Movie Predictions with Machine Learning
By Matt Brierley

Problem Statement:

I worked with the IMDB and Movielens datasets last semester in Databases and thought it would be interesting to revisit them from a machine learning perspective. My initial intent was to create a recommendation system, such as the one used by Netflix, but wanted to also include a rating prediction system. My overall goals were to create an accurate movie rating system, and two recommendation systems, one that takes a movie and returns a list of similar movies (using KNN), and one that takes a user and returns a list of highest predicted ratings based on data from similar users (using Collaborative filtering).

Dataset:

3 main options were considered:

- IMDB
- Movielens (Full)
- Movielens (Small)

I started with IMDB, but the dataset proved far too large to process for this project. I had similar issues with the full Movielens set, and so settled on the small Movielens set. This decision resulted in much faster iterative processing and testing, but at the cost of reduced information.

The Movielens small dataset used consisted of 3 CSV files:

- Movies.csv
  - movieId, title, genres
    - Genres may include; Adventure, Comedy, etc
  - 9742 unique titles
- Ratings.csv
  - userId, movieId, rating, timestamp
  - 100836 unique ratings from 610 users
- Tags.csv
  - userId, movieId, tag, timestamp
    - Tags may include; Quirky, family, Pixar, Will Ferrell, etc
  - 3683 tags from 610 users

Preprocessing:

This proved to be the bulk of the process, and where the bulk of performance gains were found, with process consisting of;

- Load the 3 CSV files

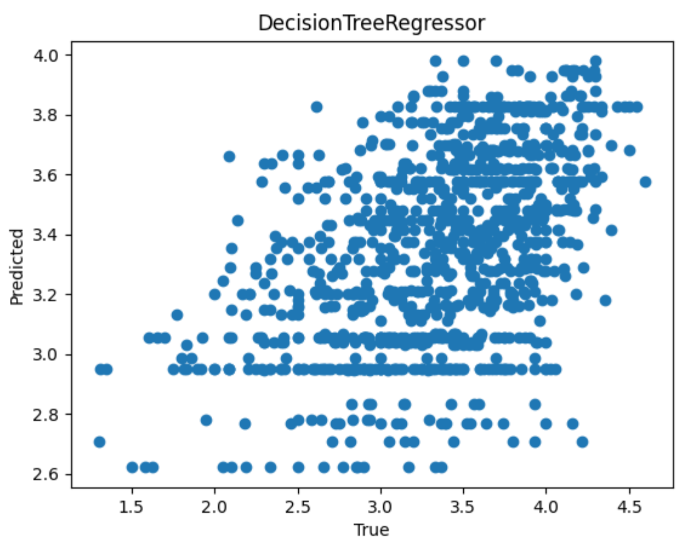
- Group tags per movie, split genres, and encode both
- Group movies by id and average rating, dropping any movie with less than minimum review threshold (a parameter for cross validation)
- Fill missing data, drop timestamps and ids
- Merge into one dataset

Final processed dataset shape ~2500x1300

PART I: Rating predictions from regression model training

Initial Results:

While all models were trained using cross validation techniques from the beginning, it wasn't until later in training that I tried averaging and adding a minimum review threshold.
This resulted is very poor initial results and correlation, with MSE scores near 1 and R2 scores near 0.



Final trained models:

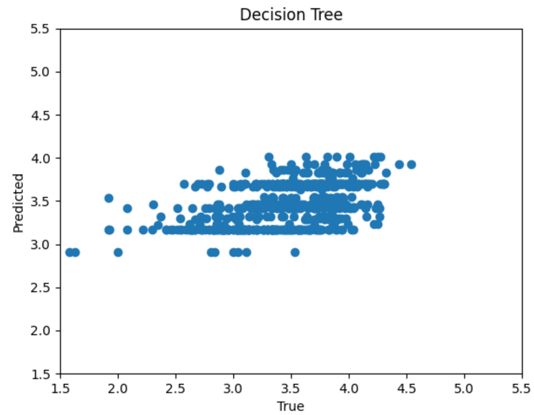
After the initial unsatisfactory results, I went back to preprocessing, averaging ratings per movie and adding a minimum number of reviews per movie as a cross validation parameter. These each gave substantial improvement in accuracy, and resulted in much better trained models as follow;

Trained Decision Tree Regressor:

Configuration:

- criterion: mse
- max_depth: 6
- min_samples_leaf: 10
- min_samples_split: 40
- min_reviews: 12

MSE : 0.17532608954033382
R2 Score: 0.22889578427828328
Adjusted combination: 0.9464303052620505

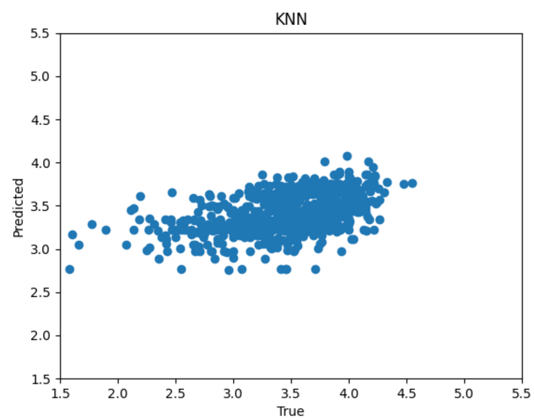


Trained K Neighbors Regressor:

Configuration:

- n_neighbors: 17
- leaf_size: 1
- p: 1
- min_reviews: 11

MSE: 0.18914463803277415
R2 score: 0.2216259747095427
Adjusted combination: 0.9675186633232314

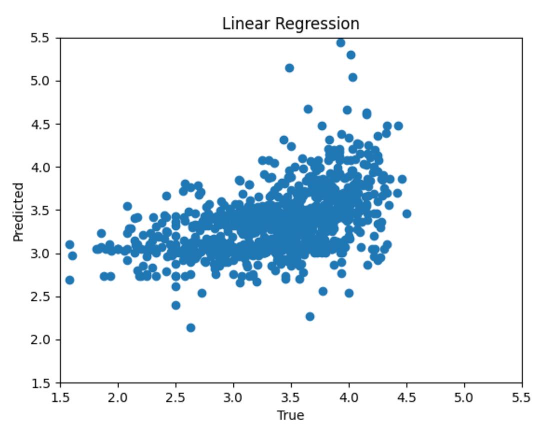


Trained Linear Regressor:

Configuration:

- normalize: False
- min_reviews: 6

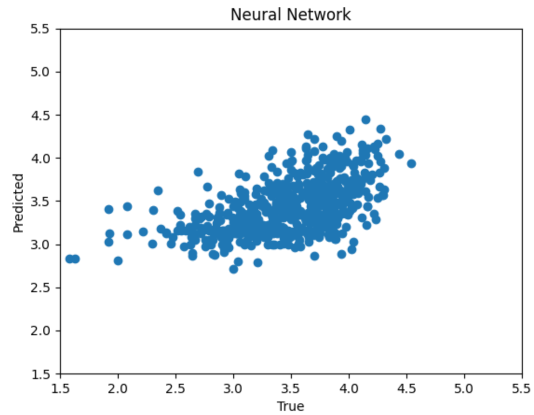
MSE: 0.24634565737450623
R2 Score: 0.19206553876723853
Adjusted combination: 1.0542801186072677



Trained Neural Network:

Configuration:
- learning_rate: 9e-4
- 3 hidden FC layers
  - nodes: 32, activation: relu
  - nodes: 32, activation: elu
  - nodes: 32, activation: tanh
- dropout layer: none
- optimizer: RMSprop, loss: mse
- min_reviews: 15

MSE: 0.1658526001776463
R2 Score: 0.2705612751605325
Adjusted combination: 0.8952913250171137



Result:

Ultimately, the trained Neural Network provided the best accuracy on validation (and test) data, and was the model that I used for all rating predictions going forward.

Sample Predictions from trained Neural Network model (First 10):

Movie: 10 Things I Hate About You (1999)
Rating: 3.5277777777777777
Predicted: 3.57578444480896

Movie: Notebook, The (2004)
Rating: 3.5657894736842106
Predicted: 3.8350586891174316

Movie: Taxi Driver (1976)
Rating: 4.105769230769231
Predicted: 4.125102996826172

Movie: On Her Majesty's Secret Service (1969)
Rating: 3.264705882352941
Predicted: 3.110677480697632

Movie: Never Been Kissed (1999)
Rating: 2.9583333333333335
Predicted: 3.242267608642578

Movie: King's Speech, The (2010)
Rating: 4.043103448275862
Predicted: 3.4516947269439697

Movie: Roger & Me (1989)
Rating: 3.838709677419355
Predicted: 3.8367197513580322

Movie: Conspiracy Theory (1997)
Rating: 2.941860465116279
Predicted: 3.3730456829071045

Movie: Clash of the Titans (2010)
Rating: 2.3076923076923075
Predicted: 3.4028358459472656

Movie: World Is Not Enough, The (1999)
Rating: 3.3333333333333335
Predicted: 3.1551103591918945

PART II: Recommendations with KNN

With the rating prediction system done, the next step was creating the first of two recommendation systems. The first and simplest recommendation system involves returning a list of movies similar to a given movie. This is achieved through KNN.

Recommendations - KNN (Process):

Objective: Take a movie as input and return K most similar movies

- Preprocess data with established techniques for rating prediction
- Fit NearestNeighbors to full dataset
    - Not training or testing at this point, so we want to see full results
- Return list of Kneighbors, excluding input

Recommendations - KNN (Sample Results):

Movie: GoldenEye (1995) Recommendations: Mission: Impossible III (2006) Man with the Golden Gun, The (1974) Casino Royale (2006) Living Daylights, The (1987) Mission: Impossible II (2000) Con Air (1997) Spy Who Loved Me, The (1977) Diamonds Are Forever (1971) Anaconda (1997) Die Another Day (2002)	Movie: Dracula: Dead and Loving It (1995) Recommendations: Bubba Ho-tep (2002) Idle Hands (1999) Gremlins (1984) Scary Movie 4 (2006) Scary Movie 3 (2003) Tales from the Crypt Presents: Bordello of Blood (1996) Gremlins 2: The New Batch (1990) Scary Movie (2000) National Lampoon's Vacation (1983)

PART III: Recommendations with Collaborative Filtering

The next rating system involves use of Collaborative Filtering, where a selection of recommendations is generated based on information gained from similar users.

Recommendations - Collaborative Filtering (Process):

Objective: take a user's review history as input, and return a list of movie recommendations based on similar users.

- Determine what "similar users" are, using some distance function.
  - For each other user, sum a similarity score for each common movie rated, with similar reviews increasing this score, and drastically different reviews reducing it.
    - Similarty_score += 0.5 - abs(target_rating - user_rating)/5
    - Reward for more common movies, Punishment for conflicting reviews
- Create a subset of all reviews from 100 most similar users
- Fit this subset to the previously trained NN model
- Make predictions for all movies in the full dataset
- Return movies with the 10 highest predicted ratings

Recommendations - Collaborative Filtering (Sample Results):

Top rated movies for user 1:	Rating:
M*A*S*H (a.k.a. MASH) (1970)	5.0
Transformers: The Movie (1986)	4.0
What About Bob? (1991)	4.0
X-Men (2000)	5.0
Shaft (2000)	4.0
Big Trouble in Little China (1986)	4.0
Shaft (1971)	5.0
Road Warrior, The (Mad Max 2)	5.0
Mad Max (1979)	5.0
Blazing Saddles (1974)	5.0
Recommendations:	**Predicted rating:**
Departed, The (2006)	4.247440338134766
Double Indemnity (1944)	4.248120307922363
Grave of the Fireflies (Hotaru no haka) (1988)	4.316830158233643
Shawshank Redemption, The (1994)	4.466507911682129
Reservoir Dogs (1992)	4.263433933258057
Prisoners (2013)	4.409069538116455
Star Wars: Episode V - The Empire Strikes Back (1980)	4.309878826141357
Fight Club (1999)	4.382254600524902
Eternal Sunshine of the Spotless Mind (2004)	4.248813629150391
Memento (2000)	4.390100002288818

Next Steps:

While I am very happy with the results for all parts, the system has plenty of room for improvements. Ideally I'd be able to use the more complete Movielens data set, with much more data available and additional parameters for much potential accuracy gains. Another time/processing limitation, much more could have been done to improve the Neural Network, I have no doubt, with additional/modified layers and more extensive cross validation. Additionally, the problem could have been converted into a Classification problem, with 10 rating classes, which may have resulted in improved accuracy.

Conclusion:

My objectives were to create an accurate movie prediction system, and separate movie recommendation systems using KNN and Collaborative Filtering. While there is still room for improvement, I achieved my goals for the project, and am very satisfied with the results.

Code:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
from sklearn.tree import DecisionTreeRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.linear_model import LinearRegression
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
tf.random.set_seed(42)


class Project:
    def load_data(self, min_reviews, split=True, target_userId=None):
        # load csv files
        movies = pd.read_csv('ml-latest-small/movies.csv')
        ratings = pd.read_csv('ml-latest-small/ratings.csv')
        tags = pd.read_csv('ml-latest-small/tags.csv')

        # for collaborative filtering, theres probably a more efficient
built-in way I could have done this, its quite slow
        if target_userId:
```

```
            # get list of movies each user has watched
            # for each other user (one not passed), compare lists, and sum
0.5-(user1rating/5 - user2rating/5) for each movie in both lists
            # create new ratings list consisting of x most similar users,
or add similar users until x number of total ratings
            user_movies =
ratings.groupby('userId')['movieId'].apply(list).to_frame().reset_index()
            target_user = user_movies[user_movies.userId == target_userId]
            user_movies = user_movies[user_movies.userId != target_userId]
            similarity_scores = {}

            '''
            # print target top 10 for reference
            target_top_movies = ratings[ratings['userId'] ==
target_userId].nlargest(10, 'movieId')
            print(f"Top rated movies for user {target_userId}:\n")
            for index, row in target_top_movies.iterrows():
                movie_id = (int)(row['movieId'])
                print(f"{movies[movies['movieId'] ==
movie_id]['title'].item()} - Rating: {row['rating']}")
            '''

            for id in user_movies.userId.unique():
                similarity_scores[id] = 0

            for movie in target_user['movieId'][0]:
                target_rating = ratings.query('userId == @target_userId
and movieId == @movie')['rating'].item()
                for index, row in user_movies.iterrows():
                    user_id = row['userId']
                    if movie in row['movieId']:
                        user_rating = ratings.query('userId == @user_id
and movieId == @movie')['rating'].item()
                        similarity_scores[user_id] += (0.5 -
(abs(target_rating - user_rating) / 5))

            similar_users = sorted(similarity_scores,
key=similarity_scores.get, reverse=True)[:100]
            similar_users.append(target_user)
            ratings = ratings[ratings['userId'].isin(similar_users)]
```

```
        # make list of tags per title
        tags = tags.drop(columns=['userId', 'timestamp'])
        tags = tags.drop_duplicates()
        tags = tags.groupby('movieId')['tag'].apply(list).to_frame()

        # average rating per movieId
        ratings = ratings.groupby('movieId').filter(lambda x: len(x) >=
min_reviews)
        ratings =
ratings.groupby('movieId')['rating'].mean().reset_index()

        # build full lens df
        movie_ratings = pd.merge(movies, ratings)
        lens = pd.merge(movie_ratings, tags, on='movieId', how='outer')
        lens = lens.dropna(subset=['title'])
        lens['tag'] = lens['tag'].fillna('Unknown')
        #lens = lens.drop(columns=['userId', 'timestamp']) # if not
averaging rating

        # split and one hot encode genres
        lens['genres'] = lens['genres'].apply(lambda x: x.split('|'))
        from sklearn.preprocessing import MultiLabelBinarizer
        mlb = MultiLabelBinarizer(sparse_output=True)
        lens = lens.join(
            pd.DataFrame.sparse.from_spmatrix(
                mlb.fit_transform(lens.pop('genres')),
                index=lens.index,
                columns=mlb.classes_).add_prefix('genre_'))
        if 'genre_(no genres listed)' in lens.columns:
            lens = lens.drop(columns=['genre_(no genres listed)'])

        # one hot encode tags, not sure if i could do both at the same
time
        lens = lens.join(
            pd.DataFrame.sparse.from_spmatrix(
                mlb.fit_transform(lens.pop('tag')),
                index=lens.index,
                columns=mlb.classes_).add_prefix('tag_'))
```

```
        if split == False:
            return lens

        titles = lens['title']
        lens = lens.drop(columns=['movieId', 'title'])

        y = lens['rating']
        X = lens.drop(columns=['rating'])
        X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=42)

        return titles, X_train, X_test, y_train, y_test

    def train_decision_tree(self):
        # Decision Tree
        print("DecisionTreeRegressor")

        best_dtr_config = []
        for min_reviews in range(0,16):
            print(f"\nMin Reviews: {min_reviews}\n")

            titles, X_train, X_test, y_train, y_test =
self.load_data(min_reviews)
            X_train_holdout, X_train_validate, y_train_holdout,
y_train_validate = train_test_split(X_train, y_train, test_size=0.2,
random_state=42)

            criterion = ['mse'] # when tuning for mse, mse criterion
always comes out ahead, so removed others to save a ton of time
            max_depth = [None, 2, 4, 6, 8, 10]
            max_leaf_nodes = [None, 5, 10, 20, 100]
            min_samples_leaf = [1, 10, 20, 40]
            min_samples_split = [2, 10, 20, 40]

            for c in criterion:
                for max_d in max_depth:
                    for max_ln in max_leaf_nodes:
                        for min_sl in min_samples_leaf:
                            for min_sp in min_samples_split:
                                dtr =
```

```
DecisionTreeRegressor(random_state=42, criterion=c, max_depth=max_d,
max_leaf_nodes=max_ln, min_samples_leaf=min_sl, min_samples_split=min_sp)
                                dtr.fit(X_train_holdout, y_train_holdout)
                                y_pred = dtr.predict(X_train_validate)

                                mse = mean_squared_error(y_train_validate,
y_pred) # should be near 0
                                r2 = r2_score(y_train_validate, y_pred) #
should be near 1

                                if not best_dtr_config or mse <
best_dtr_config[0]:
                                    best_dtr_config = [mse, r2, c, max_d,
max_ln, min_sl, min_sp, min_reviews]
                                    print(f"New best MSE:
{best_dtr_config}, Combined: {mse + (1-r2)}")
        print(best_dtr_config) # 'mse', 6, None, 10, 40, 15

        titles, X_train, X_test, y_train, y_test =
self.load_data(best_dtr_config[7])
        dtr = DecisionTreeRegressor(random_state=42,
criterion=best_dtr_config[2], max_depth=best_dtr_config[3],
max_leaf_nodes=best_dtr_config[4], min_samples_leaf=best_dtr_config[5],
min_samples_split=best_dtr_config[6])
        dtr.fit(X_train, y_train)
        y_pred = dtr.predict(X_test)

        mse = mean_squared_error(y_test, y_pred) # should be near 0
        r2 = r2_score(y_test, y_pred) # should be near 1

        print(f"Best MSE on test data with DecisionTreeRegressor: {mse}")
        print(f"Best R2 on test data with DecisionTreeRegressor: {r2}")
        print(f"Adjusted combination: {mse + (1-r2)}\n")

        plt.scatter(y_test, y_pred)
        plt.xlim(1.5,5.5)
        plt.ylim(1.5,5.5)
        plt.title('DecisionTreeRegressor')
        plt.xlabel('True')
        plt.ylabel('Predicted')
```

```
        plt.show()

        return dtr

    def get_trained_decision_tree(self):
        best_dtr_config = ['mse', 6, None, 10, 40, 12] # 15 is probably
overfit, better results on 12 (2nd best in training)

        titles, X_train, X_test, y_train, y_test =
self.load_data(best_dtr_config[5])

        dtr = DecisionTreeRegressor(random_state=42,
criterion=best_dtr_config[0], max_depth=best_dtr_config[1],
max_leaf_nodes=best_dtr_config[2], min_samples_leaf=best_dtr_config[3],
min_samples_split=best_dtr_config[4])
        dtr.fit(X_train, y_train)

        return dtr, titles, X_train, X_test, y_train, y_test

    def train_knn(self):
        # KNN
        print("KNeighborsRegressor")

        best_knnr_config = []
        for min_reviews in range(0,16):
            print(f"\nMin Reviews: {min_reviews}\n")

            titles, X_train, X_test, y_train, y_test =
self.load_data(min_reviews)
            X_train_holdout, X_train_validate, y_train_holdout,
y_train_validate = train_test_split(X_train, y_train, test_size=0.2,
random_state=42)

            n_neighbors = list(range(1,31))
            leaf_size = [1] # list(range(1,50)) # doesn't appear to have
any impact on this dataset
            p=[1,2]

            for n in n_neighbors:
                for l in leaf_size:
```

```
                    for param in p:
                        knnr = KNeighborsRegressor(n_neighbors=n,
leaf_size=l, p=param)
                        knnr.fit(X_train_holdout, y_train_holdout)
                        y_pred = knnr.predict(X_train_validate)

                        mse = mean_squared_error(y_train_validate, y_pred)
# should be near 0
                        r2 = r2_score(y_train_validate, y_pred) # should
be near 1

                        if not best_knnr_config or mse <
best_knnr_config[0]:
                            best_knnr_config = [mse, r2, n, l, param,
min_reviews]
                            print(f"New best MSE: {best_knnr_config},
Combined: {mse + (1-r2)}")
        print(best_knnr_config) # [17, 1, 1, 15]

        titles, X_train, X_test, y_train, y_test =
self.load_data(best_knnr_config[5])

        knnr = KNeighborsRegressor(n_neighbors=best_knnr_config[2],
leaf_size=best_knnr_config[3], p=best_knnr_config[4])
        knnr.fit(X_train, y_train)
        y_pred = knnr.predict(X_test)

        mse = mean_squared_error(y_test, y_pred) # should be near 0
        r2 = r2_score(y_test, y_pred) # should be near 1

        print(f"\nBest MSE on test data with KNeighborsRegressor: {mse}")
        print(f"Best R2 on test data with KNeighborsRegressor: {r2}")
        print(f"Adjusted combination: {mse + (1-r2)}\n")

        plt.scatter(y_test, y_pred)
        plt.xlim(1.5,5.5)
        plt.ylim(1.5,5.5)
        plt.title('KNeighborsRegressor')
        plt.xlabel('True')
        plt.ylabel('Predicted')
```

```
        plt.show()

    def get_trained_knn(self):
        best_knnr_config = [17, 1, 1, 11] # 15 seems overfit, much better results with 11 (2nd best in training)

        titles, X_train, X_test, y_train, y_test = self.load_data(best_knnr_config[3])

        knnr = KNeighborsRegressor(n_neighbors=best_knnr_config[0], leaf_size=best_knnr_config[1], p=best_knnr_config[2])
        knnr.fit(X_train, y_train)

        return knnr, titles, X_train, X_test, y_train, y_test

    def train_linear_regression(self):
        # Linear Regression
        print("LinearRegression")

        best_lr_config = []
        for min_reviews in range(0,16):
            print(f"\nMin Reviews: {min_reviews}")

            titles, X_train, X_test, y_train, y_test = self.load_data(min_reviews)
            X_train_holdout, X_train_validate, y_train_holdout, y_train_validate = train_test_split(X_train, y_train, test_size=0.2, random_state=42)

            normalize = [False, True]

            for n in normalize:
                lr = LinearRegression(normalize=n)
                lr.fit(X_train_holdout, y_train_holdout)
                y_pred = lr.predict(X_train_validate)

                mse = mean_squared_error(y_train_validate, y_pred) # should be near 0
                r2 = r2_score(y_train_validate, y_pred) # should be near 1
```

```
                if not best_lr_config or mse < best_lr_config[0]:
                    best_lr_config = [mse, r2, n, min_reviews]
                    print(f"New best MSE: {best_lr_config}, Combined: {mse
+ (1-r2)}")
        print(best_lr_config) # [False, 14]

        titles, X_train, X_test, y_train, y_test =
self.load_data(best_lr_config[3])

        lr = LinearRegression(normalize=best_lr_config[2])
        lr.fit(X_train, y_train)
        y_pred = lr.predict(X_test)

        mse = mean_squared_error(y_test, y_pred) # should be near 0
        r2 = r2_score(y_test, y_pred) # should be near 1

        print(f"Best MSE on test data with LinearRegression: {mse}")
        print(f"Best R2 on test data with LinearRegression: {r2}")
        print(f"Adjusted combination: {mse + (1-r2)}\n")

        plt.scatter(y_test, y_pred)
        plt.xlim(1.5,5.5)
        plt.ylim(1.5,5.5)
        plt.title('LinearRegression')
        plt.xlabel('True')
        plt.ylabel('Predicted')
        plt.show()

    def get_trained_linear_regression(self):
        best_lr_config = [False, 6] # 14 was best from training, but is
overfit. 6 second best, and better results

        titles, X_train, X_test, y_train, y_test =
self.load_data(best_lr_config[1])

        lr = LinearRegression(normalize=best_lr_config[0])
        lr.fit(X_train, y_train)

        return lr, titles, X_train, X_test, y_train, y_test
```

```
    def train_nn(self):
        # NN
        print("Neural Network\n")

        # Train Learning Rate
        titles, X_train, X_test, y_train, y_test = self.load_data(15)
        X_train_holdout, X_train_validate, y_train_holdout,
y_train_validate = train_test_split(X_train, y_train, test_size=0.2,
random_state=42)
        '''
        # coarse search
        max_count = 100
        for count in range(max_count):
            model = keras.Sequential()
            model.add(layers.Dense(64, activation='relu'))
            model.add(layers.Dense(64, activation='relu'))
            model.add(layers.Dense(1))

            lr = 10**np.random.uniform(-3, -6)
            model.compile(loss="mse",
optimizer=keras.optimizers.RMSprop(learning_rate=lr), metrics=["mse"])
            model.fit(X_train_holdout, y_train_holdout, batch_size=32,
epochs=5, validation_data=(X_train_validate, y_train_validate), verbose=0)
            scores = model.evaluate(X_train_validate, y_train_validate,
verbose=0)
            print(f"lr: {lr}, loss: {scores[0]}, val_mse: {scores[1]}")

        # fine search
        max_count = 100
        for count in range(max_count):
            model = keras.Sequential()
            model.add(layers.Dense(32, activation='relu'))
            model.add(layers.Dense(32, activation='relu'))
            model.add(layers.Dense(1))

            lr = 10**np.random.uniform(-3, -4)
            model.compile(loss="mse",
optimizer=keras.optimizers.RMSprop(learning_rate=lr), metrics=["mse"])
            model.fit(X_train_holdout, y_train_holdout, batch_size=32,
epochs=10, validation_data=(X_train_validate, y_train_validate),
```

```
verbose=0)
            scores = model.evaluate(X_train_validate, y_train_validate,
verbose=0)
            print(f"lr: {lr}, loss: {scores[0]}, val_mse: {scores[1]}")
        '''
        lr = 9e-4 # from observation

        # Train layers
        # tested 2-4 layers and various node amounts, and found the best
results with 3 layers of 32 nodes
        activation_functions = ['relu', 'elu', 'tanh']
        dropout_layer = [round(x * 0.1, 1) for x in range(0,10)] # 0.0 for
no dropout
        best_nn_model_config = []

        for a1 in activation_functions:
            for a2 in activation_functions:
                for a3 in activation_functions:
                    for d in dropout_layer:
                        model = keras.Sequential()
                        model.add(layers.Dense(32, activation=a1))
                        model.add(layers.Dense(32, activation=a2))
                        model.add(layers.Dense(32, activation=a3))
                        model.add(layers.Dropout(d))
                        model.add(layers.Dense(1))

                        model.compile(loss="mse",
optimizer=keras.optimizers.RMSprop(learning_rate=lr), metrics=["mse"])
                        model.fit(X_train_holdout, y_train_holdout,
batch_size=32, epochs=10, validation_data=(X_train_validate,
y_train_validate), verbose=0)

                        y_pred = model.predict(X_train_validate)
                        mse = mean_squared_error(y_train_validate, y_pred)

                        if not best_nn_model_config or mse <
best_nn_model_config[0]:
                            best_nn_model_config = [mse, a1, a2, a3, d]
                            print(f"New best model found:
{best_nn_model_config}")
```

```
        print(f"\nBest trained model:\nLearning rate: {lr}\nLayer1
activation: {best_nn_model_config[1]}\nLayer2 activation:
{best_nn_model_config[2]}\nLayer3 activation:
{best_nn_model_config[3]}\nDropout: {best_nn_model_config[4] if
best_nn_model_config[4] else 'None'}")

        # Test vs Test data
        model = keras.Sequential()
        model.add(layers.Dense(32, activation=best_nn_model_config[1]))
        model.add(layers.Dense(32, activation=best_nn_model_config[2]))
        model.add(layers.Dense(32, activation=best_nn_model_config[3]))
        model.add(layers.Dropout(best_nn_model_config[4]))
        model.add(layers.Dense(1))

        model.compile(loss="mse",
optimizer=keras.optimizers.RMSprop(learning_rate=lr), metrics=["mse"])
        model.fit(X_train_holdout, y_train_holdout, batch_size=32,
epochs=10, validation_data=(X_train_validate, y_train_validate),
verbose=0)

        y_pred = model.predict(X_test)
        mse = mean_squared_error(y_test, y_pred) # should be near 0
        r2 = r2_score(y_test, y_pred) # should be near 1

        print(f"Best MSE on test data with Neural Network: {mse}")
        print(f"Best R2 on test data with Neural Network: {r2}")
        print(f"Adjusted combination: {mse + (1-r2)}\n")

        plt.scatter(y_test, y_pred)
        plt.xlim(1.5,5.5)
        plt.ylim(1.5,5.5)
        plt.title('Neural Network')
        plt.xlabel('True')
        plt.ylabel('Predicted')
        plt.show()

    def get_trained_nn(self, fit=True):
        best_nn_model_config = ['elu', 'tanh', 'tanh', 0.0, 9e-4, 12] # 15
was overfit, 12 was 2nd best in training
```

```
        # Test vs Test data
        model = keras.Sequential()
        model.add(layers.Dense(32, activation=best_nn_model_config[0]))
        model.add(layers.Dense(32, activation=best_nn_model_config[1]))
        model.add(layers.Dense(32, activation=best_nn_model_config[2]))
        model.add(layers.Dropout(best_nn_model_config[3]))
        model.add(layers.Dense(1))

        model.compile(loss="mse",
optimizer=keras.optimizers.RMSprop(learning_rate=best_nn_model_config[4]),
metrics=["mse"])

        if not fit:
            return model

        titles, X_train, X_test, y_train, y_test =
self.load_data(best_nn_model_config[5])
        model.fit(X_train, y_train, batch_size=32, epochs=10, verbose=0)

        return model, titles, X_train, X_test, y_train, y_test

    def display_trained_models(self):
        # Decision Tree
        model, titles, X_train, X_test, y_train, y_test =
proj.get_trained_decision_tree()

        y_pred = model.predict(X_test)

        mse = mean_squared_error(y_test, y_pred) # should be near 0
        r2 = r2_score(y_test, y_pred) # should be near 1

        print(f"\nBest MSE on test data with DecisionTreeRegressor:
{mse}")
        print(f"Best R2 on test data with DecisionTreeRegressor: {r2}")
        print(f"Adjusted combination: {mse + (1-r2)}\n")

        plt.scatter(y_test, y_pred)
        plt.xlim(1.5,5.5)
        plt.ylim(1.5,5.5)
        plt.title('Decision Tree')
```

```
        plt.xlabel('True')
        plt.ylabel('Predicted')
        plt.show()

        # KNN
        model, titles, X_train, X_test, y_train, y_test =
proj.get_trained_knn()

        y_pred = model.predict(X_test)

        mse = mean_squared_error(y_test, y_pred) # should be near 0
        r2 = r2_score(y_test, y_pred) # should be near 1

        print(f"\nBest MSE on test data with KNeighborsRegressor: {mse}")
        print(f"Best R2 on test data with KNeighborsRegressor: {r2}")
        print(f"Adjusted combination: {mse + (1-r2)}\n")

        plt.scatter(y_test, y_pred)
        plt.xlim(1.5,5.5)
        plt.ylim(1.5,5.5)
        plt.title('KNN')
        plt.xlabel('True')
        plt.ylabel('Predicted')
        plt.show()

        # LR
        model, titles, X_train, X_test, y_train, y_test =
proj.get_trained_linear_regression()

        y_pred = model.predict(X_test)

        mse = mean_squared_error(y_test, y_pred) # should be near 0
        r2 = r2_score(y_test, y_pred) # should be near 1

        print(f"\nBest MSE on test data with LinearRegression: {mse}")
        print(f"Best R2 on test data with LinearRegression: {r2}")
        print(f"Adjusted combination: {mse + (1-r2)}\n")

        plt.scatter(y_test, y_pred)
        plt.xlim(1.5,5.5)
```

```
        plt.ylim(1.5,5.5)
        plt.title('Linear Regression')
        plt.xlabel('True')
        plt.ylabel('Predicted')
        plt.show()

        # NN
        model, titles, X_train, X_test, y_train, y_test =
proj.get_trained_nn()

        y_pred = model.predict(X_test)

        mse = mean_squared_error(y_test, y_pred) # should be near 0
        r2 = r2_score(y_test, y_pred) # should be near 1

        print(f"\nBest MSE on test data with Neural Network: {mse}")
        print(f"Best R2 on test data with Neural Network: {r2}")
        print(f"Adjusted combination: {mse + (1-r2)}\n")

        plt.scatter(y_test, y_pred)
        plt.xlim(1.5,5.5)
        plt.ylim(1.5,5.5)
        plt.title('Neural Network')
        plt.xlabel('True')
        plt.ylabel('Predicted')
        plt.show()

    def collaborative_recommendations(self, userId, recommendations=10,
min_reviews=10):
        training_data = proj.load_data(min_reviews=min_reviews,
target_userId=userId, split=False)
        training_data_y = training_data['rating']
        training_data_titles = training_data['title']
        training_data = training_data.drop(columns=['rating', 'movieId',
'title'])

        test_data = proj.load_data(min_reviews=min_reviews, split=False)
        test_data_y = test_data['rating']
        test_data_titles = test_data['title']
        test_data = test_data.drop(columns=['rating', 'movieId', 'title'])
```

```
        test_data = test_data[training_data.columns]

        model = self.get_trained_nn(fit=False)
        model.fit(training_data, training_data_y, batch_size=32,
epochs=10, verbose=0)
        y_pred = model.predict(test_data)

        ind = np.argpartition(y_pred, -recommendations,
axis=None)[-recommendations:]
        for i in ind:
            print(f"{test_data_titles[i]} - Predicted rating:
{y_pred[i].item()}")


proj = Project()

print("Rating Prediction Training")
print("==========================")

# Show training
#proj.train_decision_tree()
#proj.train_knn()
#proj.train_linear_regression()
#proj.train_nn()

# Show stats on trained models
proj.display_trained_models()

# Show a few predictions on trained NN model
print("Predictions from trained NN model")
print("=================================")
model, titles, X_train, X_test, y_train, y_test = proj.get_trained_nn()

for i in range(10):
    movie_data = X_test[i:i+1]
    movie_index = movie_data.index[0]
    movie_title = titles[movie_index]
    predicted = model.predict(movie_data)

    print(f"\nMovie: {movie_title}")
```

```python
    print(f"Rating: {y_test[movie_index]}, Predicted: {predicted[0,0]}")

# Recommendation System
print("\nMovie recommendations - KNearest")
print("================================")

data = proj.load_data(min_reviews=10, split=False)
titles = data['title']
data = data.drop(columns=['movieId', 'title', 'rating'])

from sklearn.neighbors import NearestNeighbors
recommender = NearestNeighbors()
recommender.fit(data)

for i in range(10):
    movie_data = data[i:i+1]
    movie_index = movie_data.index[0]
    movie_title = titles[movie_index]

    print(f"\nMovie: {movie_title}")

    recommendations = recommender.kneighbors(X=movie_data, n_neighbors=10,
return_distance=False)

    print("\nRecommendations:")
    for i in recommendations[0]:
        if i != movie_index:
            print(titles[i])

print("\nMovie recommendations - Collaborative Filtering")
print("===============================================\n")

proj.collaborative_recommendations(userId=1, recommendations=10)
```