package. Decision Tree Classifier method has multiple options for each parameter that we were able to use. For instance, criterion parameter measures the quality of a split either as Gini impurity which is a variance across the different classes or entropy which is a measure of chaos within the node for the information gain. We were also able to define maximum depth of the tree which expand until all leaves are pure as a default setting [6]. Setting maximum depth of the tree is important because it helps to prevent over-fitting.

Since there's so many combinations between multiple parameters to create individual prediction model for, we utilized GridSearchCV method which tries all and picks the best parameters out of the given options. For our model, we compared based on accuracy between different combinations of tree with 11 different maximum depth and two criterion. And as a result, we were able to find the best tree model which had gini entropy and maximum depth of four with 76.02% of validation accuracy.

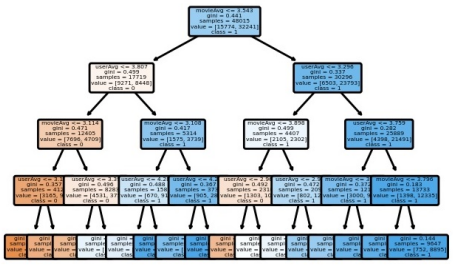

Figure 10. Above figure shows generated Decision Tree diagram

4.5. Random Forest

For our Random Forest ensemble, we used GridSearchCV on the two parameters n_estimators and criterion. For n_estimators we used [10, 20, 50, 100, 200, 300, 400, 500, 1000] and for criterion we tried "gini" and "entropy". We found that the best parameters we obtained were "entropy" for criterion and 500 for n_estimators.

Model	Validation Accuracy	Test Accuracy
XGBoost	78.07 %	77.91 %
Random Forest	77.50 %	77.18 %
Decision Tree	76.02 %	75.91 %
KNN	76.70 %	75.27 %
Baseline Movie Average	72.16 %	70.00 %
Baseline User Average	71.82 %	69.73 %

Table 1. Model and baseline accuracy can be seen for the validation and test sets.

4.6. XGBoost

For XGBoost, the hyperparameters we tuned were eta, n_estimators, subsample, gamma, reg_lambda, max_depth, and min_child_weight. First, we used RandomizedSearchCV on a subset of these features. We used a range of [2, 10] for max_depth, [0, 10] for min_child_weight, [0.25, 1.0] for subsample, [0.01, 0.3] for eta, and [10, 1000] for n_estimators. Next, we used GridSearchCV with a parameter grid of [2, 3, 4, 5] for min_child_weight, [0.7, 0.8, 0.9, 0.98] for subsample, [0.01, 0.23, 0.235, 0.25, 0.245, 0.25, 0.26, 0.265] for eta, and [200, 300, 350, 400] for n_estimators. Once we received the best parameters from GridSearchCV, we manually tried a few values for some of these parameters. Eventually we settled on the parameters: eta = 0.07, n_estimators = 216, subsample = 0.70, max_depth = 5, min_child_weight = 4, eval_metric = 'logloss', reg_lambda = 0, gamma = 5.

4.7. Software

For this project, we used Python in Jupyter Notebook and git commands to synchronize our works on GitHub.

4.8. Hardware

Each member used their own laptop.

5. Results and Discussion

5.1. KNN

When the K-Nearest Neighbors model was run on the test data, the test accuracy was 75.27%. The features that were most important int his model are userAvg and movieAvg. These metrics describe the average ratings that each user makes and the average rating that each movie gets. These features work for this model because when trying to classify ratings, it is important to factor in the user and

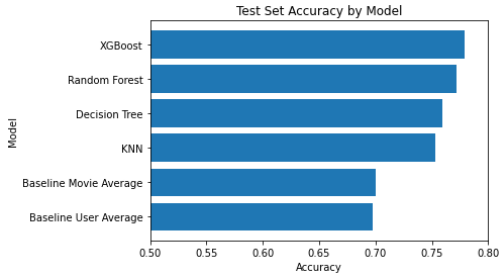

Figure 11. Bar plot showing the accuracy of our models and baselines on the test set.

movie ratings. Figure 12 shows the results of the K-Nearest Neighbors model ran on the test data. The KNN model took approximately 0.03s to fit on Sam's laptop.

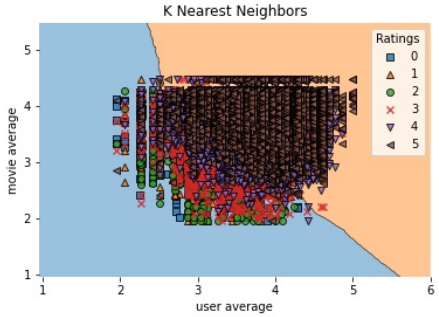

Figure 12. K-Nearest Neighbor model on test data plotted

5.2. Decision Tree

Decision Tree model generated 76.02% of the validation accuracy and 75.91% of test accuracy using gini entropy and maximum tree depth of 4 as shown in Figure 10

The movie average and user average were the two most important features with 0.56 and 0.43 importance values as shown in Figure 13. From the bar graph, we were able to examine that movie's quality or general social standard affects predicting individuals' rate the most, and user average feature affecting actual rate second most. For instance, some people could send out 0 for disliked and 3 for liked movie, but some other people could just send out 5 for all the movies they have watched. Also, we couldn't examine deeply into how genres of the movie affects individual's rating from the decision tree model. The Decision Tree model took approximately 0.12s to fit on Sam's laptop.

5.3. Random Forest

Our final Random Forest ensemble achieved a validation accuracy of 77.50% and a test accuracy of 77.18%. The feature importances of our final Random Forest model are considerable different from that of the Decision Tree and

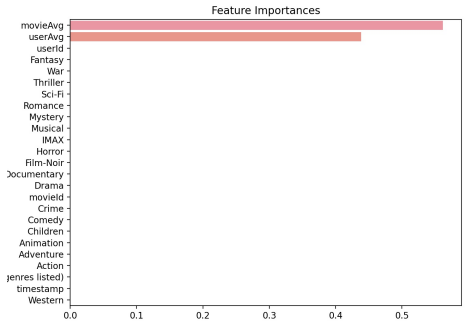

Figure 13. Figure shows feature importance of the decision tree

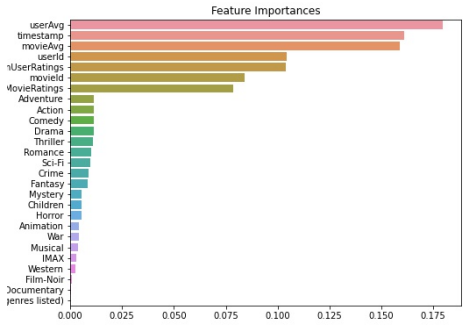

Figure 14. Bar plot showing the importance of all features in the Random Forest model.

XGBoost models. While these other models heavily relied upon the average movie rating and average user rating features, the Random Forest model also puts great importance on the timestamp, user Id, number of user ratings, movie Id, and the number of movie ratings. On Sam's laptop, the Random Forest model took approximately 28.55s to fit.

5.4. XGBoost

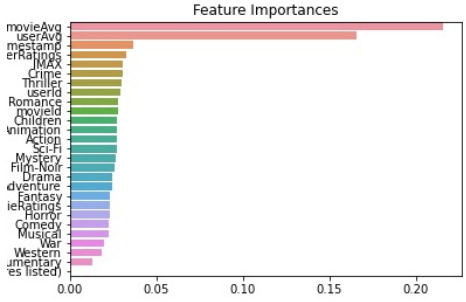

Figure 15. Bar plot showing the importance of all features in the XGBoost model.

For XGBoost, we received an accuracy of 78.08% on the validation set and 77.91% on the test set. This makes XGBoost the best performing model on both the validation and test sets. However, due to our use of RandomizedSearchCV, and then our use of GridSearchCV in the area around our results from the RandomizedSearchCV, it is likely that we found a local maximum during hyperparameter tuning.

Looking at figure 15, we see that similar to the Decision Tree model, the XGBoost's two most important features are the average movie rating and average user rating. However, unlike the Decision Tree, the XGBoost model puts some importance in all of the features it was trained on. On Sam's laptop, the XGBoost model took approximately 3.35s to fit.

5.5. Summary of Results

In general, all of KNN, Decision Tree, Random Forest and XGBoost models showed dominant importance of user average and movie average features. Among the four models that we tried, the Random Forest and XGBoost models showed the top two accuracy and they both included more features than the other two models. However, our most accurate model didn't exceed 80% accuracy, so we believe we should have created more features such as genre correlation mentioned in section 2.2, or from collecting more data if we had more time and computing resources.

6. Conclusions

The goal of our report was to create a model that can achieve high performance in predicting what a user will rate a movie. The use of such a model ultimately would be to recommend movies to users based upon their previous watch history and preferences. We attempted this through the use of various algorithms such as K-Nearest Neighbors, Decision Trees, Random Forest, and XGBoost. We evaluated the performance of each model by removing the 10 most recent ratings for each user in the validation and test sets and using these to predict.

XGBoost was our best performing model, with an accuracy of 77.91% on the test set 5. However, our Random Forest model was not far behind with a 77.18% on the test set. Furthermore, even KNN, our worst performing model, still achieved an accuracy 5.27% higher than our movie average baseline. While most models heavily relied upon the user and movie averages, the XGBoost and especially the Random Forest model put less weight on these features. We did not attempt to use any ensemble methods such as Stacking Classifiers or Voting Classifiers, however they may have promise.

Even though we achieved good accuracy with our models, we reduced the 10 classes of the original 0.5 - 5.0 scale into just two classes, like and dislike. While this binary approach is what Netflix recommendation systems are forced to use, we believe that better performance may be achieved using the original 10 classes, or perhaps even using regression on the 0.5 - 5.0 scale. In future experiments we would attempt those approaches. Furthermore, we believe that more time should be devoted to exploring the classification performance as the number of training samples per user increases, as there tends to be a "cold start" problem where users with very few ratings are not predicted well since they don't have many training examples. Finally, the dataset we used originally had only 100,000 ratings. GroupLens provides versions of this dataset that contain up to 25 million ratings. We believe that with more resource conscious models and better computing resources, these larger datasets can be utilized to achieve greater results.

7. Acknowledgements

We used a data set provided by GroupLens Research which has been collected from March 29, 1996 to September 24, 2018 [3]. It contains approximately 100,000 ratings, with 3,683 descriptive tags applied across 9,742 movies by 610 randomly selected users.

8. Contributions

For the project proposal, Chaiyeen Oh wrote Introduction, Related Works, Resources and Contribution, Andrew Fix for Motivation, and Samuel Peters for Related Works and Evaluation. All three of us agreed to proofread all parts of the report before the submission.

For the computational experiments, Samuel Peters worked on data cleaning, feature generation, baseline generation, Random Forest, XGBoost, and the corresponding plots/analysis.Chaiyeen Oh worked on Decision Tree, KNN Model, and corresponding plots/analysis.Andrew Fix also worked on KNN.

For the final project report, Samuel Peters wrote about evaluation, the dataset, exploratory data analysis and feature generation, the tuning and results of the XGBoost and Random Forest models, as well as the conclusion. Chaiyeen Oh wrote definition, process of finding the best parameters, and analysis of the Decision Tree. Andrew Fix wrote the Abstract, KNN analysis, and Random Forest initial description (3.4)

References

[1] R. L. Chang and T. Pavlidis. Fuzzy decision tree algorithms. *IEEE Transactions on systems, Man, and cybernetics*, 7(1):28–35, 1977.

[2] C. A. Gomez-Uribe and N. Hunt. The netflix recommender system: Algorithms, business value, and innovation. *ACM Trans. Manage. Inf. Syst.*, 6(4), Dec. 2016.

[3] K. Harper. Harper fm, konstan ja. *The movielens datasets: History and context, TiiS*, 5(4):1–19, 2015.

[4] T.-G. Hwang, C.-S. Park, J.-H. Hong, and S. K. Kim. An algorithm for movie classification and recommendation using genre correlation. *Multimedia Tools and Applications*, 75(20):12843–12858, 2016.

[5] Y. Koren. The bellkor solution to the netflix grand prize. *Netflix prize documentation*, 81(2009):1–10, 2009.

[6] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[7] E. Seo and H.-J. Choi. Movie recommendation with k-means clustering and self-organizing map methods. In *ICAART (1)*, pages 385–390, 2010.