Predictive Data Analysis

CS5812 Predictive Data Analysis (A 2022/3)

Brunel ID: 2211885

IMDB Movies Data Analysis

Data description and research question

The Internet Movie Database (IMDB) is a well-known online movie database. It offers information on over a million films, television episodes, actors, and directors. Many scholars have utilised it to analyse the popularity of movies based on user ratings or reviews.
The IMDB Movies dataset was utilised for this research from 1916 to 2016, and it provides information on movies as names, genres, ratings, and reviews etc. The purpose of this research is to predict the movie's genre based on its other features. The dataset includes 5043 rows and 28 columns.

Based on our analysis of dataset, we have come up with following research questions:

1. predict the genre of the movie given we have other details like, rating, actors ,director etc. using ML model
2. Predict the genre of the movie from poster using DL model

Data preparation and cleaning

Cleaning the data is the initial step in every data analysis task. We had to deal with missing values, duplicate rows, and inconsistent data types in this situation. I used python for my whole analysis task. To clean the data, we used the pandas package. Unneeded columns such as 'Unnamed: 0'and'movie_category' were dropped. We also deleted duplicates and discarded entries with missing data.
In the raw dataset after checking for the null values we find that there are a total of 1287 null values. We tried to come up with a way to replace NA values but there is no proper way to impute it. Therefore all records with NA values have been deleted as we have enough records, and columns like budget and gross have highest NAN values, and we can't simply impute the values for budget and gross earning of movies.

There are movies which contain more than one genre, like “fantasy|thriller|fiction”, to assign a movie a single genre out of all three , we sort the genre based on the joy factor (ref: https://stephenfollows.com/understanding-movie-genres-emotions/) . Based on the ranking we are assigning one prevalent genre to the movie.
After that we creates a new column based on imdb rating named movie_category as follows:
rating from 1-3 : flop movie
rating from 4-5 : average movie
rating from 6-8 : hit
rating from 9-10 : blockbuster

Exploratory Data Analysis

Exploratory Data Analysis, also known as EDA, examines the general properties of datasets used for our models' training. We analyse our data in this way to identify patterns, linkages, and trends. This can support additional modelling and analysis. We used EDA to obtain insights into the data after cleaning it. To understand the distribution of the data and identify any outliers, We employed several visualisation techniques such as histograms, scatter plots etc.. We also looked at the correlation between other variables to see if there were any strong links. As for this report, we are only allowed to use 12 pages, therefore, I will not be able to use all EDA charts and visuals in the report.

Observations Made:

1. The majority of the films are from the United States, and 94% of movies are in English.
2. The Shawshank Redemption is the top voted movie with IMDB rating of 9.3.
3. There is no significant relationship between the genre of the movie and the number of faces in the poster.
4. 2002 is the year with the maximum number of movies i.e. 190.
5. We also identified that the average film length was around 110 minutes.
6. We noticed that the budget of a film was related to its gross income.
7. Avatar was the biggest grossing film in the world, and it also gained the most Facebook likes.
8. Robert De Niro has the highest number of movie appearances.
9. Leonardo DiCaprio has the highest appearance in top voted movies as main actor.
10. Steven Spielberg is the most well-known filmmaker, and is on top in mostly occurring directors.
11. There are 20 unique genres after cleaning the data. Most of the movies are of the romance genre.

As part of our unsupervised learning processing we have used Principal Component Analysis (PCA) and clustering.
Few other EDA steps are as following:

```
df.shape
```

The output:

```
(3756, 27)
```

This tells us that our dataset after cleaning is divided into 3756 rows and 27 columns in total. This means we are working with 26 sets of features to work with and 1 column is for the label. For both cases the prediction column name is 'genres'. For the CNN model, we are only concerned with the prediction column. We use this feature and the rest of the features for downloading the movie poster images using the urllib library to fetch data from the internet.

```
def get_link():
    for i,item in enumerate(df['movie_title']):
        try:
            #record = df[df["movie_title"] == item]
            item = item.replace(u'\xa0', u'')
            rating = str(df.iloc[i]['imdb_score'])
            genre =  df.iloc[i]['genres']
            link = mp.get_poster(title=item)
            path = "/content/posters/"+item+"_imdb,"+rating+"_genre,"+genre+".jpg"
            print("rating:-",rating,"loc:-",i)
            print("downloading",item ,"::",path)
            print("link:--",link)
            urllib.request.urlretrieve(link,path)
        except:
            print("error occurred")
```

We use the get_link() function to achieve this. We use the movie title, rating and genre to construct the filename. The movieposter library is responsible for getting the movie according to the title of the movie to create the link. The filename and the location where the images will be stored together makes the path. The path and the link are passed as arguments to the

urlib.request.urlretrieve function. All of this is kept in a try block. If there is any error while downloading the images, we get notified that an error has occurred.

The next, step is to learn information about each of these features, also known as the metadata.:

```
df.info()
```

Output

```
RangeIndex: 3756 entries, 0 to 3755
Data columns (total 28 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   Unnamed: 0                 3756 non-null   int64
 1   director_name              3756 non-null   object
 2   num_critic_for_reviews     3756 non-null   float64
 3   duration                   3756 non-null   float64
 4   director_facebook_likes    3756 non-null   float64
 5   actor_3_facebook_likes     3756 non-null   float64
 6   actor_2_name               3756 non-null   object
 7   actor_1_facebook_likes     3756 non-null   float64
 8   gross                      3756 non-null   float64
 9   genres                     3756 non-null   object
 10  actor_1_name               3756 non-null   object
 11  movie_title                3756 non-null   object
 12  num_voted_users            3756 non-null   int64
 13  cast_total_facebook_likes  3756 non-null   int64
 14  actor_3_name               3756 non-null   object
 15  facenumber_in_poster       3756 non-null   float64
 16  plot_keywords              3756 non-null   object
 17  num_user_for_reviews       3756 non-null   float64
 18  language                   3756 non-null   object
 19  country                    3756 non-null   object
 20  content_rating             3756 non-null   object
 21  budget                     3756 non-null   float64
 22  title_year                 3756 non-null   float64
 23  actor_2_facebook_likes     3756 non-null   float64
 24  imdb_score                 3756 non-null   float64
 25  aspect_ratio               3756 non-null   float64
 26  movie_facebook_likes       3756 non-null   int64
 27  movie_category             3756 non-null   object
dtypes: float64(13), int64(4), object(11)
memory usage: 821.8+ KB
```

As we can observe there are 13 types of features with data type float64(13), 4 types with int64(4) and 11 objects. For training to take place the features with object data types like director_name or movie_titles need to be converted to float or integer data types. The genre's feature is our dependent feature. This needs to be separated from the other features. As we can observe, all the features have non-null values and all of the features have the same number of samples. This means there are no missing values.

For training the SVM all these features need to be considered, and decisions have to be made whether they are important or not. For training the CNN we only need the Genre’s and the Images. In this way the two models serve different purposes where the SVM model gives an idea of how much the genre of the model is affected by other feature sets such as the director name, the actor name or the content rating. Whereas the CNN model gives us an understanding of how the poster image of a movie is related to the genre the movie belongs to.

The data is then described

```
df.describe()
```

The output

	num_critic_for_reviews	duration	director_facebook_likes	actor_3_facebook_likes	actor_1_facebook_likes	gross
count	3756.000000	3756.000000	3756.000000	3756.000000	3756.000000	3.756000e+03
mean	167.378328	110.257987	807.336528	771.279553	7751.338658	5.261282e+07
std	123.452040	22.646717	3068.171683	1894.249869	15519.339621	7.031787e+07
min	2.000000	37.000000	0.000000	0.000000	0.000000	1.620000e+02
25%	77.000000	96.000000	11.000000	194.000000	745.000000	8.270233e+06
50%	138.500000	106.000000	64.000000	436.000000	1000.000000	3.009311e+07
75%	224.000000	120.000000	235.000000	691.000000	13000.000000	6.688194e+07
max	813.000000	330.000000	23000.000000	23000.000000	640000.000000	7.605058e+08

num_voted_users	cast_total_facebook_likes	facenumber_in_poster	num_user_for_reviews	budget	title_year	actor_2_facebook_likes	imdb_score	aspect_ratio	movie_facebook_likes
3.756000e+03	3756.000000	3756.000000	3756.000000	3.756000e+03	3756.000000	3756.000000	3756.000000	3756.000000	3756.000000
1.058267e+05	11527.101970	1.377263	336.843184	4.623685e+07	2002.976571	2021.775825	6.465282	2.111014	9353.829340
1.520354e+05	19122.176905	2.041541	411.227368	2.260103e+08	9.888108	4544.908236	1.056247	0.353068	21462.889123
9.100000e+01	0.000000	0.000000	4.000000	2.180000e+02	1927.000000	0.000000	1.600000	1.180000	0.000000
1.966700e+04	1919.750000	0.000000	110.000000	1.000000e+07	1999.000000	384.750000	5.900000	1.850000	0.000000
5.397350e+04	4059.500000	1.000000	210.000000	2.500000e+07	2004.000000	685.500000	6.600000	2.350000	227.000000
1.286020e+05	16240.000000	2.000000	398.250000	5.000000e+07	2010.000000	976.000000	7.200000	2.350000	11000.000000
1.689764e+06	656730.000000	43.000000	5060.000000	1.221550e+10	2016.000000	137000.000000	9.300000	16.000000	349000.000000

Observing the features with integers and float data types, it is observed that the features have different ranges of data. For example, we compare the mean of gross feature, and that of the duration, there is a big difference between the two values. This means the dataset values need to be normalized to ensure training takes place efficiently.

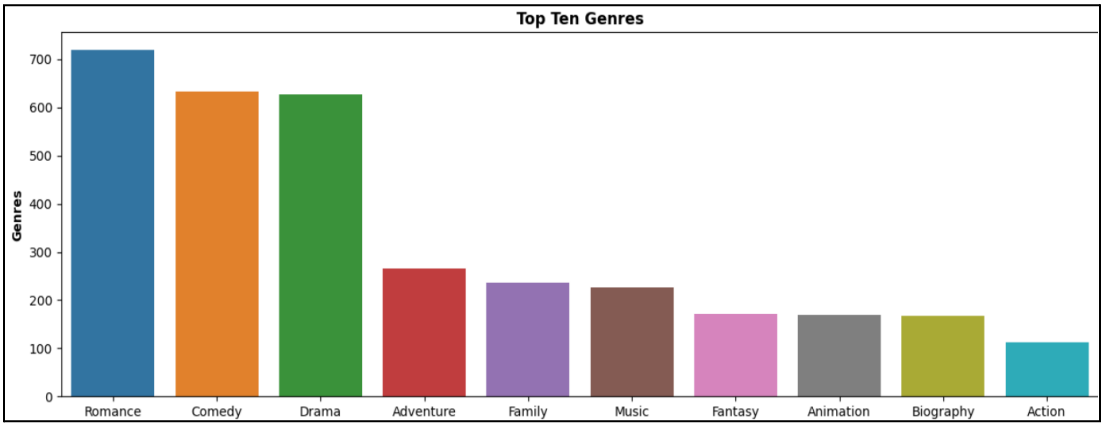
We also observe the unique values of our dependent features. For the movie genre dataset we run.

```
print("Number of unique values",df['genres'].nunique())
print("Type of unique values",df['genres'].unique())
```

The output

```
Number of unique values 20
Type of unique values ['Fantasy' 'Adventure' 'Action' 'Romance' 'Music' 'Family' 'Animation'
 'Comedy' 'Drama' 'Sci-Fi' 'Mystery' 'History' 'Crime' 'Biography' 'Sport'
 'Horror' 'Western' 'War' 'Documentary' 'Thriller']
```

As we can observe, there are 20 types of unique values in this dataset. Here is the graph of the top 10 genres.



Next step is to do label-encoding.

Here the features with object data types are label encoded.

```
from sklearn import preprocessing
label_encoder = preprocessing.LabelEncoder()
```

```
cols =
['director_name','actor_2_name','genres','language','country','content_rating','movie_category','actor_1_name','plot_keywords','actor_3_name','movie_title']
test[cols] = test[cols].apply(label_encoder.fit_transform)
```

Sklearn is used for label encoding purposes.
The dataset now looks like this:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3756 entries, 0 to 3755
Data columns (total 27 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   director_name              3756 non-null   int64
 1   num_critic_for_reviews     3756 non-null   float64
 2   duration                   3756 non-null   float64
 3   director_facebook_likes    3756 non-null   float64
 4   actor_3_facebook_likes     3756 non-null   float64
 5   actor_2_name               3756 non-null   int64
 6   actor_1_facebook_likes     3756 non-null   float64
 7   gross                      3756 non-null   float64
 8   genres                     3756 non-null   int64
 9   actor_1_name               3756 non-null   int64
 10  movie_title                3756 non-null   int64
 11  num_voted_users            3756 non-null   int64
 12  cast_total_facebook_likes  3756 non-null   int64
 13  actor_3_name               3756 non-null   int64
 14  facenumber_in_poster       3756 non-null   float64
 15  plot_keywords              3756 non-null   int64
 16  num_user_for_reviews       3756 non-null   float64
 17  language                   3756 non-null   int64
 18  country                    3756 non-null   int64
 19  content_rating             3756 non-null   int64
 20  budget                     3756 non-null   float64
 21  title_year                 3756 non-null   float64
 22  actor_2_facebook_likes     3756 non-null   float64
 23  imdb_score                 3756 non-null   float64
 24  aspect_ratio               3756 non-null   float64
 25  movie_facebook_likes       3756 non-null   int64
 26  movie_category             3756 non-null   int64
dtypes: float64(13), int64(14)
memory usage: 792.4 KB
```

After this, we normalize the dataset. This ensures all the values in the dataset are between 0 and 1. To normalize the data we use the MinMaxScaler function of sklearn.preprocessing.
During the normalization process, we drop the genres column to ensure it's not part of the normalization.
X_standard_deviation = (X - X.min(axis=0)) / (X.max(axis=0) - X.min(axis=0))
X_scaled = X_standard_deviation * (max - min) + min

MinMaxScalar normalization works by calculating the standard deviation and then using that standard deviation along with minimum and maximum values of the input.

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
scaled = scaler.fit(test.drop('genres',axis=1)).transform(test.drop('genres',axis=1))
```

After normalization process the output now looks like this

```
RangeIndex: 3756 entries, 0 to 3755
Data columns (total 26 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   director_name              3756 non-null   float64
 1   num_critic_for_reviews     3756 non-null   float64
 2   duration                   3756 non-null   float64
 3   director_facebook_likes    3756 non-null   float64
 4   actor_3_facebook_likes     3756 non-null   float64
 5   actor_2_name               3756 non-null   float64
 6   actor_1_facebook_likes     3756 non-null   float64
 7   gross                      3756 non-null   float64
 8   actor_1_name               3756 non-null   float64
 9   movie_title                3756 non-null   float64
 10  num_voted_users            3756 non-null   float64
 11  cast_total_facebook_likes  3756 non-null   float64
 12  actor_3_name               3756 non-null   float64
 13  facenumber_in_poster       3756 non-null   float64
 14  plot_keywords              3756 non-null   float64
 15  num_user_for_reviews       3756 non-null   float64
 16  language                   3756 non-null   float64
 17  country                    3756 non-null   float64
 18  content_rating             3756 non-null   float64
 19  budget                     3756 non-null   float64
 20  title_year                 3756 non-null   float64
 21  actor_2_facebook_likes     3756 non-null   float64
 22  imdb_score                 3756 non-null   float64
 23  aspect_ratio               3756 non-null   float64
 24  movie_facebook_likes       3756 non-null   float64
 25  movie_category             3756 non-null   float64
dtypes: float64(26)
memory usage: 763.1 KB
```

From this, we can observe that all the features now have the same data type which is float64. In addition to this, the genres feature is now separate from the scaled data.

The dataset now looks like this:

	director_name	num_critic_for_reviews	duration	director_facebook_likes	actor_3_facebook_likes	actor_2_name	actor_1_facebook_likes	gross	actor_1_name	movie_title	...	language	country	content_ratir
0	0.373945	0.889026	0.481229	0.000000	0.037174	0.458162	0.001563	1.000000	0.132446	0.076902	...	0.272727	0.977273	0.6363
1	0.324487	0.369914	0.450512	0.024478	0.043478	0.727938	0.062500	0.406840	0.481430	0.551450	...	0.272727	0.977273	0.6363
2	0.841375	0.739827	0.378840	0.000000	0.007000	0.820759	0.017188	0.263080	0.180799	0.661467	...	0.272727	0.954545	0.6363
3	0.151387	1.000000	0.433447	0.956522	1.000000	0.174211	0.042188	0.589253	0.939033	0.752326	...	0.272727	0.977273	0.6363
4	0.037394	0.567201	0.324232	0.020652	0.023043	0.839963	0.001000	0.096066	0.226349	0.389710	...	0.272727	0.977273	0.6363

5 rows × 26 columns

For Image Genre Classification:

```
i=0
width = 224
height = 224
X = []
y=[]
for i,item in enumerate(df['movie_title']):
        if i==2000:
         break
        #record = df[df["movie_title"] == item]
        item = item.replace(u'\xa0', u'')
        rating = str(df.iloc[i]['imdb_score'])
```

```
        genre =  df.iloc[i]['genres']
        test=item

        path = "/content/content/posters/"+item+"_imdb,"+rating+"_genre,"+genre+".jpg"
        if os.path.exists(path):

         img=image.load_img(path,target_size=(width,height,3))
         img = image.img_to_array(img)
         img = img/255.0
         y.append(df['genres'][i])

         X.append(img)

X = np.array(X)
```

For the movie poster images, each image is first resized to size 224X224. This ensures all the images that will be input to the model will be of the same size. The pixel values are divided by 255 to get a value between 0 and 1. This ensures that the pixel values will be resulting in less expensive computations. After each image is processed, they are appended to a list which is then converted to a numpy array. At the same time the images are processed the corresponding predictions are also appended to a separate list y.

After the dataset has been prepared, it's now time to split the dataset. We use the test_train_split library from sklearn.model to achieve this.

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=0)
```

As we can observe, the dataset is split into train and test. The proportion of splitting is such that 80% of the dataset is now a training dataset and 20% of the dataset is a testing dataset. The random state is set to zero. The random state determines how shuffling is applied to the data before the data is split. Hence no shuffling was done for this case.For the CNN model the dataset is split such that 10% of the dataset was assigned to test.

Defining model:

Machine Learning Prediction:

Genre prediction using SVM:

We use the SVM model from scikit-learn and the Support Vector Classification library is imported.

```
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
import time
```

```
svc=SVC(kernel="rbf",probability=False)
svc.fit(x_train, y_train)
```

Let's look at the parameters of this model

```
{'C': 1.0,
 'break_ties': False,
 'cache_size': 200,
 'class_weight': None,
 'coef0': 0.0,
 'decision_function_shape': 'ovr',
 'degree': 3,
 'gamma': 'scale',
 'kernel': 'rbf',
 'max_iter': -1,
 'probability': False,
 'random_state': None,
 'shrinking': True,
 'tol': 0.001,
 'verbose': False}
```

Here the kernel we are using initially is RBF. The RBF kernel is responsible for mapping data into high dimensional space. It does this by finding squares and dot products of all the features in the dataset. Classification is then performed using LinearSVM.
The probability parameter is set to false. If this is set to true the model would internally use 5-fold cross-validation.

Training the model:

using SVM:

```
start_time = time.time()
y_pred=svc.predict(x_test)
inference_time = time.time() - start_time
print("Inference time: {:.5f} seconds".format(inference_time))
print('Model accuracy : {0:0.3f}'. format(accuracy_score(y_test, y_pred)))
```

Time library is used to track the time of prediction.
The evaluation metric we are using is the accuracy score. This determines the proportion of correctly classified predictions made by the model out of the total number of predictions made.

Performance evaluation and comparison of methods:

Output

```
Inference time: 0.23224 seconds
Model accuracy : 0.269
```

As we can observe initially the model accuracy is 26.9% which is not very good. So we change the kernel type to linear. Linear kernel is generally used when the data we are working with is linearly separable. This kernel is used when there are a large number of features in a specific dataset. Compared to other kernels, this kernel is the fastest to train the SVM with.
Now the output becomes:

```
Inference time: 0.13890 seconds
Model accuracy : 0.294
```

From what we can observe, the accuracy of the model has increased and the inference time has decreased.
We then change the kernel type to poly. The poly kernel is responsible for representing the similarity of the training samples over polynomials of the original variables in a feature space.
The output now is:

```
Inference time: 0.14524 seconds
Model accuracy : 0.330
```

This is the highest model accuracy we could observe after hyperparameter tuning.
After applying PCA and clustering on the dataset I applied SVM but my performance decreased to 19% which is worse than what I am getting without PCA and clustering.
Output with PCA:

```
Inference time: 0.10324 seconds
Model accuracy : 0.194
```

Output with clustering:

```
Inference time: 0.09701 seconds
Model accuracy : 0.194
```

On the other hand my team member Sunny Kumar used the Random Forest model for machine learning prediction and got random forest accuracy : **0.409.**
Parameters of his model are:

parameters used:

```
{'n_estimators': 822,
 'min_samples_split': 5,
 'min_samples_leaf': 1,
 'max_features': 'sqrt',
 'max_depth': 70,
 'bootstrap': False}
```

Another team member Parth Mangukiya used KNN for ML prediction and get following results:

```
n_neighbors=5, weights = 'uniform',
algorithm='ball_tree', leaf_size=350, p=25
accuracy: 43.95
```

Popular machine learning methods for prediction problems are K-Nearest Neighbors (KNN), Support Vector Machines (SVM) and Random Forest. SVM is a strong method that can handle both linear and non-linear data by determining the best hyperplane for categorising the data(Cortes & Vapnik, 1995). KNN is a non-parametric technique that classifies new data points based on their k-nearest neighbours' majority class (Cover & Hart, 1967). Random Forest, on the other hand, is an ensemble learning algorithm that makes predictions by combining numerous decision trees(Breiman, 2001). It is well-known for its ability to deal with high-dimensional data while avoiding overfitting.
All three algorithms have demonstrated accurate predictions in a variety of fields, including image classification, text classification, and financial forecasting(Chen et al., 2012; Wang et al., 2016; Zhang et al., 2019). Furthermore, SVM may be

computationally costly and may perform poorly with unbalanced datasets, while KNN can be sensitive to the value of k and may struggle with high-dimensional data. Random Forest, on the other hand, can be difficult to understand and may not perform well with noisy data.

When utilised correctly, both SVM and Random Forest may provide accurate predictions. The benefits and limitations of prediction, however, must be recognised. Predictive models are only as good as the data on which they are trained, and the accuracy of the predictions is determined by the quality and amount of the training data. Furthermore, predictive models can be constrained by the assumptions and biases embedded into the model, and their performance might suffer if their data distribution changes. Thus, it is critical to utilise predictions as a decision-making tool, but not to depend solely on them.

Discussion of findings of ML:

From our analysis we observed that model accuracy could be made higher by changing the hyperparameters. Also, for the dataset we have, Random Forest Classifier is performing better than SVM and KNN is performing best of all with highest accuracy of 43.95 using 5 n_neighbors and weights as uniform. We can also increase the accuracy by increasing the size of our datasets i.e number of samples per category. For the SVM model, we also observed that after hyperparameter tuning, the poly kernel is an efficient kernel for multi class classification purposes using SVM as it is giving highest accuracy as compared to other kernels.

Deep Learning Prediction:

For Image Genre classification:

For the CNN model we use the MobileNet model for transfer learning purposes.

```
from keras.applications.mobilenet_v2 import MobileNetV2
from keras.layers import Dense,GlobalAveragePooling2D
from keras.applications import MobileNet
from keras.applications.mobilenet import preprocess_input

base_model=MobileNet(weights='imagenet',include_top=False) #imports the mobilenet model and discards the last 1000 neuron layer

x=base_model.output
x=GlobalAveragePooling2D()(x)
x=Dense(1024,activation='relu')(x) #we add dense layers so that the model can learn more complex functions and classify for better results.
x=Dense(1024,activation='relu')(x)
x=Dense(512,activation='relu')(x)
preds=Dense(701,activation='softmax')(x)
```

Here the weights used for the transfer learning purpose is that of the ImageNet dataset. Transfer Learning is the process where the knowledge gained from training a model for a particular task is used as the initial point for a model on the related second task. In this case the MobileNet has the pretrained weights from being trained on the imagenet dataset. This knowledge is passed to the rest of the CNN network. To achieve this the top part of the architecture is removed from MobileNet, i.e the last 1000 neuron layers.

```
for layer in model.layers[:20]:
  layer.trainable=False
for layer in model.layers[20:]:
  layer.trainable=True
```

It is also ensured that the first 20 layers are not allowed to be updated during training i.e they are frozen. This is done to make sure that overfitting is reduced or avoided during training, allowing for higher accuracy to be achieved for smaller datasets. After importing the pretrained weights, the model then undergoes global average pooling and then it is passed through Dense layers. Finally Softmax activation is used for prediction.

For Genre Image classification:

```
model.compile(optimizer='Adam',loss='sparse_categorical_crossentropy',metrics=['accuracy'])
history = model.fit(X_train,y_train,epochs=50,validation_data=(X_test,y_test),batch_size=32)
```

We first compile the model. The optimizer we used was the Adam optimizer and the loss function was the sparse_categorical _crossentropy. The metric we used was accuracy as it was with the previous case. We then train the model for 50 epochs with a batch size of 32

Performance evaluation and comparison:

```
model.evaluate(X_test, y_test)
```

Output

```
7/7 [==============================] - 0s 52ms/step - loss: 4.5560 - accuracy: 0.2700
[4.555971145629883, 0.27000001072883606]
```

As we can observe the accuracy we received is 27%.

My team member Sunny Kumar used CNN conv2D and get following results:

DL score for CNN

```
Test Score:  0.46189001202583313
Test accuracy:  0.21906355023384094
```

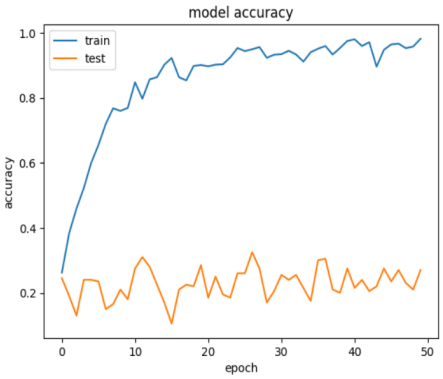
While Parth Mangukiya used CNN VGG-16 and get following results:

```
Accuracy =  0.2750677506775068
```

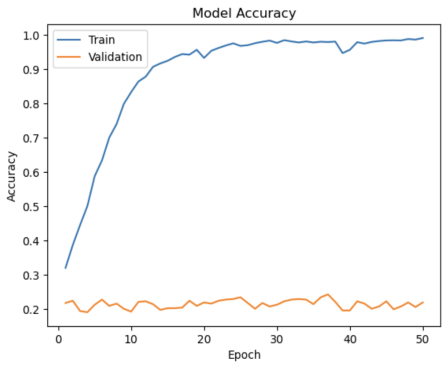
Discussion of the findings for DL:

CNN models such as VGG16, Conv2D, and MobileNetv2 employ convolutional layers to extract features from images and fully connected layers to categorise them. These models have demonstrated great accuracy in image recognition and classification tasks, but their performance is dependent on the quality and amount of training data. Furthermore, they may not be ideal for real-time processing or for limited computational resources. From our analysis, we understand that the model performance is not very strong. A possible reason could be overfitting as the model is performing well on training data. This could be due to the dataset size being too small i.e number of images per category is too small. Another reasoning could be that the

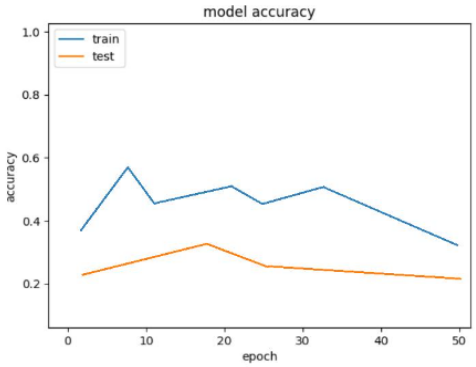
hyperparameters used can be improved. We can further try to increase the accuracy by increasing the number of epochs or by changing the hyperparameters like the optimizer. After analysing accuracy from our model Parth's model is performing little better than mine and Sunny's model on test data but performing worse on training data as compared to ours. Overall the models are not good enough to rely on as the results are very poor. The Accuracy graph for test and train data looks like:



Deepti's Model



Sunny's Model



Parth's Model

Overall, I believe the data we used is not good enough as we are all getting similar performance to our models that is very low.

Data Management Plan and Author Contribution statement:

Author Contribution statement: We used Github for code sharing.

Prediction of movie genre using ML and DL is group effort project. Below table illustrates contribution from each member.

Work	Member	Completion data
Data preparation	Sunny Kumar	28 MAR 2023
Data cleaning	Parth Mangukiya And Sunny Kumar	10 APR 2023
EDA and PCA	Sunny Kumar and Deepti	15 APR 2023
ML implementation	individual	17 APR
DL implementation	individual	17 APR

I have included the pdf of the Data Management Plan in Appendix.

While working on this analysis project, I worked as a team for the first time. Understanding each other's thinking and view point is the main task throughout the project. I learnt about different types of models and code. I learnt to use unsupervised learning methods like clustering for exploratory data analysis. On the dataset we used, most other people have done analysis of rating or gross of movies but we came up with a new research question that is different from what others have done. This way we challenged ourselves with a new and complex task. Even our models were not performing well enough but we learnt new skills and how to work as a team for the project success. All the skills I learnt during this module will help me in my future work as well as in my upcoming examination.

References:

Breiman, L. (2001). Random forests. Machine learning, 45(1), 5-32.

Chen, T., Li, M., Li, Y., Lin, M., Wang, N., Wang, M., … & Zhang, Z. (2012). Mlsp 2012 bird classification challenge. In 2012 IEEE international workshop on machine learning for signal processing (pp. 1-6). IEEE.

Cortes, C. and Vapnik, V. (1995) Support-Vector Networks. Machine Learning, 20, 273-297.

Cover, T.M. and Hart, P.E. (1967) Nearest Neighbor Pattern Classification. IEEE Transactions on Information Theory, 13, 21-27.

I have used stackoverflow, geeksforgeeks, youtube and google search to help in this project.
This page helped me in downloading posters: https://www.geeksforgeeks.org/python-imdbpy-getting-cover-url-of-the-series/

Appendix:
I have included all the python files, data csv files, and DMP file in a folder which I have uploaded in Appendix.