ALMA MATER STUDIORUM

Second Cycle Degree in Computer Science and Computer Engineering

Predicition of The Mean Rating for a Movie

Data Analytics Project Report

2022/2023

Authors:

Michelle Zanotti - 1038859
Antonio Iannotta - 1024859

1 Introduction

The goal of this analysis is the prediction of the mean rating for a certain film starting from its characteristics. The data are retrieved from MovieLens 25M dataset.
It's important to consider which are the characteristics for a film and the files in which these ones are stored:

- ***tags.csv***: this file contains tags that can be assigned to a certain film by a specific user.
- ***movies.csv***: this file contains relevant informations about every movie subjected to the survey. These informations are the movieId for a film, the title of the film and the genre (or the several genres) to which a specific film belongs to.
- ***genome_tags.csv***: this file represents the list of the tags that have been assigned to some user to some movie.
- ***genome_scores.csv***: this film represents the relevance of every tag for every film.
- ***ratings.csv***: this file contains the list of ratings for the movies.

2 Methodology

To reach the goal of our analysis, the prediction of the mean rating for a certain film starting from its characteristics, it was decided to use various techniques and approaches based on regression.
The analysis is divided into three tasks, which are mutually independent and from which it is possible to outline the approaches used:

- Traditional non-deep supervised ML techniques
- Supervised ML techniques based on neural networks
- Supervised ML technique with deep models for Tabular Data

The methodological approach followed during the development of the various tasks followed the several steps of data analytics pipeline:

- ***Data acquisition:*** data recovery to be able to analyze them and understand their domain. In this case it is flat acquisition, as the data is acquired from csv files.
- ***Data visualization:*** type of encoding that allows the visual transformation of data into forms in order to be able to explore them.
- ***Data pre-processing:*** filling missing data, smoothing noisy data, identifying and removing anomalous values, and fixing inconsistencies.
- ***Modelling:*** application of various machine learning techniques.
- ***Performance evaluation:*** evaluation of the data-analytics process

2.1 Traditional non-deep supervised ML techniques

As Traditional non-deep supervised ML techniques, the choice fell on three techniques:

- Linear regression
- KNN regressor
- Random Forest regressor

2.2 Supervised ML techniques based on neural networks

To develop an approach based on neural networks it was decided to start from the creation of a data layer, which allows the acquisition of the dataset, the creation of tensors and the application of some preprocessing techniques.
Subsequently, the architecture of the network was developed which provides a deep feedforward network (multi layer perceptions).
Finally, both the training and the evaluation process have been foreseen.

2.3 Supervised ML technique with deep models for Tabular Data: TabNet

The model chosen to complete this task is TabNet.
TabNet is a high-performance and interpretable canonical deep tabular data learning architecture, i.e. canonical DNN architecture for tabular data.
It uses sequential attention to choose which features to reason from at each decision step, enabling interpretability and more efficient learning as the learning capacity is used for the most salient features.

2.3.1 Characteristics

The main features of TabNet are:

1. Receiving raw tabular data inputs without any preprocessing and is trained using gradient descent-based optimization.

2. Using sequential attention to choose which features to reason from at each decision step. Selecting features in this way is instance-wise, e.g. it can be different for each input, and TabNet employs a single deep learning architecture for feature selection and reasoning.

3. Two kinds of interpretability:
   – local interpretability that visualizes the importance of features and how they are combined
   – global interpretability which quantifies the contribution of each feature to the trained model

4. significant performance improvements by using unsupervised pre-training to predict masked features

2.3.2 Encoder Architecture

The TabNet encoder consists of multi-steps which are sequential, passing the inputs from one step to another. Step is composed of a feature transformer, an attentive transformer and feature masking.
A split block divides the processed representation to be used by the attentive transformer of the subsequent step as well as for the overall output.
For each step, the feature selection mask provides interpretable information about the model's functionality, and the masks can be aggregated to obtain global feature important attribution.

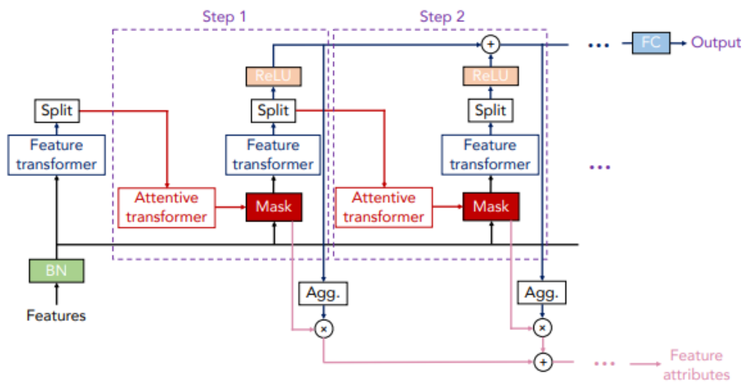

Figure 1: TabNet Encoder Architecture

Now let's see, in more detail, what happens within a single step.
First, a feature selection is made. The aim is to carry out a soft selection of the salient features by creating a mask (M[i]) using an attentive transformer. Through sparse selection of the most salient features, the learning capacity of a decision step is not wasted on irrelevant ones, and thus the model becomes more parameter efficient.
The attentive transformer allows to obtain the masks using the processed features from the preceding step; it consists of an FC[1] layer, BN[2] layer, Prior scales layer, and Sparsemax layer.
The input is passed into a fully connected layer followed by Batch normalization. It is then multiplied with the Prior scale which is a function that tells you how much you known about the features already from the previous steps and how many features have been used before in the steps. it returns P[i], the prior scale term.
– P[0] is equal to 1, so all features are equal
– P[i] is equal to $\prod_{j=1}^{i}(\gamma - M[J])$ which introduces the relaxation parameter γ. if γ is close to 1 then we can select different features at every step or if it's larger than 1 then we can reuse the same features across multiple steps.
After the prior scale, there is the sparsemax layer and here some features adding up to 0 (unused) and only the rest will add up to 1. This helps in making it an instance-wise feature selection where different features are taken at different steps.
These are then fed to the mask layer which helps to identify the selected features.

[1]Full Connected
[2]Batch Normalization

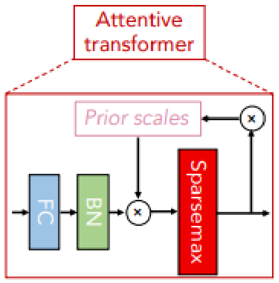

Figure 2: Attentive Transformer

Subsequently the filtered Features are processed using a feature transformer. This is composed of a concatenation of two shared layers and two decision step-dependent layers.
Each of these layers consists of an FC layer followed by a BN and gated linear unit (GLU) nonlinearity, eventually connected to a normalized residual connection with normalization.
Normalization with 0.5 helps to stabilize learning by ensuring that the variance throughout the network does not change dramatically.

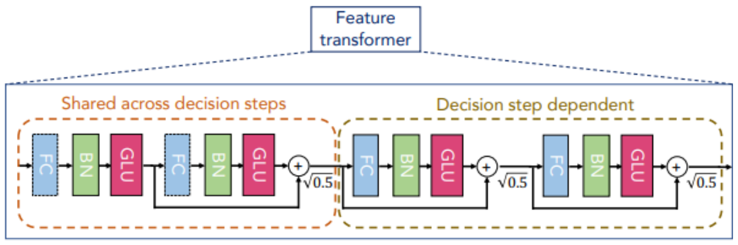

Figure 3: Features Transformer

After the operation of the feature transformer, a split takes place for the decision step output (d[i]) and information for the subsequent step (a[i]) and the overall decision embedding $\sum_{i=1}^{Nsteps} ReLU(d[i])$ (d_{out}) is constructed; and to get the output mapping the linear mapping $W_{final}d_{out}$ is applied.

Although each decision step employs non-linear processing, their outputs are combined later in a linear way. It quantifies aggregate feature importance in addition to analysis of each step.
Combining the masks at different steps requires a coefficient that can weigh the relative importance of each step in the decision.
$\eta b[i] = \sum_{c=1}^{N_d} ReLU(d_{b,c}[i])$ is used to denote the aggregate decision contribution at i^{th} decision step for the b^{th} sample.
Intuitively, if db,c[i] ¡ 0, then all features at i^{th} decision step should have 0

contribution to the overall decision. As its value increases, it plays a higher role in the overall linear combination.
$\eta b[i]$ is used to scale the decision mask at each decision step obtaining the aggregate feature importance mask.

2.3.3 Decoder Architecture

The TabNet decoder is used to perform Tabular self-supervised learning by reconstructing the tabular features from the TabNet encoded representation.
The decoder is composed of feature transformers, followed by fully-connected layers at each decision step. The outputs are summarized to obtain the reconstructed features.
What is done is the prediction of the missing columns from those present by exploiting the binary mask S, multiplied with the last FC layer and knowing that the input coming from the encoder is given by $(1 - S) \cdot \hat{f}$.

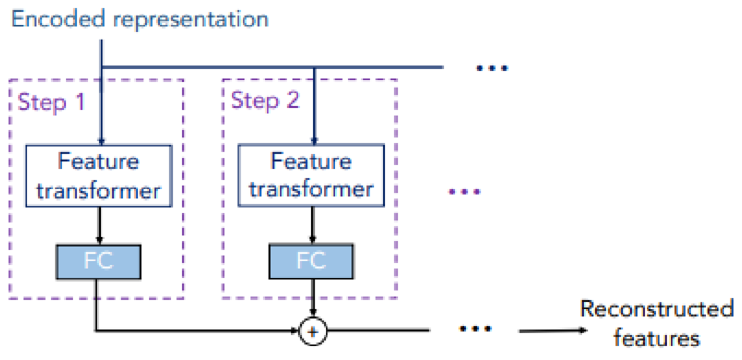

Figure 4: TabNet Decoder Architecture

3 Implementation

The first operation performed on the dataset provided was the acquisition of the five csv files of which it is composed. In order to do this, the Pandas library was exploited by creating five dataframes. Preliminary operations have been performed on these dataframes in order to obtain an aggregation of data that can be used later. First of all we worked on the dataframe containing the films with the corresponding genres. Having composed genres, multiple genres assigned to a film separated by —, it was decided to divide them by creating a column for each one. For each genre attributed to a film, a 1 is assigned for all the others, a 0. The second dataframe on which we worked was that of the ratings, obtaining the average rating for each film reviewed. The last dataframe taken into consideration is that of the genome score. A new dataframe was created consisting of a column for movie identifiers and a column for each available tag. Each line corresponds to the relevance of the tags for a particular movie. At the end, the 3 new dataframes were saved in as many csv files to be used in the subsequent phases.

3.1 Traditional non-deep supervised ML techniques

To carry out this task, having chosen to see the estimate of the average rating of a film as a prediction problem, it was decided to apply various regression techniques. Obviously there were several operations performed during the stages of implementation. These operations have comprehended the entire data pipeline. Every operation performed will be properly explained in the following:

- First of all the 3 datasets created by us were acquired by saving them in as many dataframes.
- After the creation of the dataframes, three operations about the visualization of data and correlations between them have been performed :
  - Visuaziation of the mean rating distribution
  - Visualization of *genre* - *rating* correlation
  - Visualization of *relevance* - *rating* correlation
- The three dataframes obtained from the data acquisition operated on the file created in the first point have been merged in order to obtain the final dataframe, applying a merge based on a movie id. This frame has been the starting point for the modelling and prediction operations.
- After the creation of the final dataframe an important step is the one related to the reduction of dimensionality. The usage of PCA technique has been considered to only keep the components that provided an information of at least 70% Before the application of PCA technique it was necessary to standardize the data using the *StandardScaler* of Python
- The modelling step begins with the splitting of the dataset into training and test set. On these sets the PCA technique have been applied and, this generated the dataset to be used for the training session.
- Before the prediction the hyperparameter tuning was computed using **GridSearchCV**. More specifically this tuning gave the following results:
  - For the Linear Regresison model the best hyperparameter was the one of default
  - For the KNN regressor the best hyperparameters have been the following:
    - algorithm: **ball_tree**
    - metric: **minkowski**
    - n_neighbors: **10**
  - For the RandomForest regressor the best hyperparameters have been the following:
    - max_depth: **7**
    - max_features: **sqrt**

* n_estimators: **500**
* random_state: **18**

3.2 Supervised ML techniques based on neural networks

This task required to obtain the prediction of mean rating for a given film with certain characteristics using a NN. For this task the code organization has been different from the previous one by the moment that has been choose to organize the code in several file in order to provide a cleaner view of what has been done and to make easier the debugging phase. More specifically the code has been organized in different files:

- **data_layer.py**: this file represents the data to be used as input of the neural network. More specifically in this file has been generated the tensors related to the output[3] and to the input[4]. This tensors have been create starting from the *.csv* files previously generated.
- **architecture.py**: it this file has been defined the architecture of the network. More specifically have been defined two architectures:
  - **MLP**: this architecture represents the classical Multi-Layer Perceptron in which the input is forwarded towards the output through the hidden layers. In this case have been defined two hidden layers with ReLU as activation function.
  - **MLP_Dropout**: this architecture is equal to the previous one except for the dropout, where the dropout is the way in which, at each iteration, the neural network is smaller than the previous one.
- **utils.py**: this file defines the train model and the test model. The train model is the model used during the train phase with the goal to reduce the loss through the optimization. The test model is the model used during the test phase used to evaluate the performance of the network.
- **main.py**: this file represents the file used for the training of the model, the hyperparameter tuning and the test of the model itself. More specifically has been defined a model with the following characteristics:
  - **number of epochs**: [5, 25, 50]
  - **number of layer for the hidden layers**: [8, 16, 32, 64]
  - **batch size**: [8, 16, 32]
  - **learning rate**: 0.0001

  With this parameters has been performed the training of the **MLP** model as defined in the architecture. The metrics used to evaluate the performance on the training set have been the following:

  - **loss**
  - **r2_score**

[3]the mean rating

[4]the movieId, the genres for each film and the relevance for every tag for each film

At the end of the training phase the best model resulted to be the one with the following parameters:

- **Number of epochs**: 50
- **Number of layers for the first hidden layer**: 16
- **Number of layers for the second hidden layer**: 8
- **batch size**: 8

With these parameters the results for the metrics used to evaluate the training step have been:

- **loss**: 0.03718583658337593
- **r2_score**: 0.85

- **main_dropout.py**

3.3 TabNet

The main feature provided by *tabular neural networks* is the one to receive in input a tabular input (i.e. a dataframe).
In this case after the creation of the final dataframe performed like in the other tasks have been defined all the parameters needed to create the tabular model.

- **data_config**: have been defined the target column names, categorical and numerical column names.
- **trainer_config**: have been setted all the parameters needed for the training phase. More specifically the parameters chosen for the *TabNet* model are the following:
  - **auto_lr_find = True**: this parameters allows to automatically derive the best learning rate
  - **batch_size = 1024**
  - **max_epochs = 500**
  - **min_epochs = 100**
  - **accelerator = "auto"**
- **optimizer_config**: the choise of OptimizerConfig leads to define and use several Optimizers and LearningRate Schedulers.
- **model_config**: at this point has been defined the model, in our case is **"regression"** because the task requires a prediction of the rating. It's important to notice that for this task even after several attempts to change the parameters of *TabNetModelConfig* the model that provided the best results is the one with the default value.

After the definition of the whole set of parameters needed to TabNet the network has been trained.

4 Results

In the following are reported the results of prediction with the model obtained after the hyperparameter tuning. As metrics for the results have been chosen the **mean squared error** and the **R2 score**

4.1 Traditional non-deep supervised ML techniques

Algorithm	Mse	R2
KNN regressor	0.04313929340398458	0.8127800038849485
Linear regression	0.010761137943908726	0.9532977931468383
Random Forest Regressor	0.08046306577307188	0.6507987574027296

From the previous table is possible to see how the linear regression is the best algorithm in order to both reduce the mean squared error and to increment the R2 score.

4.2 Supervised ML techniques based on neural networks

The following table reports the result obtained with the best model resulted from the training phase. It's important to remark how the loss chosen for the test and training has been the Mean Squared Error.

Dropout Loss	R2	
No	0.03749039024114609	0.84
Yes	0.06291425973176956	0.65

4.3 TabNet

The results, considering as metrics MAE[5] and MSE[6] and R2[7] of the network previously trained are the following:

MSE	MAE	R2
0.043966363347998964	0.1651230036436534	0.803270316811896

[5]mean absolute error
[6]mean squared error
[7]r2_score

References

[1] Serecan O. Arik, Tomas Pfitser, *TabNet: Attentive Interpretable Tabul*, Google Cloud AI, Sunnyvale CA.

[2] *Tabnet — Deep Learning for Tabular data: Architecture Overview*, https://vigneshwarilango.medium.com/tabnet-deep-learning-for-tabular-data-architecture-overview-448ced8f8cfc