A.I Technical Report

Contents

Introduction

This document will focus on the planning, creation and implementation of the KNN and Linear Regression machine learning models on a dataset obtained from Kaggle.

First, the problem intended to be solved will be introduced, then an overview of the data will be conducted followed by the methodology intended to create the models and solve the problem. Next, the creation of the models and a webapp will be shown and finally, the results of these models will be discussed.

The Problem

When going through the process of conceptualising, writing, filming and producing a movie, it can be hard to say if the movie will do well. Merely throwing money at a movie's production will not make it successful. For example, both "The 13th Warrior" and "Mortal Engines" were given budgets of around $150 million, but the former only grossed $62 million and the latter grossed a slightly better $84 million (Saha, 2023).

To assist this problem, this document will go over the creation of a simple machine learning model that will attempt to find a new movie's potential success from the movie's information. While a movie's rating does not show the monetary success of a movie, it is hoped that this tool will help to show if a new movie would be popular or not to help with the ambiguity of movie creation.

The Data

The data for this project is sourced from the metadata of the online movie database website known as IMDb (Verma, n.d.). IMDb's database contains the information of thousands of movies and allows users to see a movie's rating, cast and the people who produced said movie. It also allows users to find other movies similar to ones they already like (IMDb, n.d.). But most importantly, IMDb gives each movie a rating based on user ratings rather than ratings from professional reviews. This helps to give a better idea on what movies are popular rather than just being considered "good."

The data itself contains the following headers:

- name: The name of each movie.
- year: The year each movie was released.
- movie_rated: The age rating of each movie.
- run_length: Runtime of each movie.
- genres: The genres of each movie separated by a semicolon.
- release_date: The date each movie was released.
- rating: The rating of each movie from IMDb.
- num_raters: Number of people who rated each movie.
- num_reviews: Number of people who reviewed each movie.

(Verma, n.d.)

Methodology

For this project, the data will be loaded into the Python programming language and, through the use of the pandas library, will be loaded into tables. Once the data is loaded, it will be cleaned and normalised before then being used to create four models using K-Nearest Neighbours and linear regression, with those four models being:

1. Age Rating
2. Runtime Length
3. Genre
4. Release Month

Each model will give its predicted user ratings. Then, the average of each of these user ratings will be finally given as the predicted user rating for the new movie. After this, graphs will be created using another Python library known as matplotlib to show how the models came to the predicted user rating. After the models have been created, a website will be created to allow the inputting of movie information and the prediction of the potential user rating.

Code Overview

The Models

Setup (Cleaning)

```
5   def clean_remove_unused(dataset):
6       print("**Cleaning data**")
7       print("Dropping num_* columns...", end="")
8       try:
9           dataset = dataset.drop(["name", "year", "num_raters", "num_reviews"], axis=1)
10          #Dropping the name, year, num_raters and num_reviews columns as these will not be used to create the models.
11      except Exception as e:
12          error_exit(e)
13      print("Done.")
14      print("Dropping rows with blanks...", end="")
15      try:
16          dataset = dataset[~dataset.isin(['[]']).any(axis=1)]
17      except Exception as e:
18          error_exit(e)
19      print("Done.")
20      #Finally, save the modified dataset
21      try:
22          print("Saving modified dataset...", end="")
23          dataset.to_csv(saved_data_dir + "/Movie Dataset (Remove Unused Columns).csv", sep=",")
24          print("Done.")
25      except Exception as e:
26          print("Failed.")
27      dataset = dataset.reset_index(drop=True)
28      return dataset
```

To start, the data needs cleaning to ensure that all the data is compatible and that none of the rows or columns are missing data. The data also needs to be normalised so that a computer can understand it. However, the data is already clean as there are no missing values or corruptions. Just in case, any rows with blanks are dropped using the Pandas negative operator to select all rows that do not have blanks [line 16] (Include Help, n.d.).

Alongside this, the “name”, “year”, “num_raters” and “num_reviews” columns are dropped from the dataset because these columns have little to no correlation with the user ratings.

(Continued…)

```
def clean_normalise_boolean(dataset, column_name, split_type=None):
    print("\n**Normalising " + str(column_name) + " column**")
    #Loop 1
    try:
        #First get all unique values of the specified column.
        unique_vals = []
        print("Getting unique values of column...", end="")
        for row in dataset[column_name]:
            if split_type is not None:
                for value in row.split(split_type):
                    value = value.lower()
                    if value not in unique_vals and value != "" and value != " ":
                        unique_vals.append(value.lower())
            else:
                row = row.lower()
                if row not in unique_vals and row != "" and row != " ":
                    unique_vals.append(row.lower())
        print("Done.\nUnique values detected are:")
        for item in unique_vals:
            print("'" + item + "'")
    except Exception as e:
        error_exit(e)
    #Loop 2
    #Then, convert the rows to boolean lists.
    try:
        y = 0
        for row in dataset[column_name]:
            row = row.lower()
            bool_list = create_ditto_list(len(unique_vals), 0)
            if split_type is not None:
                for value in row.split(split_type):
                    if value in unique_vals and value != "" and value != " ":
                        bool_list[unique_vals.index(value)] = 1
            else:
                if row in unique_vals and row != "" and row != " ":
                    bool_list[unique_vals.index(row)] = 1
            #Make sure at least one of the values is True.
            if True not in bool_list:
                raise Exception("One or more rows have no True values when normalising column " + column_name)
            else:
                dataset[column_name] = dataset[column_name].astype(object)
                dataset.at[y, column_name] = bool_list
            y += 1
        print("Done.")
    except Exception as e:
        error_exit(e)
    #Then save the dataset to a new .csv file.
```

Next, the data needs to be normalised to ensure that the computer understands it.

For the genre and age rating columns, this means finding all unique values and representing them for each row as a list of Boolean values to represent the row containing that value. For example, for the genres column, there are 18 unique values, so the code creates a list of 18 Boolean values. However, the Linear Regression model specifically does not support this, so instead, all unique combinations of the genres and ages are put in a list and each row is given the index of that unique combination to use instead.

For the date column, only the month column is kept. The month is then converted to its numerical value (E.G: 6 for June). Lastly, for the runtime column, the text is dropped and the hours and minutes are converted into only minutes.

Model Creation

```
    def __init__(self, dataset, dataset_name):
        try:
            print("Setting up KNN model for " + dataset_name + ".")
            self.dataset = dataset
            self.dataset_name = dataset_name
            #Setting the selected column as X as in the x axis of a graph.
            #movie_rated is dropped otherwise it'd also be apart of the X axis which wouldn't make sense.
            print("Creating X and Y of model...", end="")
            self.x = self.dataset[self.dataset_name].drop(columns="rating")
            #Ditto, but movie rated is set to the Y axis and the selected column is dropped instead.
            self.y = self.dataset["rating"].drop(columns=self.dataset_name)
            print("Done.")
            #Setting up the training data. Here, 30% of the data becomes training data while the rest is test data.
            #Training data is like putting a label on a banana saying that it is a banana and then telling someone, "This is a banana".
            #Test data meanwhile, does not have that label. It's like asking someone, "what is this yellow object?" and hoping they say "banana".
            if type(self.x[0]) != list:
                print("Data is not a list, converting it to a list...", end="")
                for i in range(0, len(self.x)):
                    self.x[i] = [self.x[i]]
                print("Done.")
            print("Creating test and training data through splitting...", end="")
            self.x_train, self.x_test, self.y_train, self.y_test = train_test_split(self.x, self.y, random_state=11, test_size=0.30)
            #Converting the training and test data to numpy arrays that are integers.
            self.x_train = np.array(list(self.x_train), dtype=int)
            self.x_test = np.array(list(self.x_test), dtype=int)
            self.y_train = np.array(list(self.y_train), dtype=int)
            self.y_test = np.array(list(self.y_test), dtype=int)
            print("Done.")
            print(dataset_name + " x_train shape is " + str(self.x_train.shape) + "\ny_train shape is " + str(self.y_train.shape))
            print(dataset_name + " x_test shape is " + str(self.x_test.shape) + "\ny_test shape is " + str(self.y_test.shape))
        except Exception as e:
            error_exit(e)
        if type(self.dataset[self.dataset_name][0]) == list:
            self.confirm_length()
```

For both the Linear Regression and KNN models, classes were used to make it simpler to execute commands and, more specifically, access data from the models. Both models start with an initialiser which sets an x value (belonging to the class) equal to the requested dataset column (E.G: The first one is genres) while dropping the “rating” column. Conversely, the y value is set to the genres column while dropping the other column. Without dropping the other column, the model might not work. After this, the program makes sure that the contents of each row are lists because the models only accept data as lists.

Lastly, the initialiser uses the scikit-learn function “train_test_split” which splits the data into training data and test data for the model. It takes the x and y values as inputs, but also random_state (an integer that allows for splitting to be reproducible between machines) and test_size (a float that determines the proportion of the data to use as test data) (Scikit Learn, n.d. -a). In the case of test size, it is set to 0.30 (30%) because any lower would reduce the validation of the model and any higher reduces the accuracy of the model.

(Continued…)

```
81      def classifier_init(self):
82          #Creates an instance of the KNeighborsClassifier and train it with
83          #the test data.
84          #It then generates a "score" which is a percentage of how accurate the model is.
85          try:
86              print("Initiating KNN classifier...", end="")
87              self.classifier = KNeighborsClassifier()
88              print("Done.\nFitting training data...", end="")
89              self.classifier.fit(X=self.x_train, y=self.y_train)
90              print("Done.\nTesting model with test data...", end="")
91              self.predicted = self.classifier.predict(X=self.x_test)
92              self.expected = self.y_test
93              print("Done.")
94              print("Predicted is " + str(self.predicted) + "\nExpected is " + str(self.expected))
95              self.wrong = [(p, e) for (p, e) in zip(self.predicted, self.expected) if p != e]
96              print("KNN score is ", f'{self.classifier.score(self.x_test, self.y_test):.2%}')
97              print("")
98          except Exception as e:
99              error_exit(e)
```

Next, the actual model is initialised whereupon it is fitted with the x training data and y training data created with train_test_split. The x values being what we want to train the model with and the y values being the target values (Scikit Learn, n.d. -b). Then, the model is tested using the predict method which takes the x test data and returns the nearest values learned from the training data (Scikit Learn, n.d. -b) Finally, we compare the results of the test (self.predicted) and the actual values (self.expected) to test how accurate the model is.

```
101     def generate_confusion_matrix(self):
102         #Generats a grid that is a visual representation of how accurate the model.
103         try:
104             print("Creating confusion matrix...", end="")
105             self.confusion = confusion_matrix(y_true=self.expected, y_pred=self.predicted)
106             print("Done.\nConfusion matrix is:\n" + str(self.confusion) + "\nGenerating classification report...", end="")
107             self.class_report = classification_report(self.expected, self.predicted)
108             self.class_report_dict = classification_report(self.expected, self.predicted, output_dict=True)
109             #Get the classification report as a dictionary.
110             #This will be used for graphs.
111             print("Done.\nClassification report is:\n" + str(self.class_report))
112             print("")
113         except Exception as e:
114             error_exit(e)
```

With the model created and training having been done, a confusion matrix is created to help visualise the accuracy of the model by showing the model's predictions compared to the actual values (Lu, n.d.). Finally, a classification report is also generated which details the precision, recall, f1-score and accuracy of the model.

(Continued…)

```
131     def save_model(self):
132         #Saves the generated model using the "pickle" library.
133         #All models are saved to "./saved_data/models/knn/" for clarity.
134         try:
135             print("Saving " + self.dataset_name + " model using pickle...", end="")
136             model_pickle_file = open(saved_models_knn_dir + self.dataset_name + "_model", "wb")
137             pk.dump(self.classifier, model_pickle_file)
138             model_pickle_file.close()
139             print("Done.")
140         except Exception as e:
141             error_exit(e)
```

Lastly, the model is saved using the Python library “pickle” which saves the model to a binary file using serialisation (Python Docs, n.d.). This allows the model to be reloaded and used again on any machine by being de-serialised by the “pickle” library (Python Docs, n.d.).

With that, the models are completed and can be deployed.

Rendering (Graphs)

```python
def render_scatter_int(dataset, dataset_name, show_graph, x_label, title, classifier):
    #Creates a scatter graph from the entered dataset and saves it to the graphs folder in saved_data.
    print("Adding rows to the scattergraph...", end="")
    try:
        for i in range(0, len(dataset)):
            #For every row of the dataset, add the current row's rating and dataset_name values to the scattergraph.
            x = np.array(dataset[dataset_name][i])
            y = np.array(dataset["rating"][i])
            plt.scatter(x, y)
        plt.title(title) #Title the graph.
        plt.xlabel(x_label) #Set the X axis' title.
        plt.ylabel("User Rating (0-10)") #Set the Y axis' title.
        #If the user asked to see the graph, it will be shown to them, if not, it is skipped.
        if show_graph is True:
            plt.show()
        #Save the graph to the saved_data graphs directory with the name of its classifer followed by "_integer_", it's dataset name and then finally ".png".
        plt.savefig(saved_graphs_dir + classifier + "_integer_" + dataset_name + ".png")
        plt.close()
        print("Done.")
    except Exception as e:
        error_exit(e)
```

With the models having been created, the program finally moves on to rendering graphs. For the datasets, scatter graphs are used which are created with the above code. Scatter graphs are best suited to this because they can show all values on one graph clearly. This helps to estimate what the models might predicy. For this, the matplotlib library is used.

```python
def render_heatmap(dataset, dataset_name, show_graph, classifier):
    #Generates a heatmap of dataset using the seaborn library.
    try:
        print("Creating a heatmap from " + dataset_name + "...", end="")
        heatmap = seaborn.heatmap(dataset, annot=True, cmap='nipy_spectral_r')
        if show_graph is True:
            heatmap.figure.show()
        heatmap.figure.savefig(saved_graphs_dir + classifier + "_heatmap_" + dataset_name + ".png")
        plt.close()
        print("Done.")
    except Exception as e:
        error_exit(e)
```

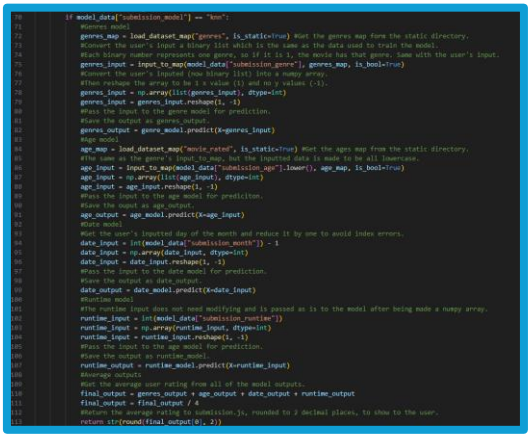
For the models themselves, heatmaps are generated to visualise the confusion matrices in a clearer format. This uses the seaborn library and shows the options that the model is most likely to select (I.E: The options the model has the most support for) using colours to represent the likelihood of selection.

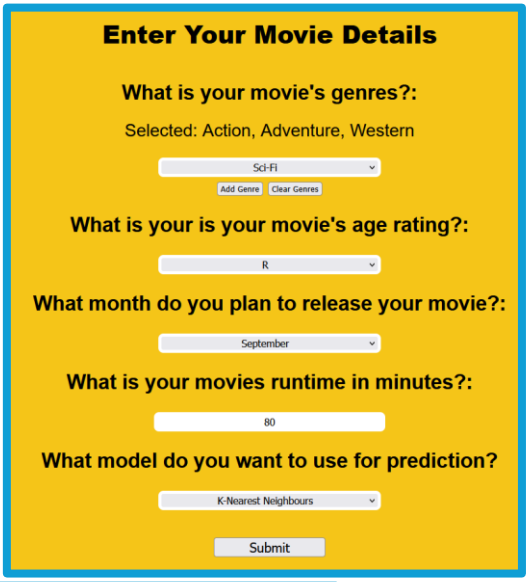
(Continued…)

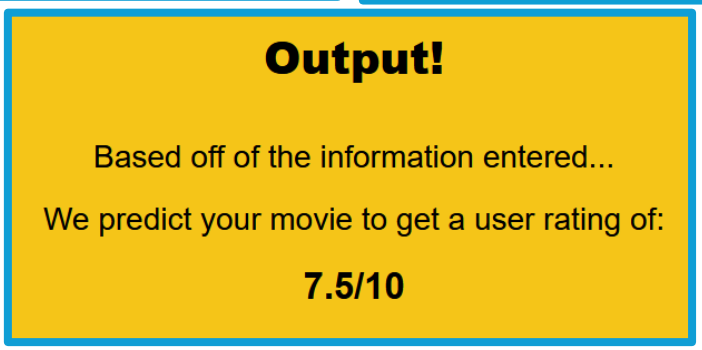
```python
def render_classification_bar(dataset_name, show_graph, classifier):
    #Generates a bar chart of the classifcation report from teh classifier.
    try:
        print("Creating a bar chart for the " + dataset_name + " model's classification report...", end="")
        for key, value in classifier.class_report_dict.items():
            if "accuracy" not in key:
                for alt_key, alt_value in value.items():
                    if "support" not in alt_key:
                        plt.bar(alt_key, alt_value, 0.2, label=alt_key.title())
            else:
                pass
            plt.legend()
            plt.xlabel("Stats")
            plt.ylabel("Score")
            if show_graph is True:
                plt.show()
            plt.savefig(saved_graphs_dir + dataset_name + "_bar_" + str(key) + ".png")
            plt.close()
        print("Done.")
    except Exception as e:
        error_exit(e)
```

Finally, the classification report of each model is rendered as a bar chart that shows the precision, recall and f1-scores of each model clearly.

The Website





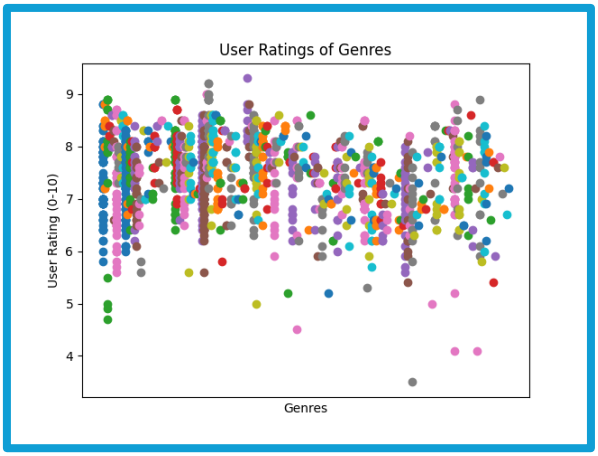


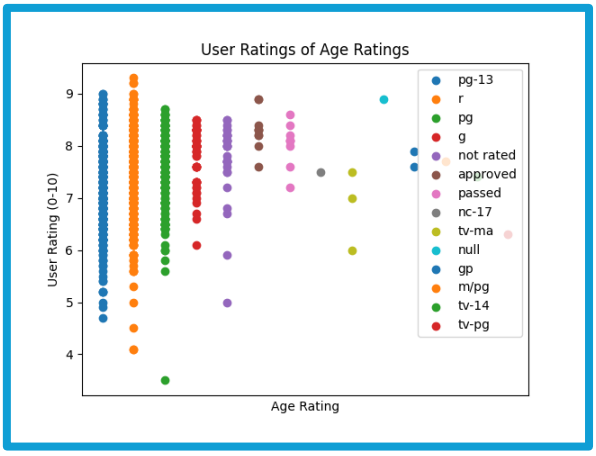
The website is a simple webapp made using the Flask library. It has three pages with one being the home page, the second being the page in which users can enter their movie's information (and select either linear regression or KNN) and then a final page which shows the user rating the models predicted.

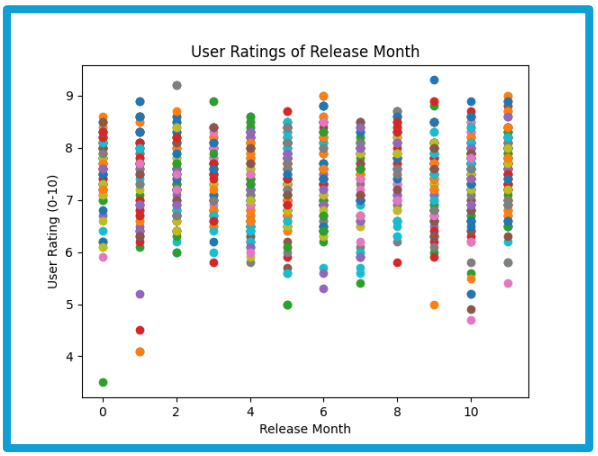
For this to work, the models are loaded from where they were saved and de-serialised by pickle (Python Docs, n.d.). Then, the data entered by the user is normalised and, using the predict method of the models, the user rating for each model is predicted. The average of these four predictions is then returned to the user as their movie's predicted rating.

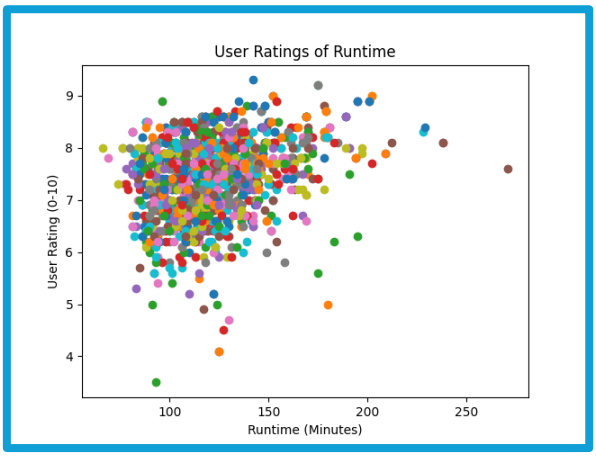
Results/Findings

The Data









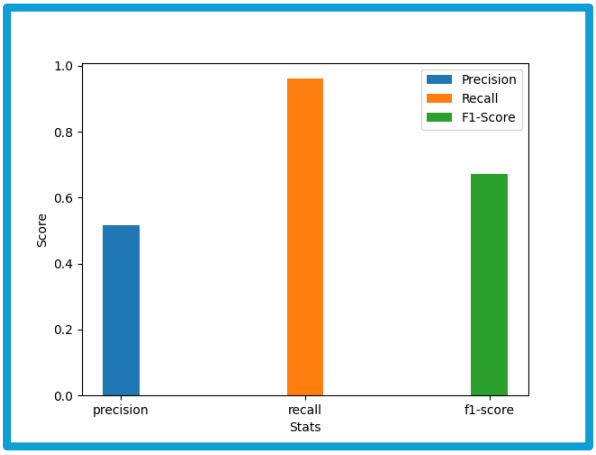
After creating the models, the program creates four graphs of the data as it appears after having been normalised. To start, there is the genres graph. This graph shows all the unique genres found by the program and what user rating they correspond with. Unfortunately, there is no clear pattern here apart from a few genres having oddly low ratings and a strange up and down “wave”.

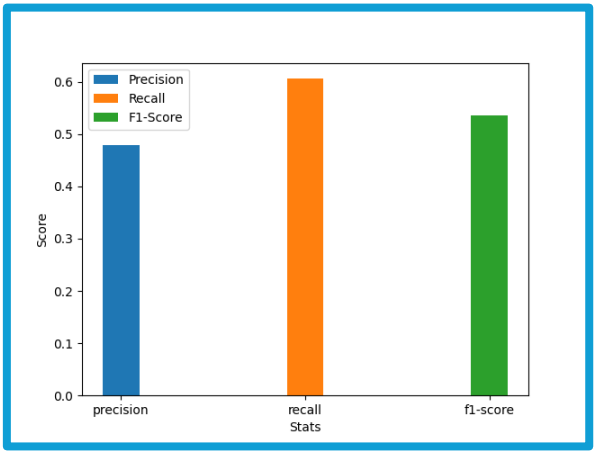
Next, there is the age ratings graph. This shows that most of the data focusses on movies rated PG-13 and R of which these ratings cover a large range of user ratings. This does show that more data is needed for the other age ranges.

Penultimately, there is the release month graph. This shows that user ratings are mostly consistent for movies throughout the year. However, there is slight decrease in the middle of the year which would suggest that movies released near holidays are rated better, but not by a significant margin.

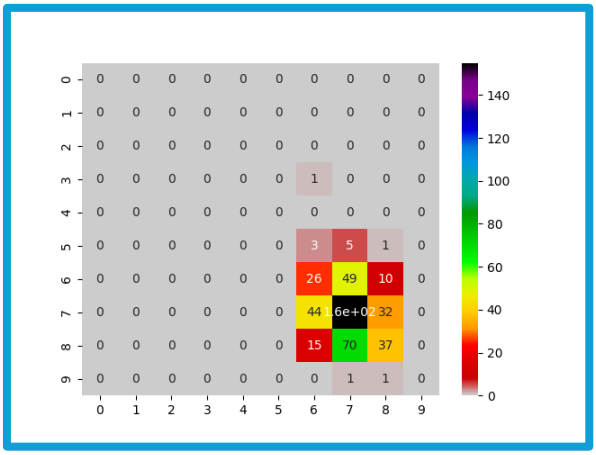
Lastly, there is the runtime graph. This graph very slightly suggests that, while movies before 150 minutes vary in rating, overall, movies get higher ratings as they get longer, before the ratings become lower again after 200 minutes. More data would be needed to prove this, however.

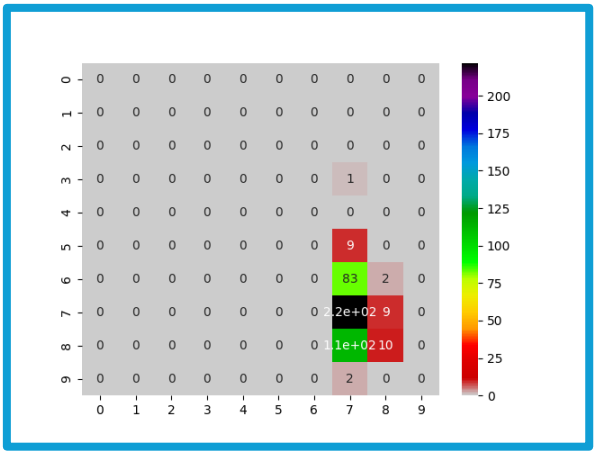
The Models

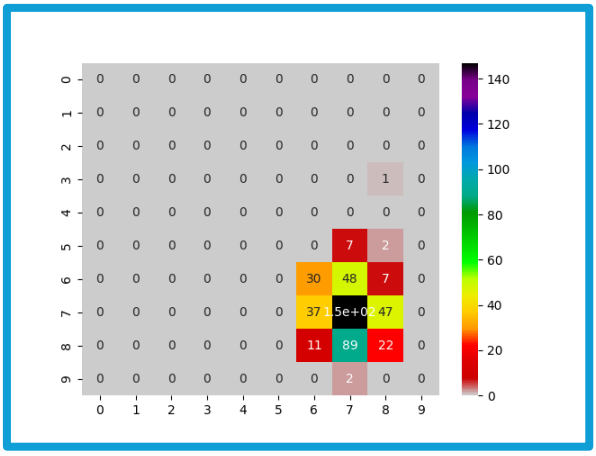


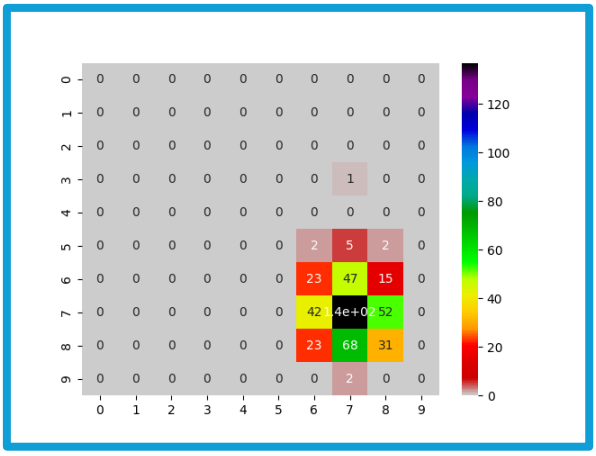


Overall, while the models work, they give a predicted user rating of around 7.0. Looking at the accuracy of the models, it would seem that the linear models have a minor trend of having better precision than the KNN models, but overall, they are consistent for recall and f1-score.









However, looking at the confusion matrices, it is possible to see where the data focused on. That is, the data only has movies with ratings between 5 and 8 and so that is all that the models can predict. The solution, then, is to get more data on more

movies from IMDb, this dataset shows that these models *could* work, but as of now, the range of prediction is too narrow.

More data does not mean the models will be more accurate, however. Currently, the models give an accuracy score between 45% and 50%, which are low, but show potential.

Conclusion

Overall, while the models work well enough, they are only useful for predicting films with average ratings (7) and not letting the user know of films that might gain lower or higher ratings simply because there is not enough data to support high rated films or low rated films. The dataset used is a good starting point for these models, but additional data from IMDb (and perhaps other websites) is critical for growing the model to be more accurate and more useful for the end user.

Also, the KNN model fits the dataset a lot more that linear regression as there is not enough of a pattern (downward or upward trend) to support it, so future iterations of the models should do away with fitting linear regression unless more of a trend can be found.

Lastly, the website, while simple, was a good base for interacting with the models and showing how they could be reloaded using pickle to allow for portability.

References

Final word count: 1999

- IMDb. (n.d.). About IMDb. IMDb. https://www.imdb.com/pressroom/about/?ref_=fea_eds_center-29_pr_related_1
- Include Help. (n.d.). Python -Tilde Sign (~) in Pandas DataFrame. Include Help.
- Python Docs. (n.d.). Pickle – Python object serialization. Python Docs. https://docs.python.org/3/library/pickle.html
- Scikit Learn. (n.d. -a). train_test_split. Scikit Learn. https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html
- Scikit Learn. (n.d. -b). KNeighborsClassifier. Scikit Learn. https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
- Saha, P. (2023, November 10th). Hollywood: These Are The Most Expensive Flops In Movie History. CEOWORLD Magazine. https://ceoworld.biz/2023/11/10/hollywood-these-are-the-most-expensive-flops-in-movie-history/
- Verma, K. (n.d.). IMDb Movies Dataset. Kaggle. https://www.kaggle.com/datasets/krishnanshverma/imdb-movies-dataset
- Lu, X. (n.d.). Model Evaluation & Evaluation Metrics. Leeds Trinity University. https://moodle.leedstrinity.net/pluginfile.php/745946/mod_folder/content/0/Week%207%20Model%20Evaluation%20%20Evaluation%20Metrics.pdf?forcedownload=1