Board Game Review predictor:

This Project Compares and contrasts the regressor models:

Linear Regressor and Random Forest Regressor Models

Data is obtained from github: Sean Beck repo

Scrappers/BoardGameGeek

Have 81000 board games information

boardGamesGeek/Games.csv

Command to install git using anaconda

$> Conda install -c anaconda git

1. Import the libraries like sys, pandas, matplotlib, seaborn, sklearn

2. Load the dataset using pandas.read\_csv(name of file)

3. Print the names of the columns in games.

print(games.columns)

print(games.shape). — no. Of rows , no. Of columns

Explore the Data using below:

4. Make a histogram of all the ratings in the average\_rating column

plt.hist(games[“average\_rating”])

plt.show()

—shows we have a lot of zeros

5. Print the first row of all the games with zero scores

print(games[games[“average\_rating”]==0].iloc[0])

6. Print the first row of games with scores greater than 0

print([games[games[“average\_rating”]>0].iloc[0])

7. Remove any rows without any user reviews

games= games[games[“user\_rated”]>0]

8. Remove any rows with missing values

games = games.dropna(axis=0)

9. Make a histogram of all the average ratings

plt.hist(games[“average\_rating”])

plt.show()

10. ID column does not tell any useful information about the games since it is assigned arbitrarily, but it may lead to over fitting. Lets show some correlation between some columns using correlation matrix

cormat= games.corr()

Fig = plt.Figure(figsize= (12,9))

Seaborn.heatmap(format, max =8, square = True)

plt.show()

—it shows correlation between different parameters , ID is highly correlated with average\_rating, min\_age is highly correlated with user\_rated, and we would remove columns after analysis, it will impact the results of our machine learning algorithm.

Dataset Preprocessing:

11. Get all the columns from the data frame

columns= games.columns.tolist()

12. Filter the columns to remove data we do not want

Columns = [c for c in columns if c not in {“bayes\_average\_rating”, “average\_rating”, “type”, “name”, “id”}]

13. Store the variable we’ll be predicting on

Target = “average\_rating”

14. Split the dataset and generate training and test datasets

From sklearn.cross\_validation import train\_test\_split

15. Generate the training set

Trains = games.sample(frac=0.8, random\_state=1)

16. Select anything not in the training set and put it in test set

Test = games.loc[~games.index.isin(train.index)]

17. Print the shape of both of these — showing no. Of games in training and testing set

Print (train.shape). — 45000, 20

print(test.shape). — 11000, 20

Using Linear Regression Model and Random Forest Regression Model and compare and contrast the results

Linear model - -

From sklearn.linear\_model import LinearRegression

From sklearn.metrics import mean\_squared\_error

Initialise model class

LR= LinearRegression()

Fit the model the training data

LR.fit(train[columns], train[target])

#Now generate the predictions and classifying the predictions for the test set

Predictions = LR.predict(test[columns])

#compute error between test predictions and actual values

mean\_squared\_error(predictions, test[target]) —- 2.078019…

Using a non linear regressor model : random forest model

From sklearn.ensemble import RandomForestRegressor

#Initialize the model

RFR= RandomForestRegressor(n\_estimator =100, min\_sample\_leaf=10, random\_state =1)

Fit to the data

RFR.fit(train[columns], train[target])

#make predictions

Predictions = RFR.predict(test[columns])

#compute the error between our test predictions and actual values

mean\_square\_error(predictions, test[target]) — 1.44587..

A non linear model is more accurate , can achieve better results than a linear regressor

test[columns].iloc[0] //we use position based indexing on pandas data frames