University at Buffalo
Department of Computer Science and Engineering
School of Engineering and Applied Sciences

CSE 4/587 - Project Report

Members : Sai Venkata Aditya Arepalli, UBID: sarepall, Ub#: 50536170
Chandrahas Gurram , UBID : cgurram , Ub# : 50538624
Gowtham Bobbili, UBID: gbobbili, Ub#: 50540347

Conclusion from Phase-1 and Phase-2 analysis

Comparatively other algorithms there is slight edge that **Random Forest Algorithm** performed well

By testing numerous approaches based on the same dataset, we detect high discrepancies between various metrics ranging from accuracy, precision, recall, to f1-score. The model with Random Forest Classifier demonstrates a good compromise earning an accuracy of 65.45%, compared to the fact that it has a better performance of classifying 'Average' and 'Hit' which is also depicted by the decent precision and recall and good coverage of categories respectively. On the contrary, the Decision Tree classifier and the Gradient Boosting classifier were heavily outperformed with a lower accuracy and striking flaws on some classes, especially 'Hit' in the case of Gradient Boosting with a huge dip in recall. The K-Nearest Neighbors (KNN) model demonstrates nearly identical results as the Random Forest with an accuracy of 64.49%, reinforcing its power of great precision and recall, especially for the 'Normal' category.

While models such as Logistic Regression and Support Vector Machine (SVM) present effectiveness metrics of around 61.6% accuracy, the latter, however, is evocative of inferior metrics. Those models, SVM mostly, can be attributed to a lower recall score in the 'Hit' category, which reveals their limitation in identifying the right instances of this kind. And out of all the models KNN and Random forest is of the top performers, due to their heightened accuracy and balanced performance across classes in terms of precision and recall. Taking the spotlight in this context, the Random Forest Classifier

stands out slightly in terms of overall accuracy and stable tuning across several evaluations as it also proves to be a dependable generalizer when dealing with this dataset's data points in general.

a) Documentation on how to run the UI model :

We have used the streamlit library to create a web interface for our model. As we are working with predictions here we have no other kinds of graphical visualizations to show. Our pipeline process for the data is explained in the demo. To run the graphical interface :

1) First, run the dicmodelcode.ipynb all the way thorough so that you run all the models and save all the pickle files. The code only consists of the preprocessing part of the raw data , then running them on the models and saving them.
2) Now run the app.py file and make sure that all the saved models and the app.py are in the same folder.
3) Please make sure that the kernel of python and the versions of the libraries used when running the notebook are the same when running the app.
4) From the command line run the app.py with the command " streamlit run app.py" to open it in a browser and then you can continue playing around with the values in the UI.

1.) **Import Required Libraries**

- The necessary libraries are imported, including `streamlit` for building the web app, `pandas` for data manipulation, `pickle` for loading pre-trained models, `re` for regular expressions, and `contractions` for expanding contractions in text.

2.) **Set Page Configuration and Custom CSS**

- The page configuration is set using `st.set_page_config`, which includes the page title and layout.
- Custom CSS styles are injected into the page using `st.markdown` to match the desired color palette and styling.

3.) **Define Helper Functions**

- `remove_special_characters` and `remove_emojis` functions are defined to clean text data by removing special characters and emojis, respectively.

- `preprocess_data` function is defined to perform various data preprocessing steps, including converting data types, removing special characters and emojis, expanding contractions, and handling missing values.

4.) **Create Streamlit User Interface**

- A styled header is displayed using `st.markdown`.
- Two columns are created using `st.columns` to separate the input form and output sections.
- In the first column (`col1`), a form is created using `st.form` to collect user input, including movie details like movie name, year, genre, one-line description, stars, votes, and runtime.
- A submit button is included in the form to trigger the prediction process.

5.) **Data Preprocessing**

- If the form is submitted, the user input is collected and preprocessed using the `preprocess_data` function to prepare the data for prediction.
- The preprocessed data is stored in a pandas DataFrame `df`, and the relevant features are selected for prediction (`X`).

6.) **Prediction**

- In the second column (`col2`), various sections are created to display prediction results.
- For classification models, pre-trained models are loaded using `pickle.load`, and predictions are made using the `predict` method.
- The predictions from different classification models are collected, and the ratio of "Hit," "Average," and "Flop" predictions is displayed.
- Detailed predictions from each classification model are displayed.
- For regression models, pre-trained models are loaded, and predictions are made to estimate the movie rating.
- The average rating predicted by all regression models is calculated and displayed.
- Individual predictions from each regression model are also displayed.

7.) **Run the Streamlit App**

- The Streamlit app is run by executing the `app.py` file, which starts the development server and makes the web application accessible through a URL

b.)

In the implementation of our Streamlit application for classifying movie popularity, we used a number of machine learning algorithms so that either categorical or numeric prediction on the possible movie success may be obtained. The classifiers we have used—decision tree, gradient booster, KNN, logistic regression, random forest, and SVM—are aimed at categorizing movies into the classes ‘Hit’, ‘Average’ or ‘Flop’. This diversity guarantees the comprehensiveness of the analysis from a different angle through the use of various automated approaches.
Furthermore, over more complex evaluations, we chose to have regressors such as gradient booster, KNN, linear, random forest, and SVM to predict numeric ratings of movies. This combination of classifiers and regressors is able to provide the application with both general movie rating category and specific numeric rating so that a better user experience can be offered to a customer. Tuning these models meant that changes were made to main settings including the count of estimators in ensemble models and the complexity settings in SVM in order to stabilize the accuracy and avoid overfitting, so that we can provide the end user with valuable insights.

c.)

According to our analysis and the development of our Streamlit application for predicting audience interest in movies, people will have beforehand notions about box office sales of upcoming films. Our product is based on the historical data analyzed through machine learning models and projects which genre the movie can belong to, classifying it as 'Hit', 'Average', or 'Flop', and generating its rating. Such foresight-data assists filmmakers, marketers, and producers in making careful decisions regarding marketing approaches, distribution schedules, and, if necessary, changes to higher the film's possibility of success.

One of the ways we can improve our project is by investigating the possibility of integrating more dynamic data sources such as social media sentiment analysis, which would give us the opportunity to enhance our understanding on people’s opinions and the trends that can lead to the failure or success of a movie.Moreover, we could potentially expand the model to incorporate higher resolution demographic data so we could make predictions that are more targeted in their nature such as local or age group performance.These additional features might help our app understand the data in more detail that could make our app a super attractive tool for movie producers and distributors.

References :

- sklearn.impute.SimpleImputer — scikit-learn 1.4.1 documentation
- https://scikit-learn.org/stable/modules/generated/sklearn.compose.ColumnTransformer.html
- https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
- https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html
- https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html
- https://scikit-learn.org/stable/modules/generated/sklearn.cluster.DBSCAN.html
- https://scikit-learn.org/stable/modules/model_evaluation.html#classification-metrics
- https://scikit-learn.org/stable/modules/model_evaluation.html#regression-metrics
- https://scikit-learn.org/stable/modules/clustering.html#clustering-performance-evaluation
- https://www.kaggle.com/datasets/bharatnatrayn/movies-dataset-for-feature-extracion-prediction?select=movies.csv
- https://www.analyticsvidhya.com/blog/2022/01/text-cleaning-methods-in-nlp/
- https://www.upwork.com/resources/data-cleaning-techniques
- https://www.geeksforgeeks.org/how-to-conduct-a-two-sample-t-test-in-python/amp/
- https://www.itl.nist.gov/div898/handbook/eda/section4/eda43.htm
- https://www.stat.cmu.edu/~hseltman/309/Book/chapter4.pdf