# MovieLens Recommender Systems

Collaborative Filtering and
Alternative Methods

# Overview

- Exploratory Analysis
- Feature Engineering for Supervised Models
- Supervised Learning Models:
  - Logistic Regression
  - Linear Regression
  - Random Forest Regression
- Matrix Factorization Models:
  - fast.ai “CollabLearner” model
  - Keras Dot Product model
- Keras Deep Learning model
- Results
- Plan for Implementation

# Recommender Algorithms

- Recommender algorithms are *everywhere*.
  - Google Search
  - Amazon: “Customers who bought this item also purchased…”
  - Facebook: “People You May Know”
  - Youtube: Up Next
  - Goodreads: book recommendations
  - Personalized commercials on social media
  - Curated question sets for online study tools
- Predicts user ratings of item based on ratings given by similar users to the same item.

# Types of Recommendation Algorithms

- Often modeled by using collaborative filtering models.
  - Memory-based: find similarity between users or items
    - Easy-to-explain results
    - Performance decreases with sparsity of data
  - Matrix Factorization
    - Latent factors allow for dimensionality reduction
    - Handles sparse data better
    - Not as transparent

# Collaborative Filtering through Matrix Factorization

- Latent Factors are embeddings.
  - Representations of data in a vector space
  - Optimal number of factors varies based on data
- Predicted rating = dot product of embeddings for user and item, hence, “factors.”
- “Latent” because factors can account for hidden features of the dataset.
  - Patterns are not inherently visible
- Factors begin as random numbers, but numbers change via training to minimize error.
- May be combined with NLP (such as sentiment analysis) when rating is accompanied by text review.
- Factors can be based on user behavior information rather than rating data.

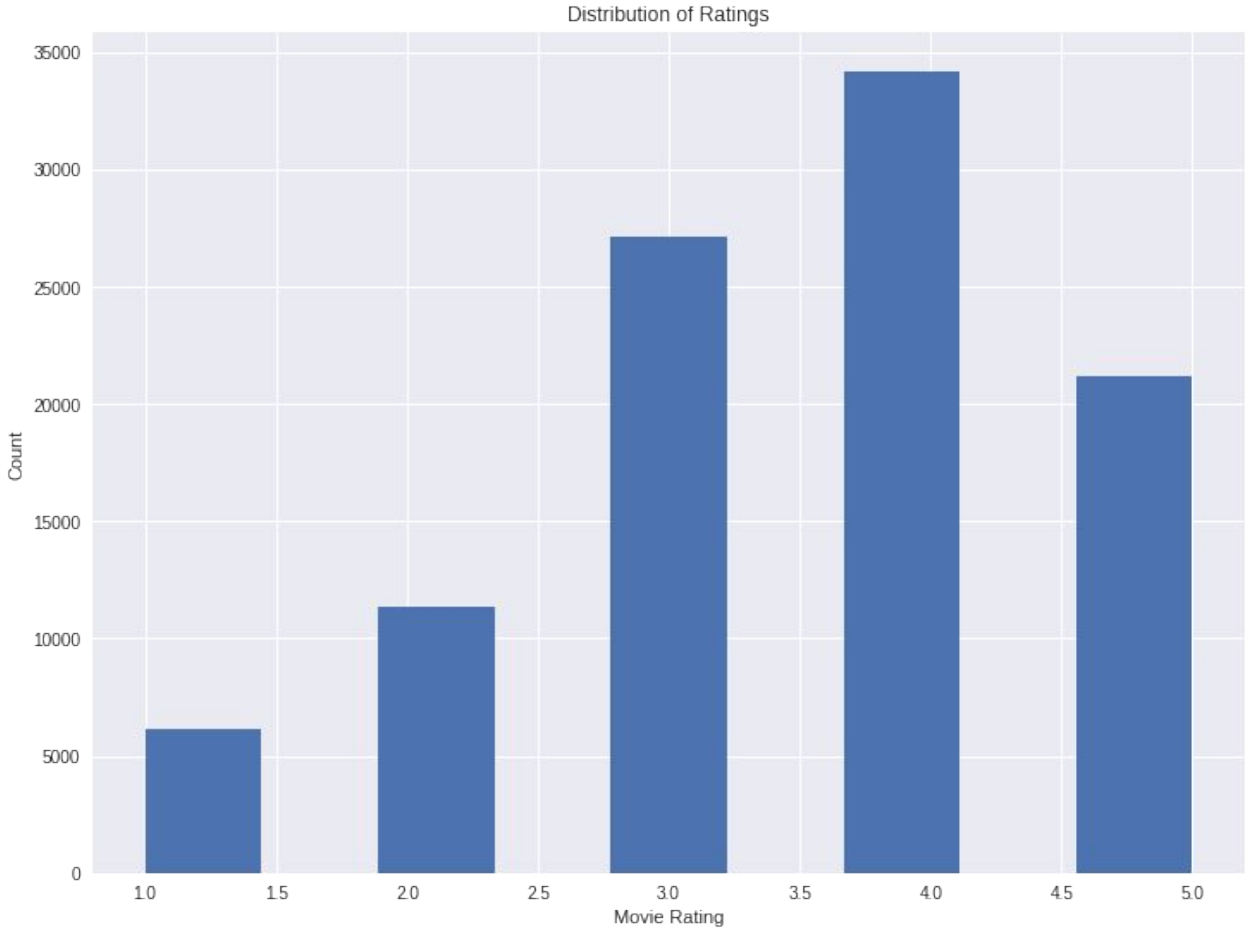
# MovieLens 100k

- Released April, 1998
- 100,000 ratings for 943 users and 1682 movies
  - Each user has a minimum of 20 reviewed movies
  - Sparse dataset: 100,000 / 1,586,126 = 6.3%
- Commonly-used dataset for building and benchmarking recommendation algorithms.
- Benchmarks: https://www.librec.net/release/v1.3/example.html
  - Lowest RMSE: 0.911
- Other information included:
  - User demographic information: age, gender, occupation, zip code
  - Movie genre, release date, IMDb URL
  - Rating timestamps

# Exploratory Analysis

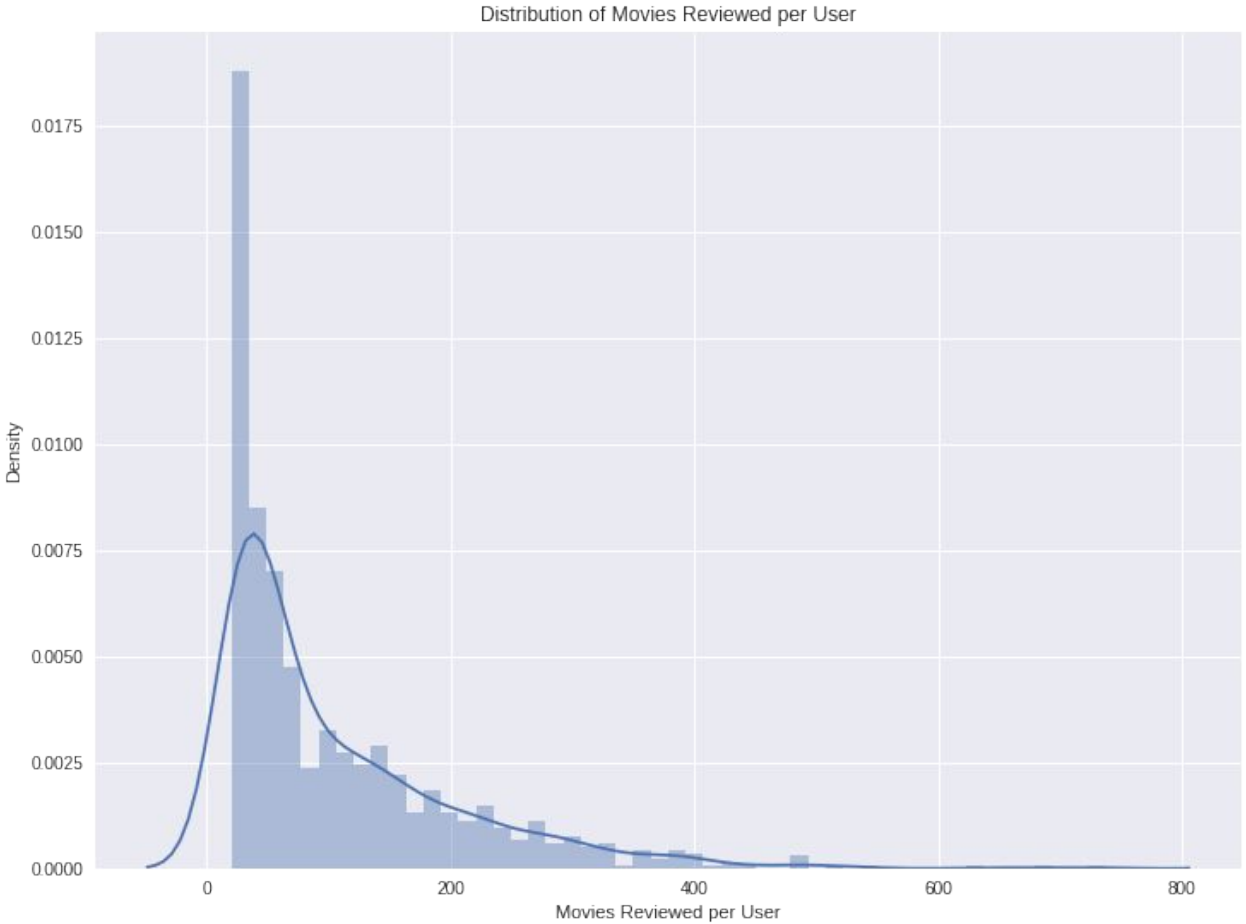
# Ratings

- Mean rating is 3.53 on a scale of 1 to 5.
- This might indicate more willingness of users to provide ratings for movies they like.
- Alternatively, it could indicate that most people are just fairly happy with most movies they see.



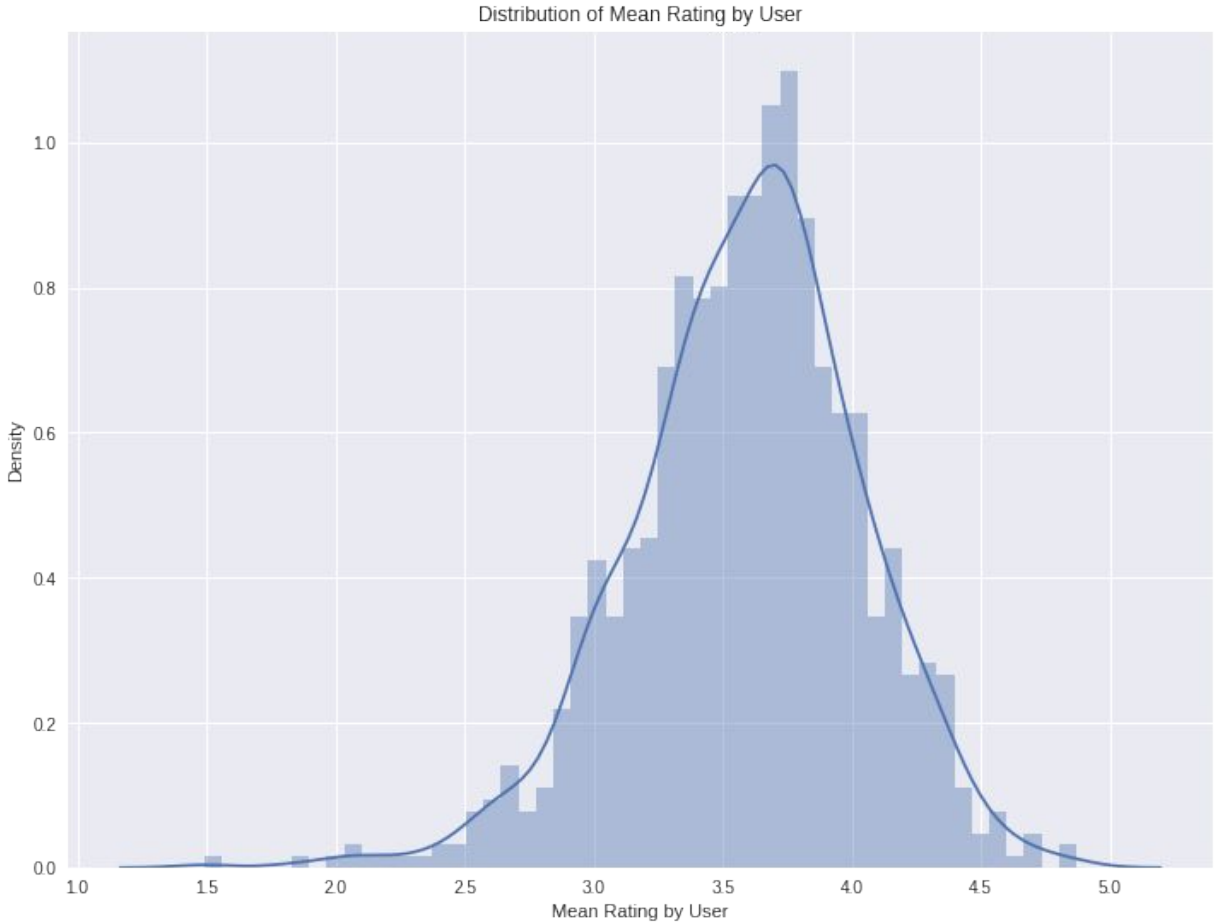
# Movies Per User

- Minimum movie count for a user is 20.
- Mean movie count is 106
- Median is 65, roughly 3.9% of the movies in the dataset.
- Highest review count is 737/1682 (43.8% of films.)



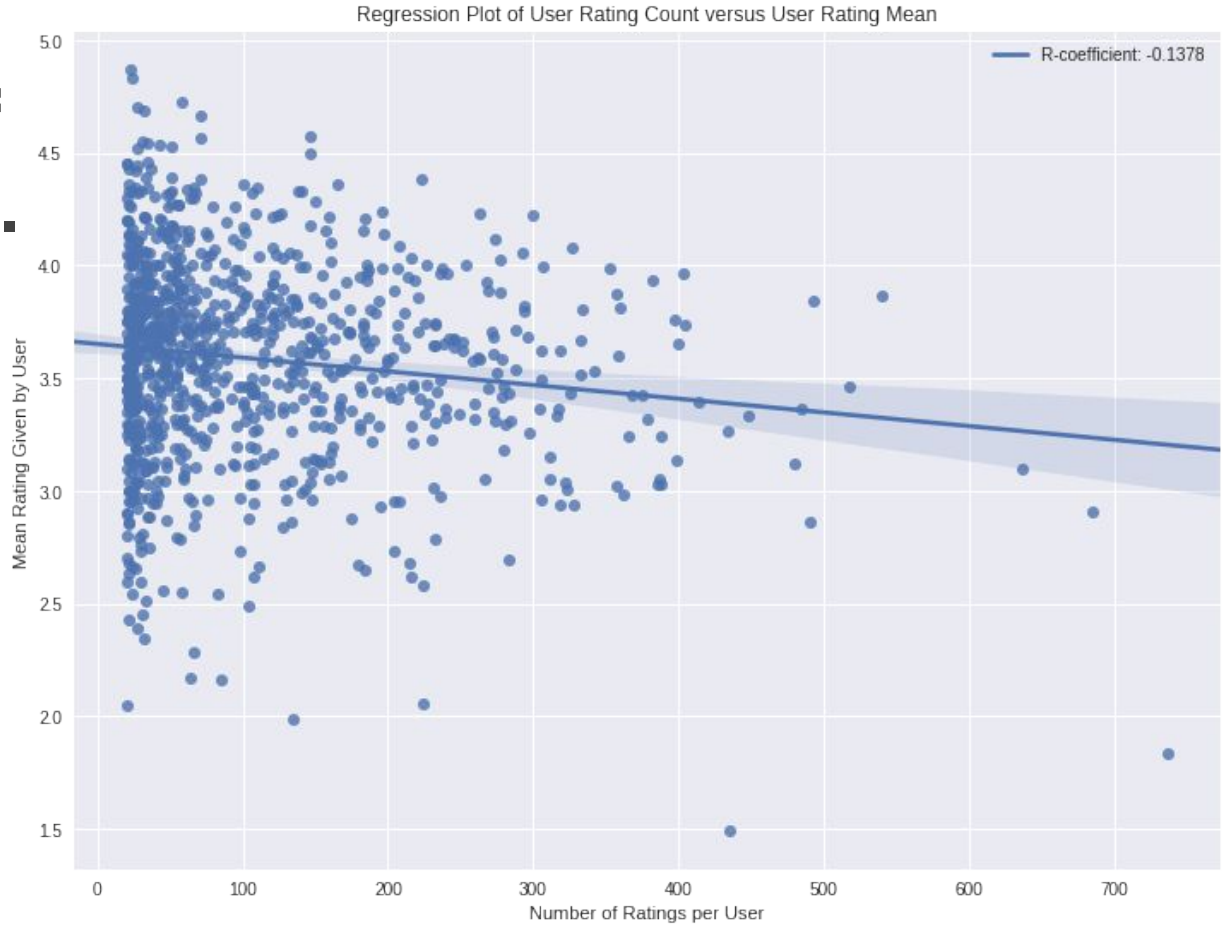
# Mean Rating Per User

- This follows the Ratings distribution fairly closely, with central tendency being above 3, closer to 4.



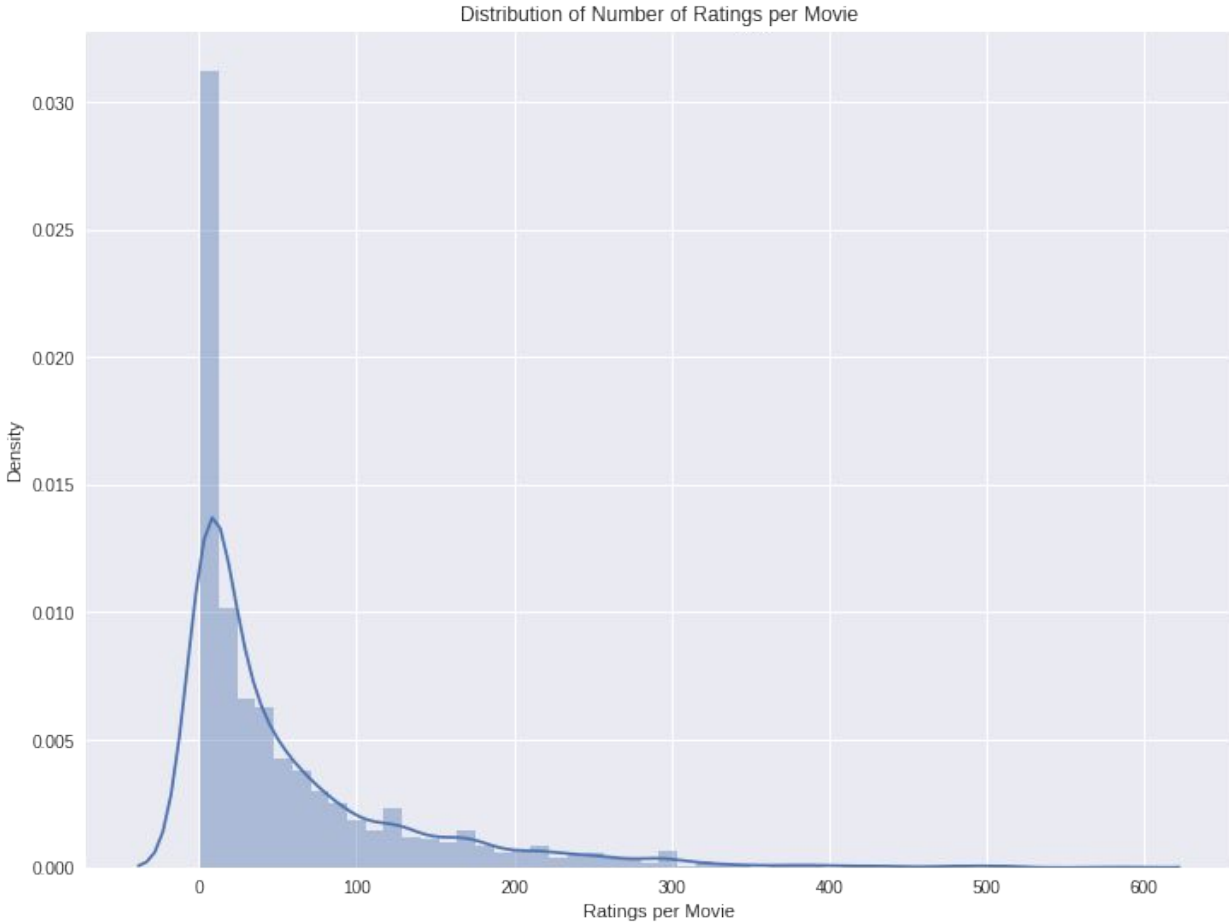
# Number of Ratings vs. Mean User Rating

- Slight negative correlation between number of movies a user has reviewed and average rating they give movies they've seen.



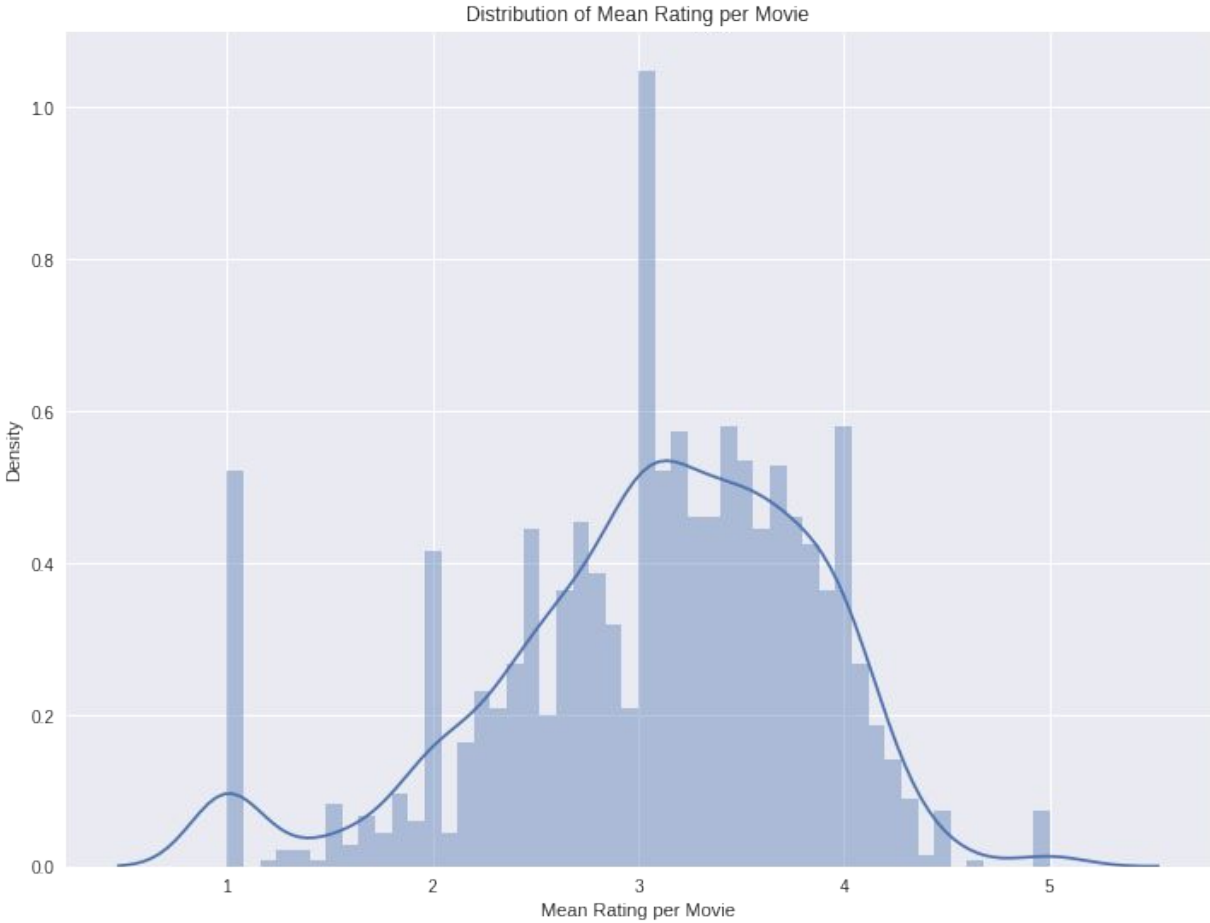
# Ratings Per Movie

- No minimum threshold (unlike Users)
- Median is 27 ratings (2.9% of having reviewed movie.)
- Maximum: 583/943 (61.8% of users) have seen Star Wars (1977).



# Mean Rating Per Movie

- Unusual distribution of mean ratings near whole number values is due to movies with only a small number of ratings.



# Number of Ratings vs. Mean Movie Rating

- Moderate positive correlation between number of ratings and mean rating score.
- More popular movies are more popular for a reason.

# Most Rated Movies

| title | rating | user_count | user_mean | movie_count | movie_mean |
|---|---|---|---|---|---|
| Star Wars (1977) | 4.358491 | 140.624357 | 3.608816 | 583.0 | 4.358491 |
| Contact (1997) | 3.803536 | 119.418468 | 3.531827 | 509.0 | 3.803536 |
| Fargo (1996) | 4.155512 | 142.080709 | 3.600765 | 508.0 | 4.155512 |
| Return of the Jedi (1983) | 4.007890 | 148.802761 | 3.583099 | 507.0 | 4.007890 |
| Liar Liar (1997) | 3.156701 | 119.292784 | 3.525744 | 485.0 | 3.156701 |
| English Patient, The (1996) | 3.656965 | 109.195426 | 3.568388 | 481.0 | 3.656965 |
| Scream (1996) | 3.441423 | 119.707113 | 3.524495 | 478.0 | 3.441423 |
| Toy Story (1995) | 3.878319 | 146.424779 | 3.579055 | 452.0 | 3.878319 |
| Air Force One (1997) | 3.631090 | 106.549884 | 3.522442 | 431.0 | 3.631090 |
| Independence Day (ID4) (1996) | 3.438228 | 156.340326 | 3.554514 | 429.0 | 3.438228 |
| Raiders of the Lost Ark (1981) | 4.252381 | 172.821429 | 3.618826 | 420.0 | 4.252381 |
| Godfather, The (1972) | 4.283293 | 146.791768 | 3.596122 | 413.0 | 4.283293 |
| Pulp Fiction (1994) | 4.060914 | 172.243655 | 3.607240 | 394.0 | 4.060914 |
| Twelve Monkeys (1995) | 3.798469 | 155.737245 | 3.557101 | 392.0 | 3.798469 |
| Silence of the Lambs, The (1991) | 4.289744 | 171.782051 | 3.628187 | 390.0 | 4.289744 |
| Jerry Maguire (1996) | 3.710938 | 146.171875 | 3.581355 | 384.0 | 3.710938 |
| Rock, The (1996) | 3.693122 | 155.896825 | 3.544660 | 378.0 | 3.693122 |
| Empire Strikes Back, The (1980) | 4.204360 | 180.822888 | 3.612773 | 367.0 | 4.204360 |
| Star Trek: First Contact (1996) | 3.660274 | 155.632877 | 3.558838 | 365.0 | 3.660274 |
| Titanic (1997) | 4.245714 | 120.645714 | 3.538866 | 350.0 | 4.245714 |

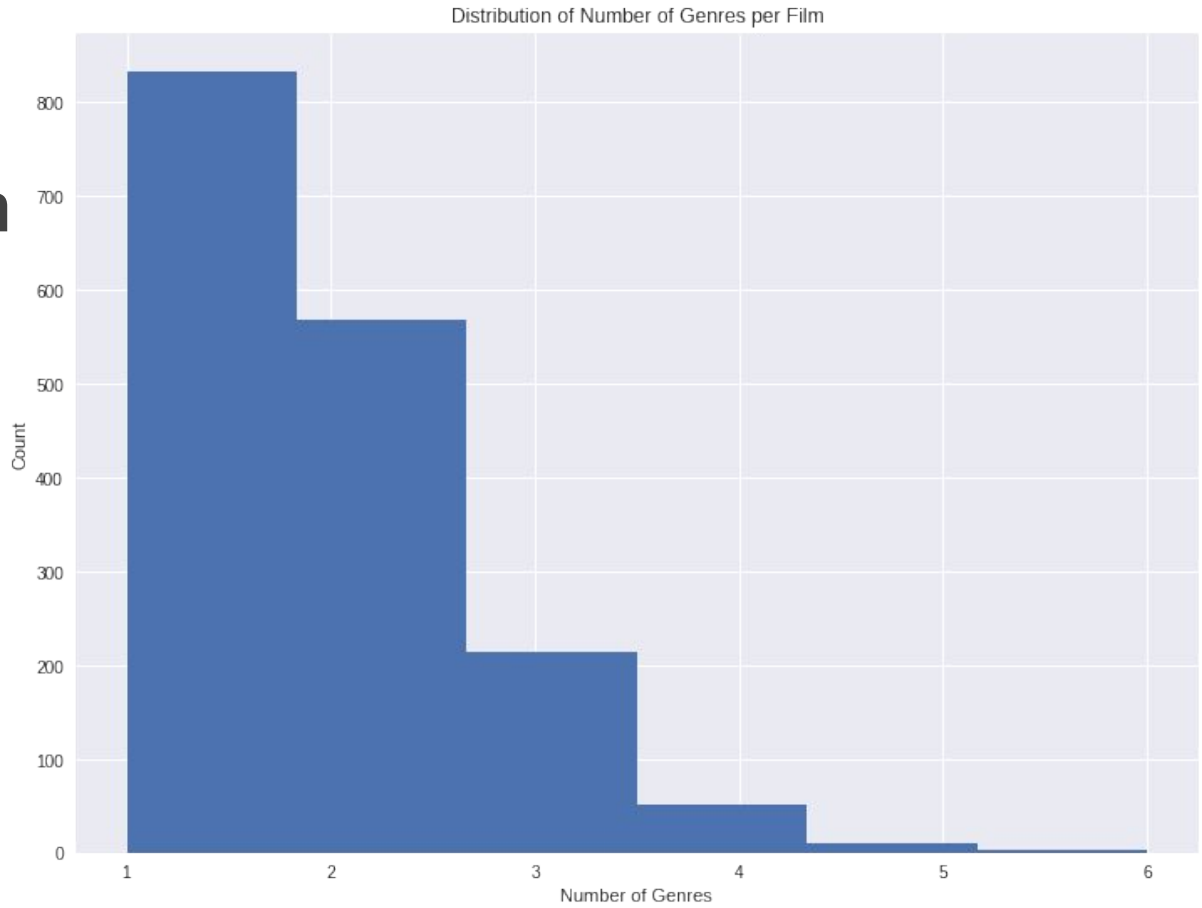
# Highest Rated Movies

- Bias due to small sample size.
- This is indicative of “cold start” problem for recommender systems.
- Users or items with few reviews will get less accurate predictions.
- We can still use data because of user rating count floor.

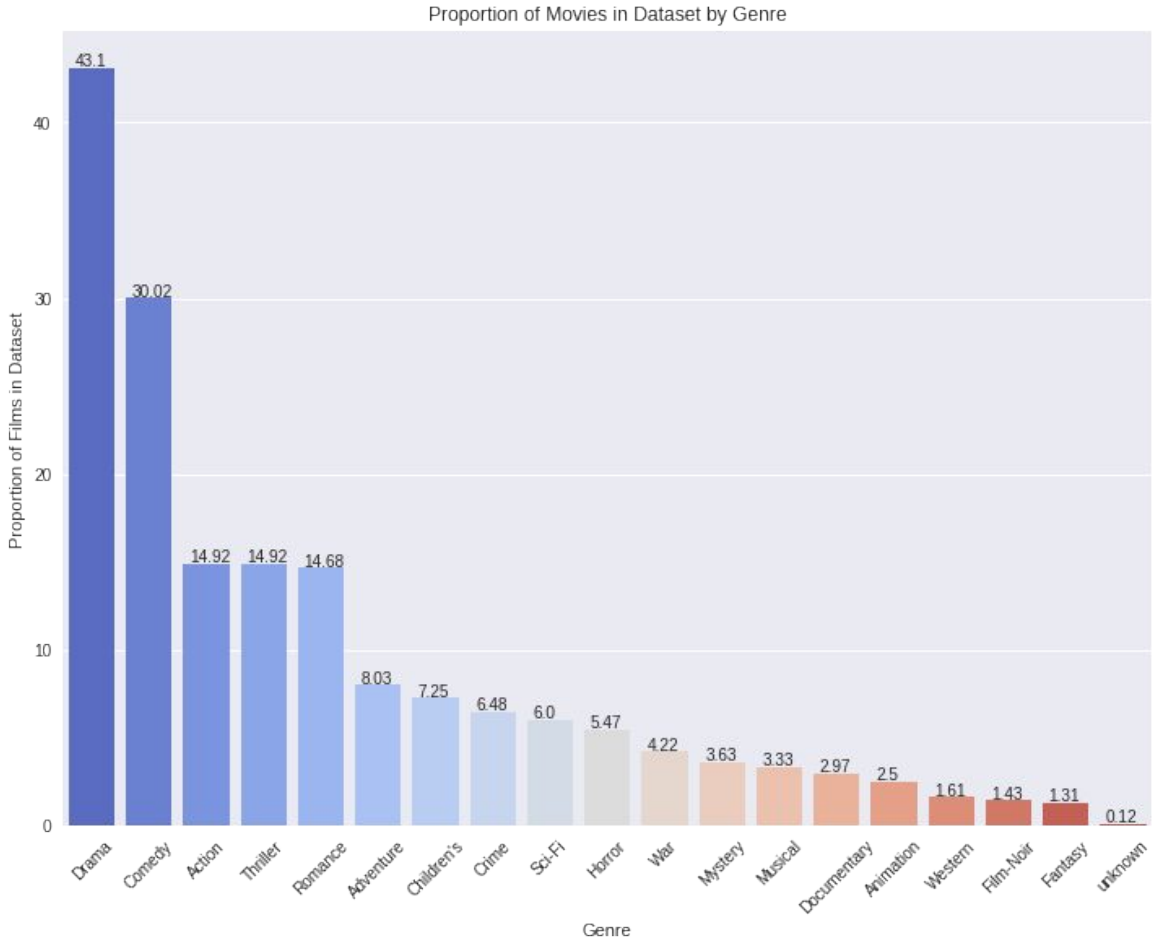
| title | rating | user_count | user_mean | movie_count | movie_mean |
|---|---|---|---|---|---|
| They Made Me a Criminal (1939) | 5.000000 | 208.000000 | 4.086538 | 1.0 | 5.000000 |
| Marlene Dietrich: Shadow and Light (1996) | 5.000000 | 300.000000 | 4.223333 | 1.0 | 5.000000 |
| Saint of Fort Washington, The (1993) | 5.000000 | 239.000000 | 3.506880 | 2.0 | 5.000000 |
| Someone Else's America (1995) | 5.000000 | 263.000000 | 3.581749 | 1.0 | 5.000000 |
| Star Kid (1997) | 5.000000 | 62.333333 | 3.822636 | 3.0 | 5.000000 |
| Great Day in Harlem, A (1994) | 5.000000 | 636.000000 | 3.097484 | 1.0 | 5.000000 |
| Aiqing wansui (1994) | 5.000000 | 281.000000 | 3.295374 | 1.0 | 5.000000 |
| Santa with Muscles (1996) | 5.000000 | 238.000000 | 3.526498 | 2.0 | 5.000000 |
| Prefontaine (1997) | 5.000000 | 326.000000 | 3.919250 | 3.0 | 5.000000 |
| Entertaining Angels: The Dorothy Day Story (1996) | 5.000000 | 34.000000 | 3.705882 | 1.0 | 5.000000 |
| Pather Panchali (1955) | 4.625000 | 247.250000 | 3.454702 | 8.0 | 4.625000 |
| Some Mother's Son (1996) | 4.500000 | 460.500000 | 3.201896 | 2.0 | 4.500000 |
| Maya Lin: A Strong Clear Vision (1994) | 4.500000 | 295.750000 | 3.592109 | 4.0 | 4.500000 |
| Anna (1996) | 4.500000 | 318.500000 | 3.346221 | 2.0 | 4.500000 |
| Everest (1998) | 4.500000 | 383.500000 | 3.983140 | 2.0 | 4.500000 |
| Close Shave, A (1995) | 4.491071 | 166.991071 | 3.558429 | 112.0 | 4.491071 |
| Schindler's List (1993) | 4.466443 | 177.409396 | 3.636567 | 298.0 | 4.466443 |
| Wrong Trousers, The (1993) | 4.466102 | 190.932203 | 3.614172 | 118.0 | 4.466102 |
| Casablanca (1942) | 4.456790 | 182.823045 | 3.663195 | 243.0 | 4.456790 |
| Wallace & Gromit: The Best of Aardman Animation (1996) | 4.447761 | 178.074627 | 3.671052 | 67.0 | 4.447761 |

# Genre Breakdown

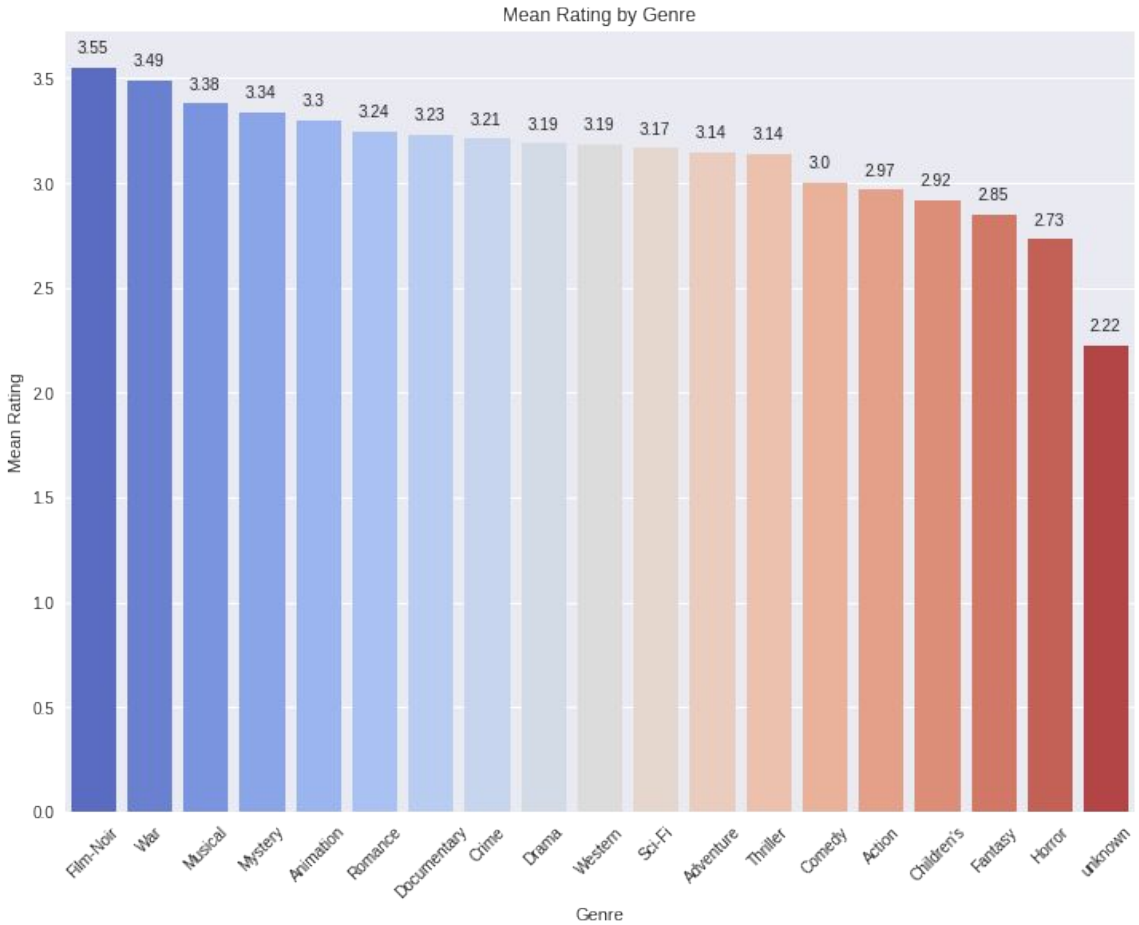
- Moderate positive correlation between number of ratings and mean rating score.
- More popular movies are more popular for a reason.



# Proportion Of Genres



# Mean Rating Of Genres



# Feature Engineering

# Matrix Factorization Model

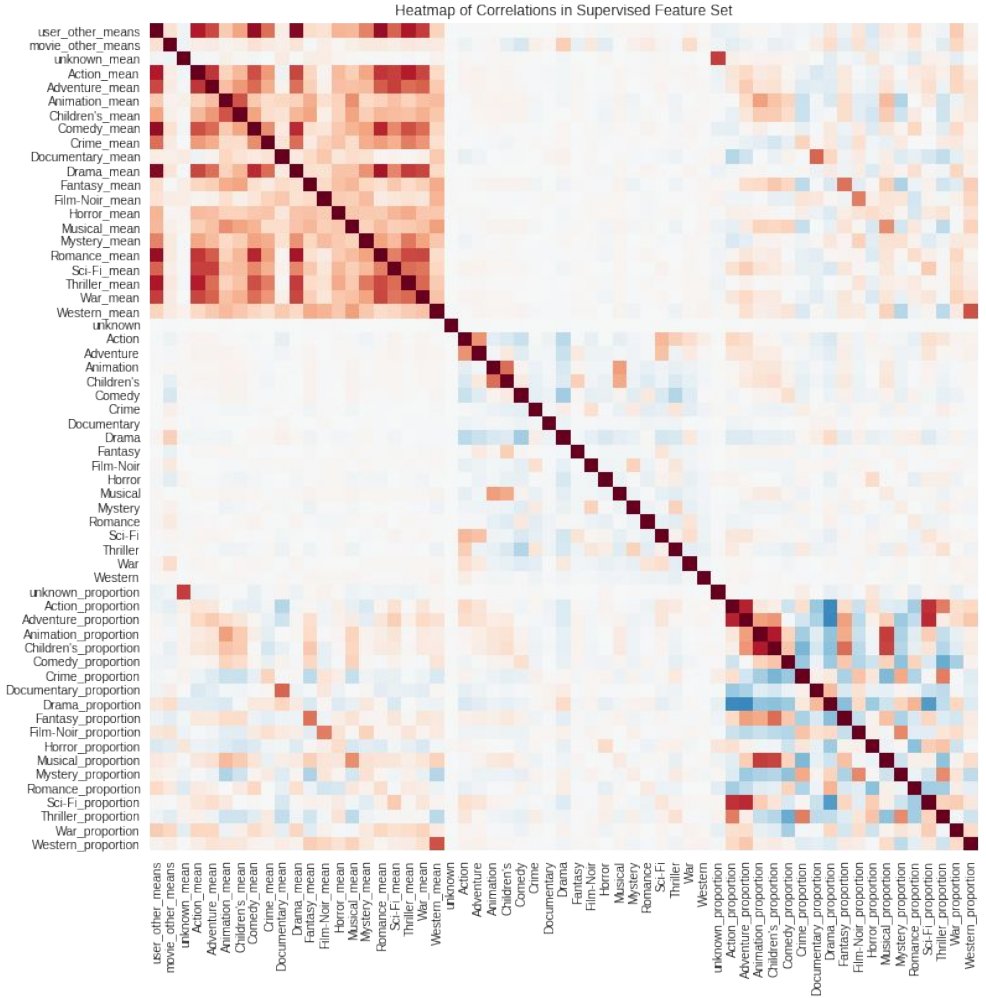
- All that's needed for matrix factorization is the dataframe "ratings," which contains userId, movieId, and rating (we can discard timestamp).
- For (relative) transparency, we can merge movie titles into this dataframe.
- That's it! The models can take care of the rest.

# Supervised Model

- Since we're not using matrix factorization for this version of the feature set, we have to do some more work.
- For each user-movie pair:
  - Mean rating given to movies by other users (exclude current user).
  - Mean rating given by user to other movies (exclude current movie).
  - N-hot-encoded genre for each movie
  - Proportion of other films of each genre rated by user (exclude current movie).
  - Rating given to other films of each genre by user (exclude current movie).
- Pro: Greater ease of interpretability.
- Con: Huge computational cost, time of execution.

# Supervised Feature Correlations

- Some collinearity here between features connected by associated genres like “Action-Adventure” and “Children’s-Animation.”
- This is somewhat unavoidable because of n-hot-encodings on genre.



# Modeling

# Supervised Learning Dataset Characteristics

- 61 features
- Some amount of collinearity between some of those features
- Mix of continuous and discrete numerical quantities, as well as n-hot-encoded categorical variables.
- We can try using multivariate logistic regression to classify star ratings.
- However, given the nature of the target variable (rating), our best options for models are likely linear regression (although we don't expect much from it) and random forest regression.
- RFR will likely perform best among these.

# A note on model evaluation:

- Train-test split for SKLearn models is often 0.60-0.40, 0.667-0.333, or 0.70-0.30.
- Train-test split for neural net models is often 0.90-0.10
- To maintain consistency between model types, we are using a compromise split of 0.80-0.20.
- This also makes 5-fold cross-validation a bit more consistent with our scores and losses from the initial train-test split.

# Multivariate Logistic Regression

- R-squared for Training set: 0.359
- R-squared for Test set: 0.366
- RMSE for Training set: 1.137
- RMSE for Test set: 1.127

The loss here is very high, worse than the RMSE for predicting the mean (3.530) for every rating, 1.121.

We'd like to reduce loss to within a star rating if possible, and ideally we'd like it to be lower than the librec.net benchmark of 0.911.

# Linear Regression

- R-squared for Training set: 0.289
- R-squared for Test set: 0.292
- RMSE for Training set: 0.950
- RMSE for Test set: 0.944
- RMSE for rounded Test set: 0.985

The loss is somewhat lower than logistic regression, but still nowhere near where we'd like to be. We are rounding the prediction because our ground truth values are discrete.

# Random Forest Regression (10 estimators)

- R-squared for Training set: 0.944
- R-squared for Test set: 0.706
- RMSE for Training set: 0.266
- RMSE for Test set: 0.610
- RMSE for rounded Test set: 0.649

This is a huge improvement! RFR appears to be overfitting, but the loss is now much lower than the benchmark.

# Random Forest Regression (100 estimators)

- R-squared for Training set: 0.963
- R-squared for Test set: 0.739
- RMSE for Training set: 0.217
- RMSE for Test set: 0.573
- RMSE for rounded Test set: 0.604

Unsurprisingly, increasing the number of estimators made our random forest model more robust. This is now significantly lower than our benchmark of 0.911 RMSE.

# Random Forest Regression Grid Search Result

- Parameter space:
  - A higher min_samples_split can sometimes reduce noise in decision trees
    - 2 (default), 4, and 8
  - The number of estimators can also have an effect on accuracy and consistency.
    - 50, 100 (future default for SKLearn), and 150
- A higher number of estimators did improve performance.
- Increasing min_samples_split did not.
- RMSE: 0.608 with 150 estimators and min_samples_split = 2

# Neural Net Modeling

- We will be exploring two options, fast.ai's collab_learner, and a some-assembly-required approach with Keras.
- fast.ai's model is ready to test with virtually no setup, but offers little in terms of flexibility in tuning models.
- Keras is the opposite; models must be constructed layer by layer, and thus offer much greater flexibility, but you have to know exactly what you're doing.

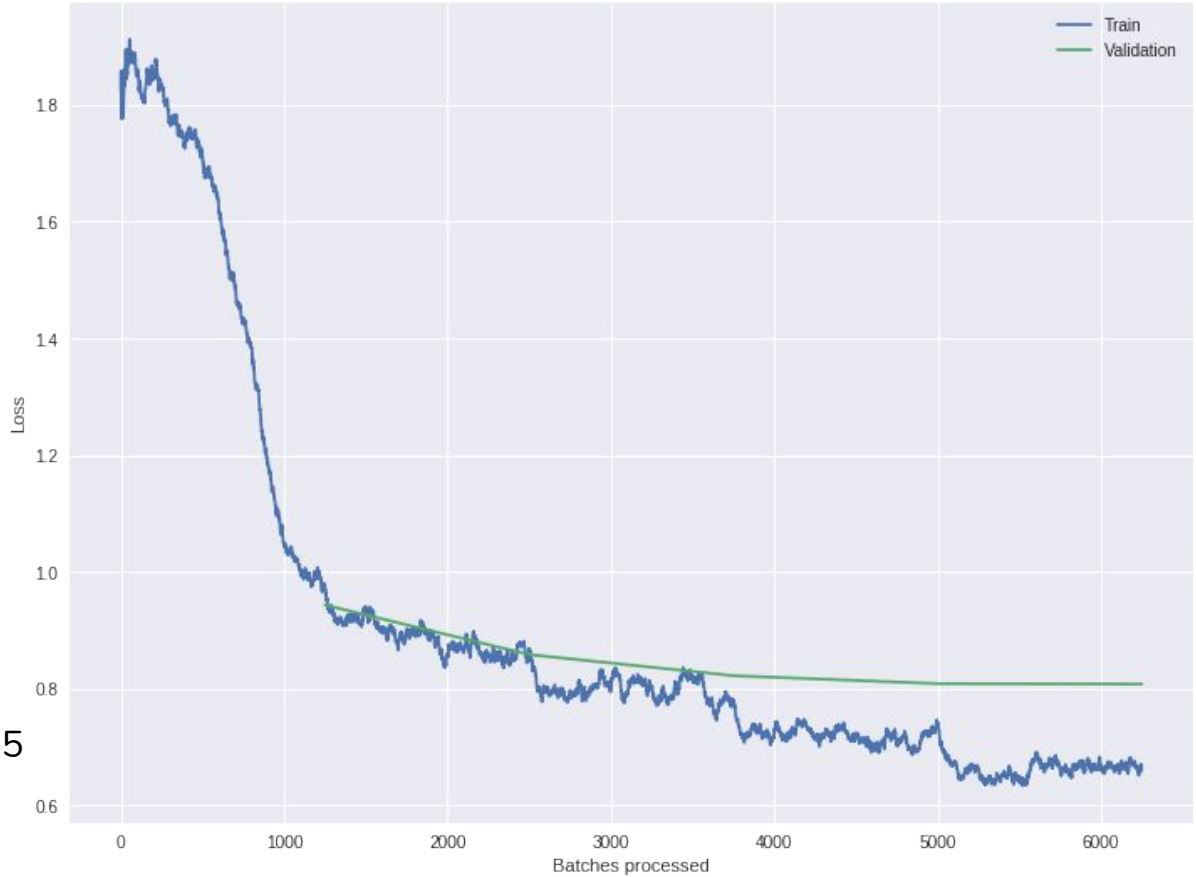
# fast.ai Model (Collab)

- fast.ai is a research foundation and MOOC.
  - Has its own library of machine learning modules
  - Built on top of PyTorch
  - Usable out-of-the-box
  - Provides a good starting point for new machine learning practitioners
- Some objects are similar to those found in other modules
- “DataBunch” is an expanded version of “DataFrame” with built-in train-test splitting.
- Works well, but is unstable, as it is constantly being developed.
  - During completion of this project, “pct_val” changed to “valid_pct.”
  - When using a platform that always imports up-to-date modules (like Google Colab does), this can break code.

# fast.ai Collab Learner: 15 Factors

First pass:

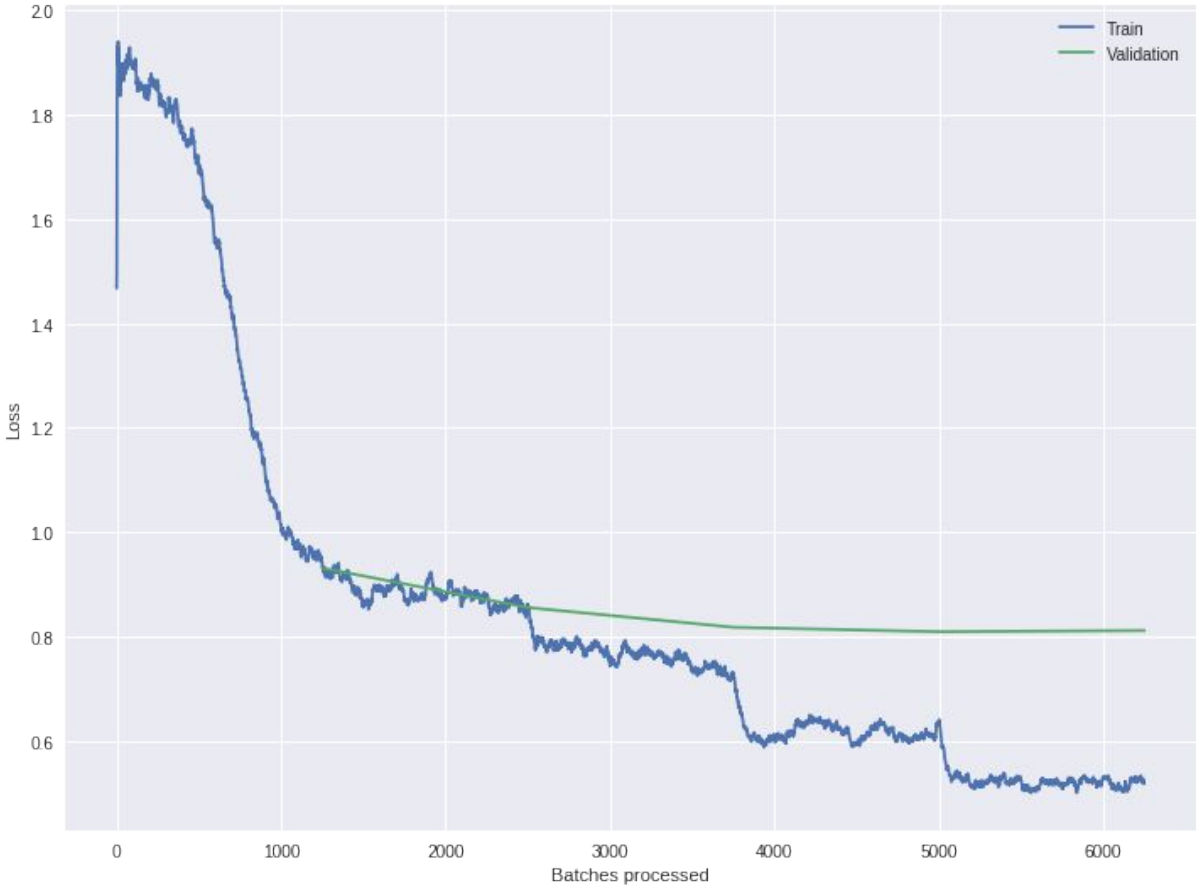
- Parameters:
  - SGD optimizer
  - learning rate: 0.005
  - weight decay: 0.05
  - batch size: 64
  - momentum: 0.95-.085
- Training loss: 0.809
- Validation loss: 0.901



# fast.ai Collab Learner: 30 Factors

First pass:

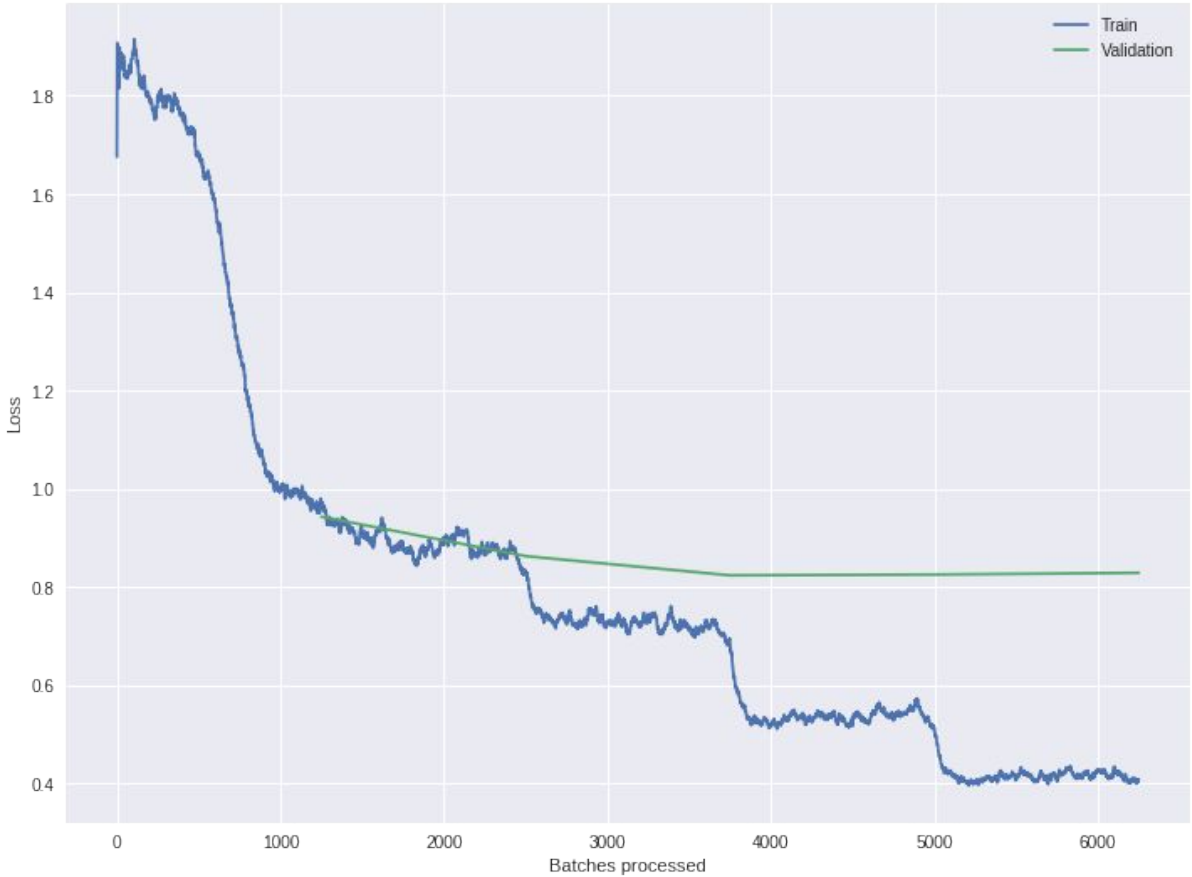
- Training loss: 0.720
- Validation loss: 0.901
- Greater degree of overfitting.



# fast.ai Collab Learner: 45 Factors

First pass:

- Training loss: 0.639
- Validation loss: 0.910
- Even more overfitting
- Loss starts to increase.



# fast.ai Grid Search

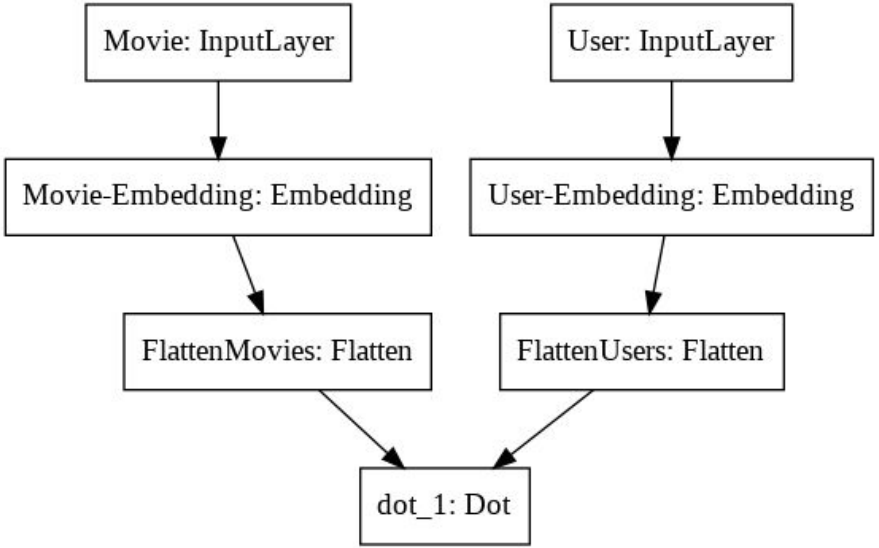
- The losses from all of the fast.ai models (15, 30, and 45 latent factors) were all pretty good!
- 15 showed the least overfitting, and 15 and 30 were close. We'll try 15 and 20 in the grid search:
- Parameter space:
  - lr: 0.005, 0.007
  - Momentums: (0.95, 0.85), (0.95, 0.7)
  - n_factors: 15, 20
- RMSE: 0.903: 0.005 Learning Rate, (0.95, 0.7) Momentums, 20 Latent Factors

# Keras

- Popular ML library that provides a cohesive interface for other libraries: TensorFlow, Microsoft Cognitive Toolkit, Theano, and PlaidML.
- Here we are using the TensorFlow backend.
- Less straightforward than fast.ai library, but provides a great deal of flexibility.
- Still easier to put a model together than working entirely in TensorFlow.

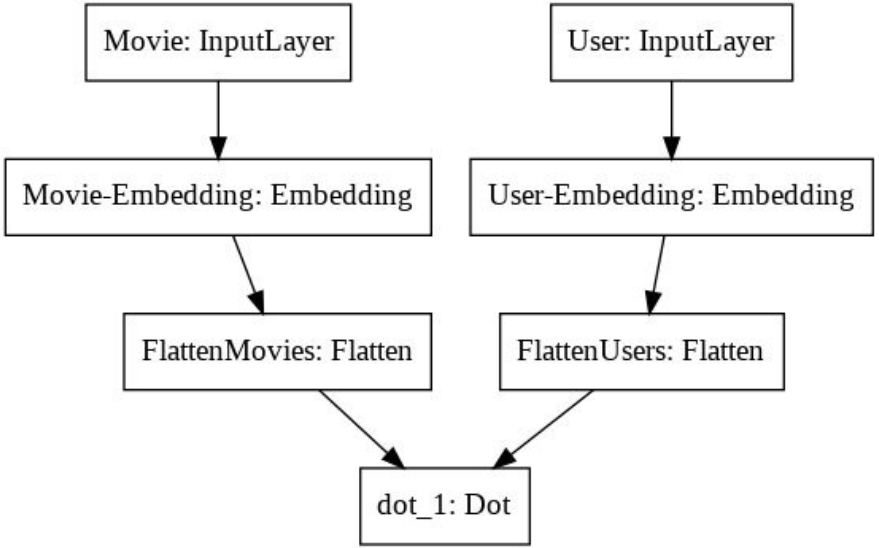
# Keras Dot Product Model 1

- Parameters as close as possible to best fast.ai model
    - 20 Latent Factors
    - Stochastic Gradient Descent Optimizer
    - Learning Rate: 0.005
    - 0.95 Momentum (Keras optimizers don't have option for a tuple)
    - Weight decay: 0.05
    - Batch size: 64 (Default for both Keras and fast.ai)
- Remarkably poor results: virtually no decrease in error over 25 epochs.
- RMSE: 3.715



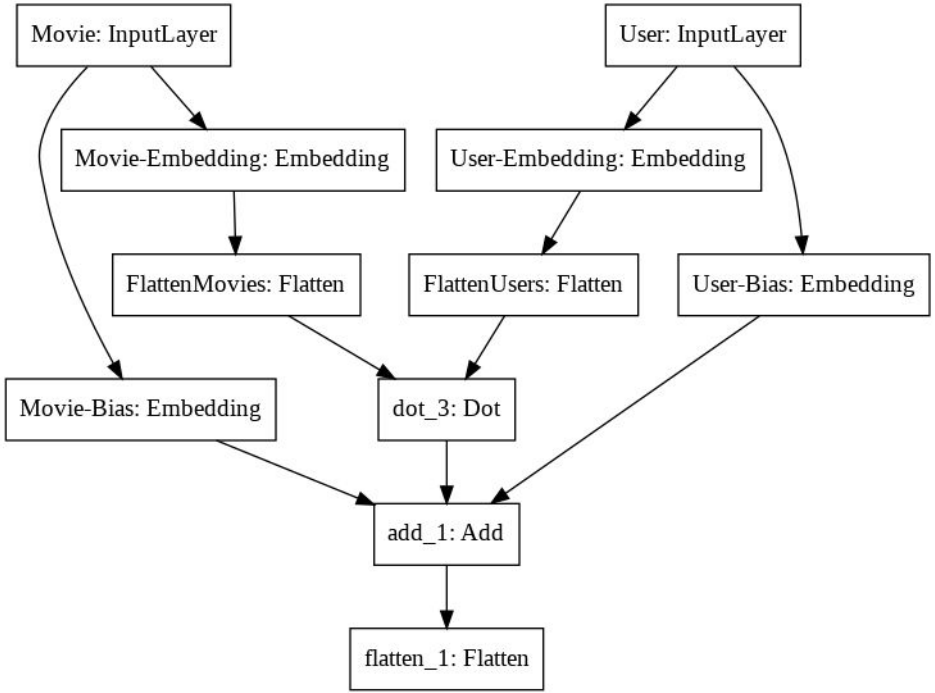
# Keras Dot Product Model 2

- Default parameters for Keras Adam Optimizer
  - Learning Rate: 0.001
  - Weight decay: 0.01
- RMSE 1.034 still over benchmark, but significantly better than before.



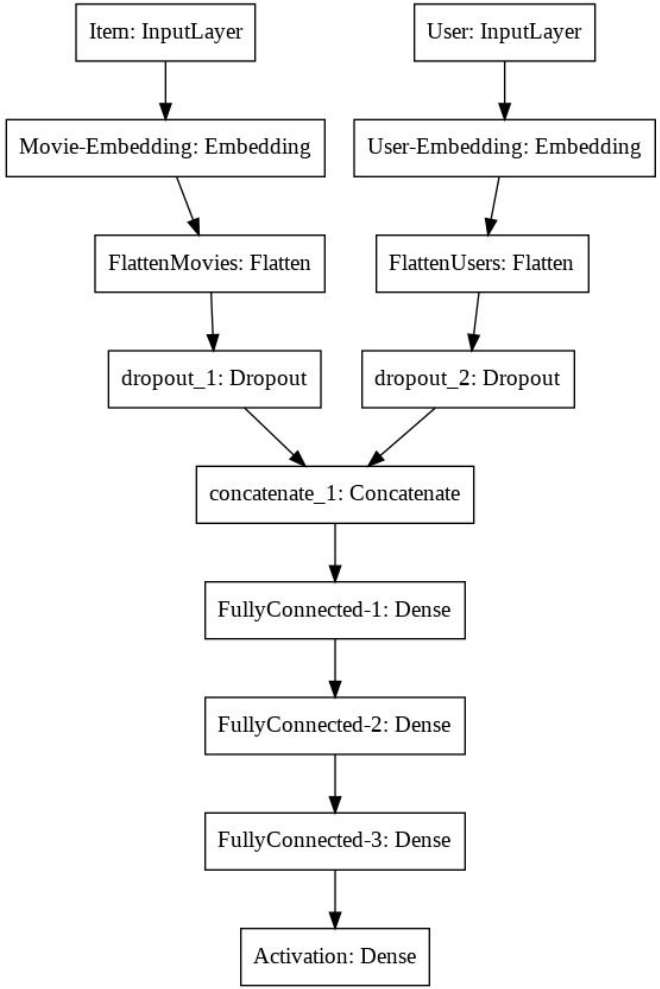
# Keras Dot Product Model With Bias Terms

- Bias terms are additional weights that accompany latent factors, but which are not included in the dot product.
- Users and Movies each get a bias layer, so each user and each movie has its own bias term
- Otherwise the same as Keras Dot Product Model 2
- RMSE: 1.052, pretty close to 2nd model.



# Keras Deep Neural Network Model

- Structurally the same as Nipun Batra's model.
- Dropout layers after Embedding layers.
- Uses Concatenate instead of Dot Product.
- After Concatenate, there are three fully-connected layers.
- Fully-connected layers separated by dropout layers to avoid overfitting.
- RMSE: 0.953 with Dropout = 0.2



# Model Comparison Results

- Random Forest Regression performed the best out of all the models, but it required the most feature engineering.
- fast.ai's models got the best performance for how much effort was required to set them up, but they show signs of instability for long-term use.
- Keras Deep Neural Network models performed reasonably well, but finding parameters that could improve performance is time-consuming.

# Potential Implementation

- Using the random forest regression model, it would be possible to take the userId of a user with at least a few movie ratings tied to their account, and pass in a dataset entirely of that userId paired with the movieId of every movie that user has not already seen.
- This would generate a list of predicted ratings for that user.
- A subset of these could be taken, and shown to the user as recommendations.
- Assuming the user is new had not provided ratings for any movies, a short survey could be given to them, asking what genres they typically enjoy.
- In order to solve the “cold start” problem with respect to movies, it would be possible to seed the list of recommended movies provided to each user with movies with few ratings.

# Expandability/Maintenance

- Streaming service
- Database set up to record user behavior connected to each movie.
- Without explicit ratings:
    - Does user watch part of a movie, then start watching something else?
    - Does user watch movie more than once?
- Re-calculate proportion and mean rating data periodically
- Random Forest on entire dataset would likely become too cumbersome to run as database grows.
    - Take random sample of dataset, and build on top of that
    - Take only most recent data
    - Alternative: neural net model on larger dataset

# Sources (Part 1)

MovieLens 100k dataset:
https://grouplens.org/datasets/movielens/

MovieLens 100k dataset on Kaggle:
https://www.kaggle.com/prajitdatta/movielens-100k-dataset

Prince Grover's article on collaborative filtering methods, covering memory-based and model-based approaches:
https://towardsdatascience.com/various-implementations-of-collaborative-filtering-100385c6dfe0

# Sources (Part 2)

fast.ai lessons on collaborative filtering, with matrix factorization in Microsoft Excel:
https://course.fast.ai/videos/?lesson=4
https://course.fast.ai/videos/?lesson=5

fast.ai changes:
https://github.com/fastai/fastai/search?q=pct_val&unscoped_q=pct_val

# Sources (Part 3)

Nipun Batra's Keras models (shallow and deep neural networks):
https://nipunbatra.github.io/blog/2017/recommend-keras.html

Will Wolf's Keras models (adding a bias term to CF):
http://willwolf.io/2017/04/07/approximating-implicit-matrix-factorization-with-shallow-neural-networks/

MAE vs. RMSE as loss function:
https://medium.com/human-in-a-machine-world/mae-and-rmse-which-metric-is-better-e60ac3bde13d

# Sources (Part 4)

JJ Espinoza on how collaborative filtering is used in the entertainment industry (tailored movie trailers):
https://twimlai.com/twiml-talk-220-building-a-recommender-system-from-scratch-at-20th-century-fox-with-jj-espinoza/

Vishakha Lall on the basics of Google Colab:
https://medium.com/lean-in-women-in-tech-india/google-colab-the-beginners-guide-5ad3b417dfa