Alma Mater Studiorum - University of Bologna
LM Informatica

Data Analytics Project

Cotugno Giosué [983620] - Pruscini Davide [1007343]

July 29, 2022

Contents

Chapter 1

Introduction

In this report we will discuss about the whole data pipeline of a project that uses different techniques of machine learning to classify some tabular data. In particular, we choose the first project proposal that aims to predict the average mark of a film from its features. The first step of the data pipeline requires to download, save and load in memory the MovieLens [1], TMDB [2] and IMDb [3] datasets. Consequently, the datasets were elaborated with the objective to generate a unified dataset that can be used as an input for some machine learning algorithms. Afterwards, during the modelling phase we built an MLP model thanks to the PyTorch framework [4]. In addition, we used other techniques like SVM, tree methods and naive bayes methods that are available into the Scikit-Learn library [5]. In order to find a good configuration during the performance analysis, it was mandatory to define a large enough hyperparameters space for all the models that we defined. Moreover, in the last phase, the cross validation technique was used to obtain more robust statistics results that has been compared between the trained models.

Chapter 2

Methodology

In the next chapter there will be the explaination of the data pipeline that the project followed. In particular, each subsection will focus on a specific task, except for the data visualization that has been used only when needed.

2.1 Data Acquisition

The used datasets are downloaded at runtime directly from the sources. These datasets come directly from MovieLens' page and provides 6 different files. More informations about the nature of the data are available here.

Dataset	**Features**
ratings.csv	userId, movieId, rating, timestamp
tags.csv	userId, movieId, tag, timestamp
movies.csv	movieId, title, genres
links.csv	movieId, imdbId, tmdbId
genome-scores.csv	movieId, tagId, relevance
genome-tags.csv	tagId, tag

Most of these datasets provides information for approximately 60000 films. The links dataset provides two identifiers that allow to collect information from the IMDB and TMDB databases. Thanks to the links' features, it has been possible to collect some more information on the running times of the films that could provide more insight into them. Talking about the ground truth of the supervisioned models, the rating mean is missing. So the target feature will be computed during the pre-process phase thanks to the ratings dataset. Further information about the features usage and the cardinality of the datasets will be discussed in the Pre-Process section.

2.2 Data Pre-process

In this section, will be discussed the pre-process phase for each of the above presented datasets. In order to achieve a major clearity, the work on each dataset will be discussed in a specific subsection where the operation computed on them will be explained. At the beginning of each subsection the cardinality will be reported.

Movies - movies.csv

Cardinality: 58098×3

Inside this dataset, the title and generes features contains multiple information. In particular, the title has been splitted in two part, where on one hand there's only the title name, and on the other, there's the year of the film production. Since the title name is a string, that doesn't add more information to a classic machine learning model, this feature has been converted into its length meanwhile the production year just constitutes a new feature. The genres feature contains a pipe separated list of a fixed possible values. Since the list is just saying if a film has a specific genre or not, to each film, all the fixed values has been added as a feature, and if a genre appears into the genres feature, that column will result into 1 that indicate the presence of that genre, 0 otherwise. At the end of this initial phase, the first sample of the movies dataset appears as shown in the Table 2.1.

movieId	**title_len**	**year**	**action**	**adventure**	**...**	**Western**	**(no genres listed)**
1	16	1995	0	1	...	0	0

Table 2.1: First sample of movis interim dataset.

In order to finish the data pre-process on this dataset, the data cleaning is required. First of all, there are some films where the year of the film production is missing. Analyzing the distribuition of these, as showed in Figure 2.1, it is possible to see that the distribuition is right skewed, so the missing values can be filled with the **median**, as suggested during the lectures.

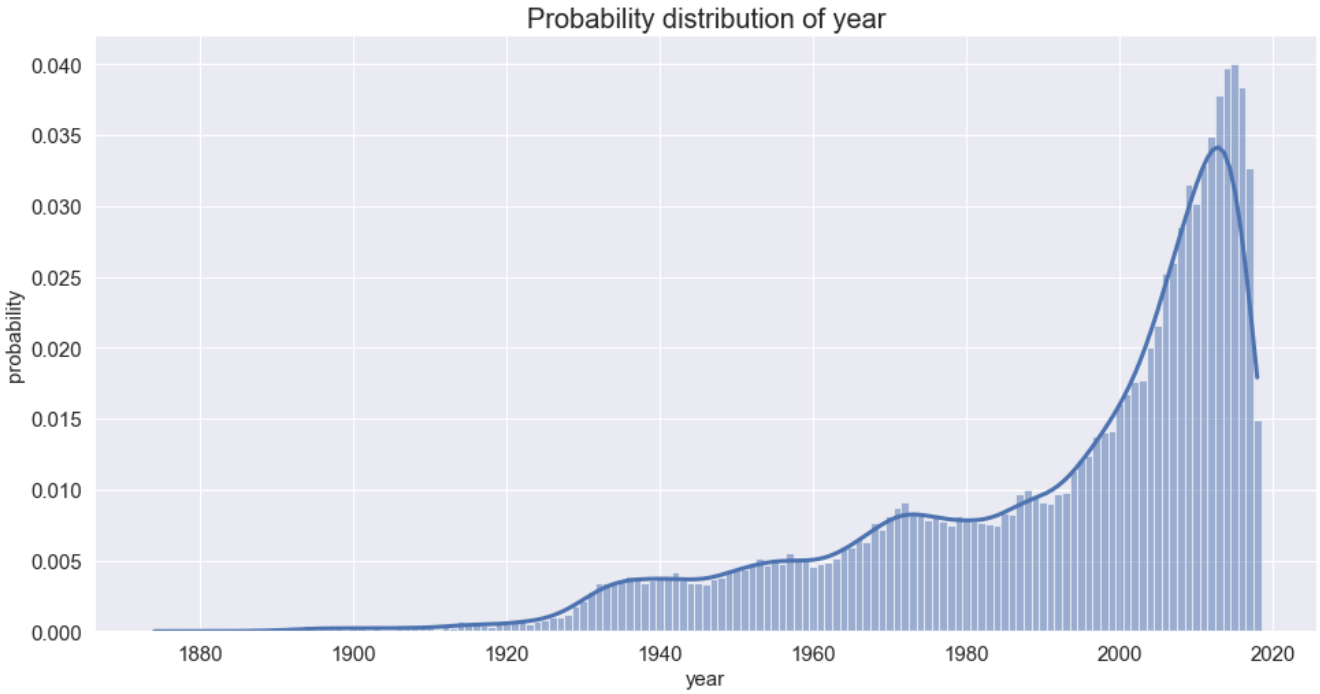


Figure 2.1: Right skewed probability distribution of the year feature.

Since the feature (no genres listed) and the films that has no genres provide no information about the film, they will be removed obtaining a final cardinality of 53832×22.

Tags - tags.csv

Cardinality: 1108997×4

In order to use the relevant data from this dataset the timestamp and userId features have been deleted because they don't explain anything more about the films. The information provided by each sample in the dataset are not very meaningful, for this reason it was decided to count the number of tags associated with each film, in order to understand how much interaction that film generated. When a film doesn't have any related tag, the tag_count can be setted to 0 because no users were interested on that film. After these operations the cardinality was reduced to 45981×2.

Ratings - ratings.csv

Cardinality: 27753443×2

The features in this dataset were necessary to find the target column. In order to compute it, the average of each films' ratings has been calculated. In addition, the number of the rating on each film has been counted. However, the task of this project is the classification, for this reason the rating_mean has been discretized in 10 bins, where each of them covers a rating range of 0.45 as showed in Figure 2.2.

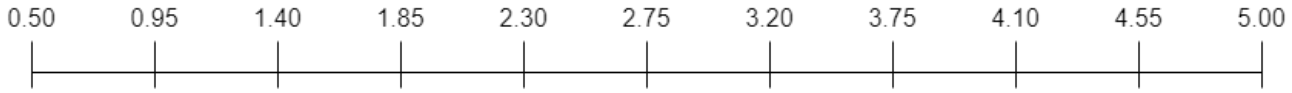


Figure 2.2: Discretization of the rating_mean feature.

At the end, the final cardinality is 53889×4.

Genome - genome-scores.csv and genome-tags.csv

Cardinality scores: 14862528×3
Cardinality tags: 1128×2

Talking about these two datasets, the merge operation over the tagId was needed because it was interesting to associate the tagId with its string name. After this union, on each sample there is the correspondence between a movieId, the tag name and the relevance of that tag. The final step consist on relate every film to the relevance of each tag, using the pivot function, as shown in Table 2.2.

movieId	007	18th century	action	absurd	...	addiction	adventure
1	0.02900	0.05425	0.66825	0.09725	...	0.07475	0.90700

Table 2.2: First sample of genome interim dataset.

The final dataset named genome.csv has the cardinality equal to 13176×1129.

Links - links.csv

Cardinality: 58098×3

On this dataset no pre-processing was needed because for each movie it contains two identifier that provide a link to external sources. The interesting one is tmdbId because it has been used to retrieve information from the TMDB database. Using the TMDB Api it has been possibile to build a new dataset that contains some additional features:

- budget
- revenue
- adult
- runtime

In order to avoid the expensive operation of the Api calls, the resulted dataset has been saved in a GitHub release. Due to the insufficient amount of valid samples, the IMDb title-basics.csv dataset was downloaded to try to fill in all missing values. After a brief exploration and analysis it has been possible to see that only the runtime feature has an enough amount of samples. Since there continue to be some missing values, the distribution of this feature was analyzed in Figure 2.3. It is possible to see that the distribuition is left skewed, so the missing values can be filled with the **median**.

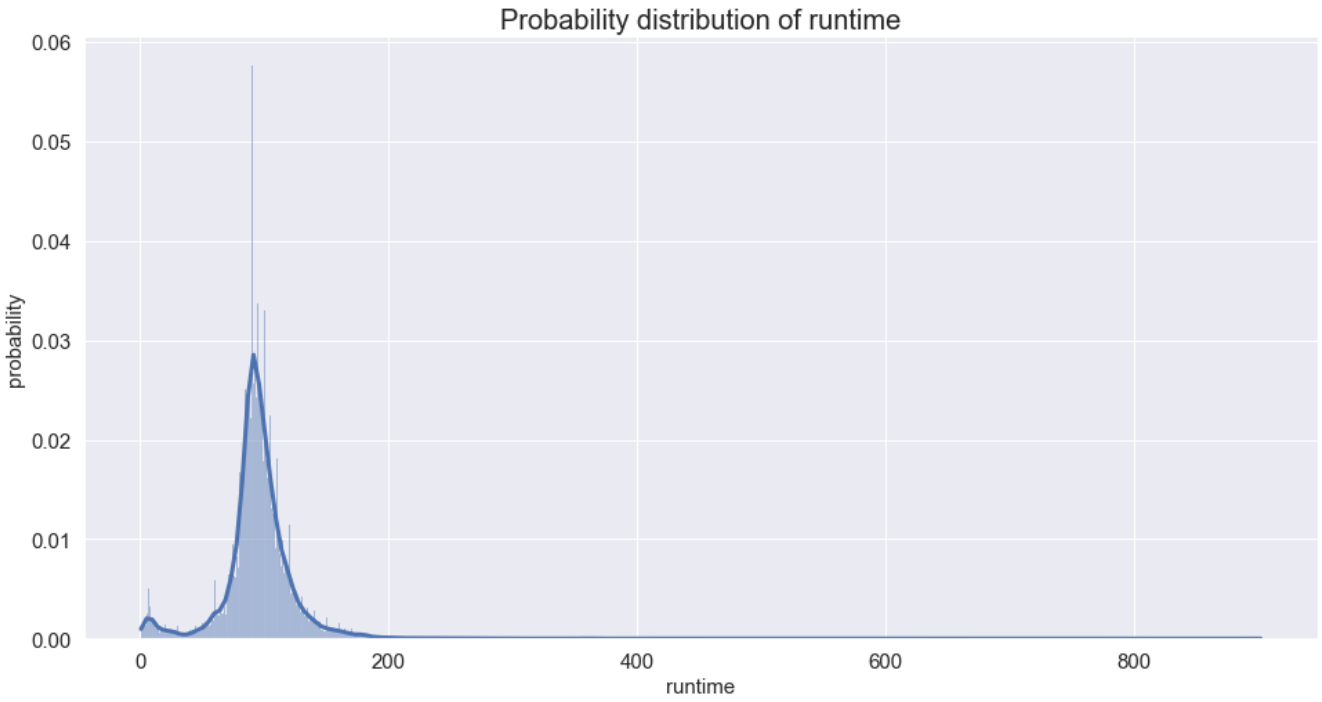


Figure 2.3: Left skewed probability distribution of the runtime feature.

The final cardinality of the acquired external resources is 58098×2.

Final - final.parquet

In this section it will be showed how the cardinality of the used dataset has been obtained, starting from all the previously processed datasets. In particular, the focus will be on the number of samples, because the number of columns will increase by the addition of the features coming from all the other datasets. Since for a classification task is interesting to know the information about a sample, the starting dataset will be the movies and then all the others will be merged, trying to preserve the maximum number of samples. The first merge has been between movies and ratings because if a film doesn't have the target feature there is no point to consider it. In addition, all films with no rating mean were discarded, resulting

in a cardinality of 50157×24. From this point on, all the merges will be performed using the previously calculated dataset, so only the new dataset will be specified. The second merge considers tags dataset, where there was an important number of samples that hasn't a tag count. In this case, these samples were not discarded because they were considered as a film that generated no users interaction, so all missing values were filled with 0, resulting in a cardinality of 50157×25. The third merge introduces external resources where for each film there is a related sample, so the cardinality depends from the previous one, resulting 50157×26. The final merge introduces the genome dataset, where there is a large gap between the cardinality of this dataset and the previous one. The choice for this merge was to discard all data that don't match with the genome dataset because it provides more interesting characteristics than the other datasets could provide. At the end, the cardinality of the final dataset is 13147×1154.

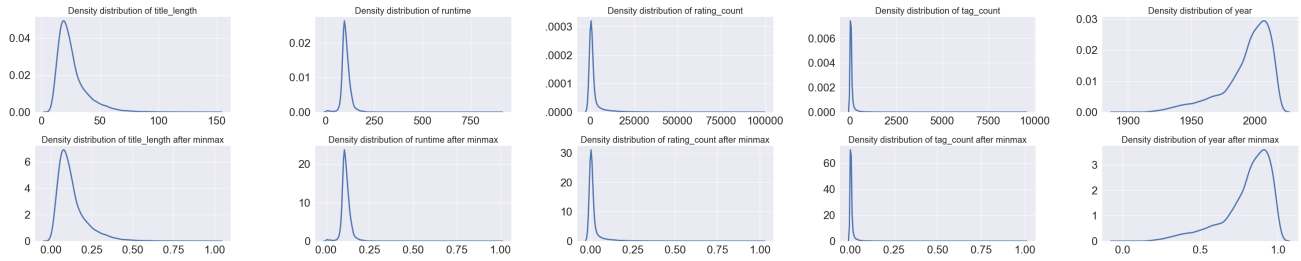
Data Transformation

After further exploration, it becomes apparent that some features did not belong to a well-defined range. For this reason, it was useful to apply certain transformation techniques such as min-max scaling and normalization. In order to apply some of these transformations, taking into account also the balancing task, the split of the dataset is needed because all these operations entail working on the training set. So, in this section, the dataset will be split with the ratios shown in Table 2.3.

Train set size	**Validation set size**	**Test set size**
72%	8%	20%

Table 2.3: Train/Validation/Test set ratios.

The features that have been analyzed are: title_length, runtime, rating_count, tag_count, year. It has been seen as the distribuition of these features are not Gaussian and for this reason the min-max scaling has been applied.



(a) Title length (b) Runtime (c) Rating count (d) Tag count (e) Year

Figure 2.4: Comparision of feature distribuition.

In order to have a greater reliability on the techniques to apply, a test function was created. This function uses the default version of the scikit models used in this project, which are: RandomForest, DecisionTree, GaussianNB, QDA and SVM. Only the scikit models were used due to their ease of use, which is why the Neural Network does not appear in these tests. From the outcomes, some graphs were produced to better interpret the results. Thanks to this function, 3 configuration has tried:

- Minmax scaling with normalization of all non categorical features
- Minmax scaling

- Normalization of all non categorical features

After the tests it turned out that the normalization technique in addition to minmax scaling did not provide any improvement, as shown in Figure 2.5. In the last configuration, only normalization was applied. The results obtained are slightly lower than those obtained with the previous technique, which is why only min-max scaling was applied to the final dataset.

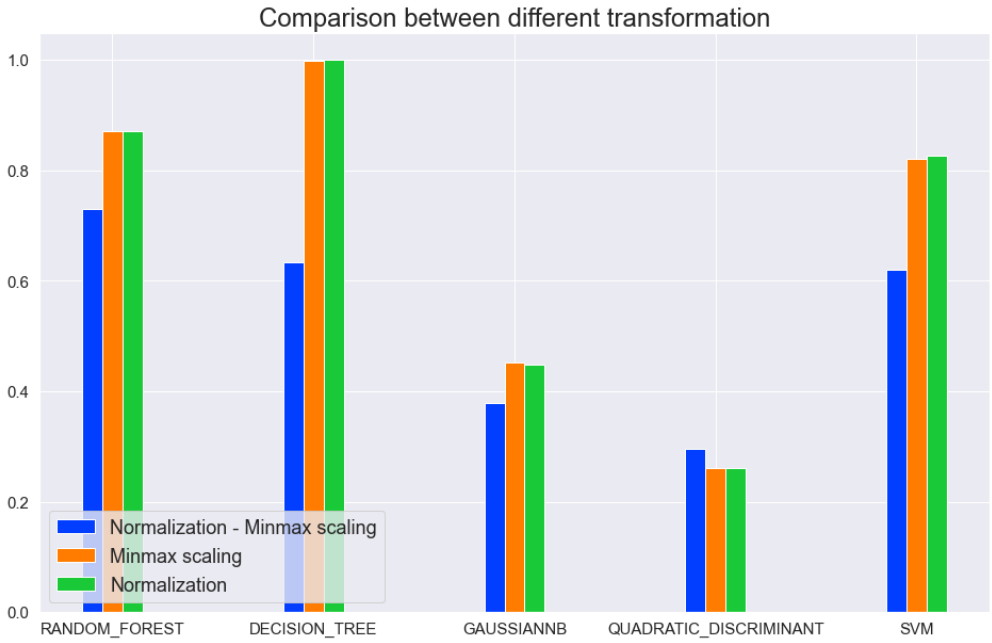


Figure 2.5: Comparision between normalization and minmax scaling

Data balancing

The dataset from all previous stages is strongly unbalanced, as shown in Figure 2.6.

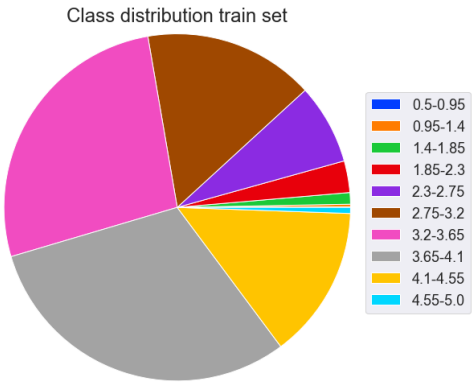


Figure 2.6: Initial class distribuition over the samples.

It is therefore necessary to apply some balancing technique in order to obtain a similar number of samples for each class. Since the imbalanced-learn [6] library provides multiple techniques, it was necessary

to carry out tests to see which of these gave the best results. To do so the function mentioned in the previous paragraph was used with the following under/over sampler techniques:

- SMOTE
- SMOTETomek
- SMOTEENN
- RandomOverSampler
- SMOTE with threshold

From the outcomes, some graphs were produced to better interpret the results.

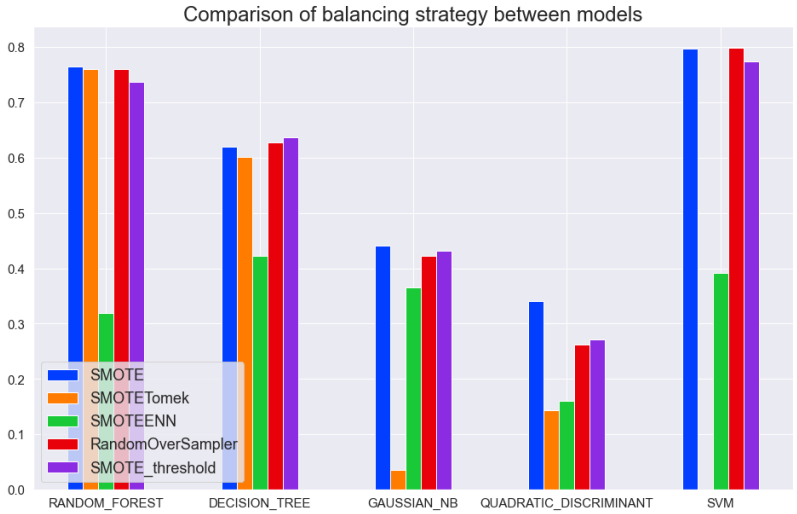


Figure 2.7: Dataset balancing comparison.

As showed in Figure 2.7 the SMOTE technique gave slightly better results than the other balancing methods. For this reason it has been applied to the dataset and the dsitribution of samples per class is plotted in Figure 2.8.

Due to the different behaviour of the neural network, it was decided to use one of the methods provided by the PyTorch framework to handle data imbalance:

- *WeightedRandomSampler*, gives a weight to each sample considering the frequency of the class it belongs to

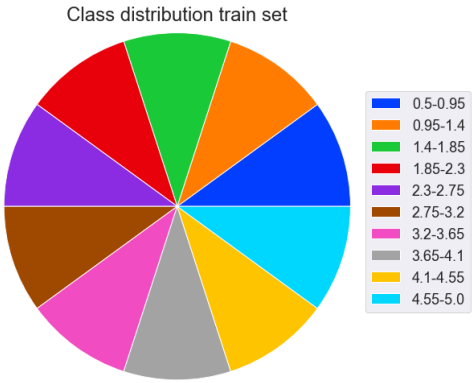


Figure 2.8: Class distribution with SMOTE.

2.3 Modeling

In order to train the proposed models with enhanced reliability the internal-external cross-validation is used. In particular, the dataset has been splitted into the train and test sets with the 5-fold cross-validation, then the same technique has been applied, but this time the purpose was to perform hyper-parameters optimization. Moreover, each cross-validator takes the classes distribution into account. The models specified in the assignments were used with the following hyperparameters:

- DecisionTree
  - *criterion*, the function to measure the quality of a split: {gini, entropy}
  - *max_depth*: {5, 10, 15}
- RandomForest
  - *n_estimator*, the number of trees in the forest: {100, 300, 500, 700, 900}
  - *max_features*, the number of features to consider when looking for the best split: {sqrt, log2}
- GaussianNB
  - *var_smoothing*: logspace(0, -9, num=100)
- Quadratic Discriminant Analysis (QDA)
  - *reg_param*: $\{x = 10^z, z \in \mathbb{Z} | -5 \leq z \leq -1\}$
  - *tol*: $\{x = 10^z, z \in \mathbb{Z} | -4 \leq z \leq -1\}$
- SVM
  - *kernel*: {rbf, poly, sigmoid}
  - *C*: $\{x = 10^z, z \in \mathbb{Z} | -1 \leq z \leq 2\}$
  - *gamma*: $\{x = 10^z, z \in \mathbb{Z} | -2 \leq z \leq -1\}$

The previously defined hyperparameters were chosen from the documentation of the various models. In particular, for categorical parameters, a sub-set of those proposed in the model documentation was chosen, while for numerical parameters, a range was created from the default value.

- Neural Network
  - *num_epochs*: {200}
  - *starting_lr*: $\{10^{-3}\}$
  - *batch_size*: {128, 256}
  - *optim*: {Adam, SGD}
  - *momentum*, used only with SGD: {0.6, 0.9}
  - *weight_decay*: $\{10^{-5}, 10^{-7}\}$

With regard to the Neural Network, it was necessary to define also the architecture:

- *input_act*: {LeakyReLU}
- *hidden_act*: {LeakyReLU}
- *hidden_size*: {512}
- *num_hidden_layers*: {3, 5}
- *dropout*: {0.2, 0.3, 0.5}
- *batch_norm*: {False, True}
- *loss_fn*: {CrossEntropy}
- *output_fn*: not used because CrossEntropy includes the SoftMax

After optimising the hyperparameters and finding the best configuration, it was decided to study the behaviour of the model when varying the *batch_size* and the use of *batch_norm*. In particular, the new tested configurations are:

- *batch_size*: {16, 32, 64, 512, 2048, 16384}
- *batch_norm*: {False, True}

2.4 Performance Analysis

In the case of study, since the dataset is strongly unbalanced, the validation f1-score is the chosen score with which the ranking of models and hyperparameters combinations is performed. At the end of each training fold on a specific configuration, the measured values are *loss*, *accuracy* and *f1-score*. Saving these values for each fold, the mean and the confidence interval could be calculated for the chosen metric.

Chapter 3

Implementation

The implementation phase includes the use of some library that has been seen during the lectures. In particular, the libraries related to data analytics were pandas, numpy, imbalance-learn, scikit-learn and torch. In order to do the data visualization stage matplotlib and seaborn were used. It was necessary to install the CUDA platform to take full advantage of the GPU, which is only supported by the neural network. The structure of the project tries to follow the cookiecutter template and is therefore defined as below:

- *data*, is created during the first execution, contains the data processed and to be processed
- *noteboooks*, contains notebooks explaining the implementation of certain project parts
- *reports*, contains LaTeX source files and figures
- *src*, contains project python files
- *.env*, useful to specify env variables
- *main.py*, entry point for the project, contains the definition of argparse to specify which phase to execute
- *requirements.txt*, specify the libraries that the project requires

Since the notebooks were written to show and explain different parts of the code, each of them will be introduced with the relative choices made to perform that operation. The first item of each sub-entry has a link to GitHub resource.

3.1 Preprocessing

It was decided to download the datasets at runtime if they were not present within the data folder, so as not to neglect the acquisition and preprocessing phase within the pipeline. To reduce the memory size of each dataset, the correct type had to be specified for each feature. In addition, these datasets were saved in .parquet format to optimise the performance of operations. To increase the readability of the code, Method Chaining was used for each DataFrame.

- *1.0-raw-data-exploration*

- *1.1-external-data-exploration*
- *1.2-add-genome-data*
- *1.3-processed-data-storage*
- *1.4-data-transformation-evaluation*
- *1.5-imbalance-evaluation*

3.2 Modeling

All sklearn models can use the processed dataset directly, unlike mlp, which must use an appropriate class to represent the dataset. The training phase includes the hyperparameters optimization, computed in a differet way beetwen both typology model. For the sklearn models, GridSearchCV was used, which allows cross-validation and simple selection of the best model. Since the PyTorch model doesn't support that class, it was needed to use itertools. It provides only a cartesian product of all configuration values and so it was necessary to define by hand cross validation and other flow controls. It is also possible to choose in which mode to execute the project, following this argparse definition:

```
main.py model [--easy | --best]
```

The next two notebooks show a demo for each of the models used. Unlike the implementation in the project files, these do not save output metrics, but merely display them.

- *2.0-sklearn-models*
- *2.1-mlp-model*

It is therefore necessary to explain how the saving of output metrics is handled. Saving output metrics involves storing the train, validation and test scores within a DataFrame where the configuration and fold to which they belong is specified.

3.3 Performance analysis

The following notebook aims to read and analyse all the csv files containing the output metrics of the various models. In particular, useful functions are implemented to find a specific configuration, find the configuration with the best f1-score and print a summary with the metrics of the best configurations.

- *3.0-performance-analysis*

Chapter 4

Results

This chapter will discuss the results of the training, validation and testing. In the first section the configurations with the best results will be shown with their average scores obtained during training and validation. Subsequently, the results obtained from the best configurations will be shown in the Testing section.

4.1 Validation

The hyperparameters of the configurations with the best validation results are shown in Tables 4.1 and 4.2. Specifically, the first table shows the scikit-learn models with their hyperparameter values. The second table shows the best configuration of the Neural Network showing only the hyperparameters that could vary in their domain. Table 4.3 shows the f1-score obtained by each model during the training and validation phase.

Model	**Best configuration**
RandomForestClassifier	n_estimators: 700 max_features: 'sqrt' max_depth: 4
DecisionTreeClassifier	criterion: 'entropy' max_depth: 15
GaussianNB	var_smoothing: 8.111308307896872e-07
QDA	reg_param: 1e-3 tol: 1e-4
SVM	C: 100 gamma: 1e-2 kernel: 'rbf'

Table 4.1: Best configurations for scikit-learn models.

Model	Best configuration
MovieNet (MLP)	n_hidden_layers: 3 dropout: 0.2 batch_norm: False batch_size: 128 optim: optimizer.Adam weight_decay: 1e-7

Table 4.2: Best configuration for PyTorch model.

Model	Training	Validation
	f1-score (%)	
RandomForestClassifier	78.9 ±0.2	78.1 ±0.2
DecisionTreeClassifier	99.8 ±0.01	86.6 ±0.1
GaussianNB	72.1 ±0.3	70.7 ±0.3
QDA	98.7 ±0.03	89.8 ±0.07
SVM	99.9 ±0.001	96.1 ±0.09
MovieNet (MLP)	87.5 ±0.1	79.5 ±0.08

Table 4.3: Training and validation results of each model with the best configurations.

4.2 Testing

The test phase is necessary to evaluate the performance of the model using samples that have never been taken. It can be seen in Table 4.4 how SVM and MovieNet achieved the highest metrics with the lowest losses. Instead, probabilistic models are the ones that have obtained worse metrics.

Model	Testing	
	f1-score (%)	loss
RandomForestClassifier	60.9 ±0.06	0.392
DecisionTreeClassifier	64 ±0.4	0.36
GaussianNB	45.2 ±1	0.548
QDA	52.1 ±0.7	0.464
SVM	82.8 ±0.4	0.17
MovieNet (MLP)	85.9 ±0.3	0.36

Table 4.4: Testing results of each model with the best configurations.

Chapter 5

Conlusions

Bibliography

[1] GroupLens. MovieLens Latest Datasets. https://grouplens.org/datasets/movielens/latest/.

[2] TMDB Community. Api Overview. https://www.themoviedb.org/documentation/api/.

[3] IMDb.com. Imdb datasets. https://www.imdb.com/interfaces/.

[4] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[5] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[6] Guillaume Lemaître, Fernando Nogueira, and Christos K. Aridas. Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning. *Journal of Machine Learning Research*, 18(17):1–5, 2017.