Google Play Store Apps

(Big Data Analytics project)

D'Apoli Clara (531889), Iannarelli Aldo (602009), Macaluso Vitalba (603160), and Tribuzio Daniele (602020)

Università di Pisa, DI

1 Data semantics

The provided dataset collects app data that users can download from Google Play Store, the Google's official pre-installed app store on Android-certified devices, which provides access to a variety of contents including apps, books, magazines, music, movies and TV shows.
The dataset consists of 10841 records and 13 variables, that we describes as following.

- **App.** The application name in the Google Play Store;
- **Category.** The category the app belongs;
- **Rating.** A quantitative variable for app evaluation, that can only assume values from 1.0 to 5.0;
- **Reviews.** Number of reviews per app;
- **Size.** App dimension, that is expressed in two different units of measurement (Mb and kb);
- **Installs.** Number of download per app, that is expressed as a categorical variable using the notation '###+';
- **Type.** If the app is free or not, that can assume only values 'Free' and 'Paid';
- **Price.** The cost of the app, expressed in dollars using the notation '###$';
- **Content Rating.** Age group the app is targeted at;
- **Genres.** Any multiple genres the app belongs to, containing the main genre, always as first, and the secondary ones;
- **Last Updated.** Date when the app was last updated on Play Store, expressed using the *'%B %d, %Y'* format;
- **Current Ver.** Current version of the app;
- **Android Ver.** Minimum required Android version to download the app.

2 Data quality evaluation

Starting from a syntactic accuracy evaluation, we noticed that Category is a categorical attribute, as we expected, but it contains a value that is not in the domain, actually a *float*. So, we investigated the reason behind the incorrect value and it was found that the 10472^{nd} record was one tab shifted, therefore we

regularized it. After the alignment, we noticed that Rating and Reviews values have been interpreted as *string* instead of *float* and *integer* types, respectively; so, we re-transformed them in the correct form.
The remained attributes are syntactically and semantically accurate, so other eventual transformations will be applied in order to make data more manageable.

2.1 Missing values detection.

At first, we observed that the target variable Rating had 1474 missing values; so, we decided to remove these records because it was improper handling missing target in any way.
The attributes Category and Genres had only one missing value at the previously aligned record, that actually is the 9117^{th} one, so we took into account all those records that had the same values for Content Rating and Size, since applications belonging to the same category have similar dimensions and are often aimed at the same audience. We noticed that the mode, for these kind of records was 'TOOLS' both for Category and Genres, so we substituted the only missing value using this information.
The attribute Current Ver had four missing values and, since current and recently updated app versions could be supported by newer android version more likely, we combined Last Updated and Android Ver variables to compute the mode of Current Ver and substituted using the obtained statistics. In order to manage the two missing values present in Android Ver, the corresponding Last Updated values was taken into account since newer updates are more likely to be supported by newer Android versions. It has been also observed that apps with either 'March 2018' and 'July 2018', as last updates, had the android version '4.1 and up' as mode, so this information it has been used to handle the missing.

3 Variables transformation

In order to have a common units of measurement for the attribute Size, all values expressed in Kilobyte have been converted to Megabyte. In addition, where the dimension was 'Varies with device', the values has been replaced with '0' and then, all values has been transformed into a *float*. As a means of making the attribute Price more manageable, it has been transformed into a *float*, after removing the dollar symbol ($).
Another transformation involved the attribute Genres, that was composed of one or two words, respectively indicating the main and the eventual secondary genres of the app, separated by a semicolon; so the variable has been split in two different columns, **Main_Genres** and **Secondary_Genres**.
The variable Installs has been rewritten as interval, giving the plus symbol (+) the meaning of "more than...", in order to manage this attribute as an ordinal one [1]. The attribute Last Updated represented the date as a *string*, so it has been transformed into the correct *datetime* format: there were considered only month and year assuming that the information about the day was not interesting for the analysis.

3.1 New variables creation

Looking at the Category, we observed that 'FAMILY' and 'GAME' represents the categories with the largest number of records. So, we wanted to keep these macro-categories separated from the others creating two new *boolean* attributes:

- **Is_Family**, assumes value 1 if the app falls into the 'FAMILY' category, 0 otherwise.
- **Is_Game**, assumes value 1 if the app is a game, 0 otherwise.

The last step of this phase was based on the creation of two additional variables: Compatibility and RTR.
Compatibility, represents the app compatibility according to the Android version. The assumption is that more Android versions are supported by an app, more they could have high ratings, due to the greater usability. In particular, apps which have 'Varies with device' are the most usable ones, because developers are able to directly identify which is the best version according to devices [2].

- 'Varies with device' is kept;
- Version between '1.0 and up' and '2.3.3 and up' and also '4.4W and up' are classified to have an high compatibility, because those apps are supported by the oldest android version and smartwatch;
- Version between '7.0 and up' and '8.0 and up' and also '5.0 - 6.0' are classified to have a low compatibility, for the opposed reason of the previous category;
- All the other Android version are classified to have a medium compatibility.

The attribute **RTR** (Reviews-Through Rate) was computed based on the fact that some apps can have the same number of reviews, but different number of download and, for this reason, some mistakes can lead because the weight of reviews is different if an app have much more installs than another. The attribute RTR (Reviews-Through Rate) has been computed to give different weights to those apps that have the same number of reviews, but different number of downloads. So, in order to make the most of this intuition, we solved the problem of Installs division into ranges, taking into account the mean of each interval.

4 Elimination of redundant variables and duplicates

4.1 Elimination of redundant variables

Dealing with data quality evaluation, the variables Type and Price gave us similar kind of information in different form, and we decided to remove the first one because it can be extracted from the second one, but not vice-versa.
The variable Genres can be removed because of the creation of the two distinct Main_Genres and Secondary_Genres. Moreover, Category and Main_Genres seemed to be redundant variables, after 'Is_Family' and 'Is_Game' creation.

4.2 Handling duplicates

At the end of the data cleaning and preparation phases, we checked for duplicated records and, as they came out, we decided to keep the records we supposed to be newer considering the number of reviews. At the end, we obtained a dataset consisting of 8197 distinct apps.

5 Pairwise correlation

Correlation analysis was made using both correlation and scatter matrix. The Fig. 1 shows that the correlation between quantitative variables are close to 0, while it is a little bit higher for Rating and RTR (corr = 0.19). In fact, looking at their plot in the scatter matrix (Fig. 2), these two variables are the only ones that show a slight trend. Despite there isn't correlation between Rating and Reviews, it is observed that, as the number of reviews increases, the rating also increases until it stabilizes around 4.0. From the scatter matrix diagonal emerges that Reviews, Price and RTR are almost uniformly distributed around the lowest values, while Rating and Size seem to have opposite distribution among each other.

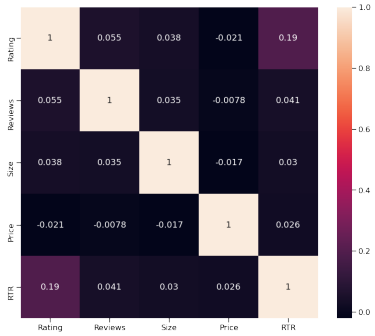

Fig. 1. Correlation matrix

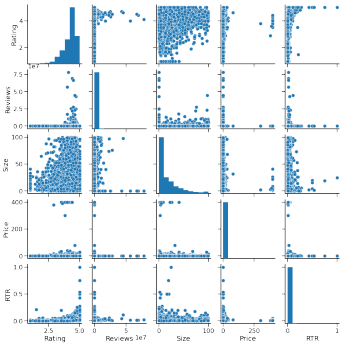

Fig. 2. Scatter matrix

Outliers and noise. The attribute Price shows in Fig. 1 that it contains potential outliers because there are some values too much greater than the others, so we decided to investigate them to find out a possible association with the app categories or between apps themselves. As result, in Tab. 1, categories these apps belong to are three different one, but, seeing at the names, they all them contains the expression 'I am rich'. About that, we discovered that these kind of apps represents such a status symbol, without any functionalities.

It was also observed only one value 'Unrated' in the attribute Content Rating, which is considered a noise, and therefore replaced with the mode according to the Main_Genres the app belongs to.

Table 1. Categories for price greater than 79.99$.

Price	Category	Frequency
299.99	LIFESTYLE	1
379.99	LIFESTYLE	1
389.99	FAMILY	1
399.99	FINANCE	6
	FAMILY	3
	LIFESTYLE	2
400.00	LIFESTYLE	1

6 Integration from external sources

The use of information about the apps world wasn't based only on the Android system. On Kaggle, we found a dataset [1], dating back to July 2018, the same period our dataset belong to, concerning the applications available on Apple Store (iOS system). This additional dataset contained attributes not related to data we have, but the presence of some apps in both stores lead us to preserve this information creating a new *boolean* attribute, called exactly **Both_Stores**. The reason behind this choice is the possible presence of differences among apps, in term of rating, due to the tendency of users to give higher evaluation when they have the possibility to use an app not only on the Android system. Since some apps are present in both stores with slightly different name, we decided to construct a sequence matcher that returns the similarity of app names in terms of percentage. We assumed a matching threshold of 80% to consider similar app names as potential same apps: 344 apps in both stores presented a match according to the threshold set and we opted for checking them one-to-one. After this procedure, additional 100 apps were found in both stores.

7 Distribution of the variables and statistics

The target variable Rating, according to the Google Play Store regulation, can assumes values from 1.0 to 5.0. Its distribution, represented in Fig. 3, shows a negative asymmetric one with a frequency spike around 4.3. Looking at the tail, frequencies decrease as ratings decrease too, until zero-frequency, increasing only for rating 1.0.

In Fig. 4 all the boolean variables are represented in the form of a boxplot with respect to Rating. At first, it seems there is no difference in the median rating if apps are not classified in family or game categories or they are not present in the Apple store. Instead, if apps are categorized as games, the median rating slightly decreases, but the main difference is observed for the variable Both_Store: when an app is available both in Google and Apple store, the median rating increases and there are no rating less than 3.0.

[1] https://www.kaggle.com/ramamet4/app-store-apple-data-set-10k-apps

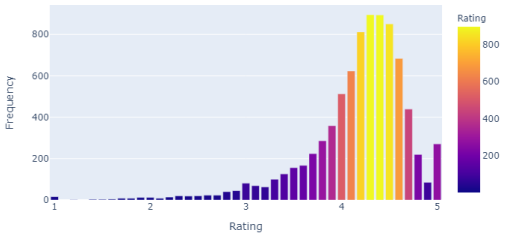

Fig. 3. Distribution of the target variable

The boxplot in Fig. 5 shows the distribution of Rating by the the variable Compatibility: apps that have compatibility which is 'Varies with device' obtains a median rating greater than the others and its distribution seems to be symmetric; on the other hand, apps that belong to the 'Medium' and 'High' compatibility categories, present largest range of rating and more potential outliers. As opposed to our expectation, also apps that have low compatibility obtained high rating.

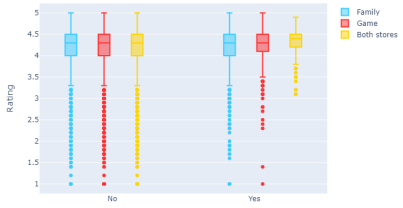

Fig. 4. Boxplots of boolean variable

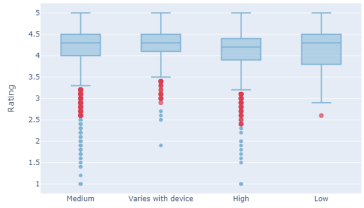

Fig. 5. Boxplot of Compatibility

Analyzing the distribution of RTR relative to Last Updated, distinguishing apps in Free and Paid, as the Fig. 7 shows, more recent updates (in 2018) corresponds to a slight increase for free apps, where these last are much more frequent than the paid ones in the dataset (Fig. 6). However, the RTR distribution is not affected by this imbalance between free and paid apps. Furthermore, what is surprisingly evident is that paid apps have a RTR much more greater than the free ones, indicating the fact that these last have a rather high review rate, probably because users tend to be more motivated to review an app they have paid for.

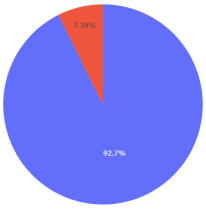

Fig. 6. Distribution of Price

Fig. 7. Distribution of RTR by Last Updated and Price

8 Task proposal

The main goal of this project is to predict ratings of the apps, according to the chosen variables, using a classification model. In particular, looking at the correlation between the app review rate and its rating, the first one variable would seem to be more useful than others for the prediction task, so we plan to verify the importance of RTR in the model we will train and whether at higher RTR values correspond higher app evaluation. Related to this aim, how rate values vary, according to free or paid apps, may reveal information about the goal of the project. At the end, the fact that an app is present in both Google and Apple store is relevant to predict the target variable?

9 Classification

In order to predict (**apps ratings** we handled an Ordinal Regression problem, which is something between classification and regression: estimating ordinal classes on a fixed, discrete rating scale. In this purpose we used different Machine Learning models: Ordinal Regressor, Support Vector Machine Regressor, Decision Tree Regressor, Random Forest Regressor and Convolutional Neural Network. As a metodology, in each model we contextually performed hyper parameters tuning and variables encoding, but all the implemented models have agreed on the building of the train & test with a 70:30 split (random_state=42). Finally, we evaluate each model in terms of Root Mean Squared Error (**RMSE**), Mean Absolute Error (**MAE**), Pearson Correlation Coefficient (**r**) and Kolmogorov-Smirnov statistics (**KS**).

9.1 Decision Tree Regressor

The first regressor we tested is the **Decision Tree Regressor (DTR)**. Before creating the prediction model, some categorical variables needed to be converted into numerical one. So, Content Rating, Current Ver, Compatibility, Installs, Secondary_Genres, Main_Genres and Android Ver have been encoded through

the Label Encoder. Regarding Last Updated, we extracted the month and the year the last update was released and we made two new columns Month and Year respectively.
At the first model implementation all numerical variables have been used as input and after Hyper-parameters Tuning implementation, using MSE as scoring object and cross validation with number of folds 5, the parameters for testing are chosen from the best estimator and set in the DTR (Table 2).

Table 2. Decision Trees results

Criterion	Max-Depth	Min SS	Min SL	MAE	MSE	RMSE	Pearson (r)	KS
MSE	10	2	200	0.34	0.25	0.50	0.37	0.30

First of all we noticed that the model returned reasonable values for all metrics, especially for the error calculation ones, but we chose to investigate the feature importance so as to try to improve the tree. Fig. 8 summarizes the results obtained.

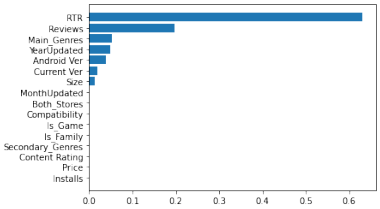

Fig. 8. Features Importance distribution

What immediately emerges is that the new variable created RTR is evaluated by the algorithm as the best for splitting, while the engineered Both_Stores and Compatibility do not provide any contribution, with an importance equal to 0 in both cases. The model also seems to have benefited from the Last Updated division. However, after doing several experiments by considering different parameters and features, we noticed that the same performances were achieved by removing meaningless attributes. So, we opted for decreasing the number of variables from 17 to 4 (RTR, Reviews, Main_Genres and YearUpdated) in order to reduce the complexity of the model. Indeed, after a replication of the Grid search approach, the best estimator simplified the tree with a reduction in depth and in the number of leaf. Below, in Table 3 it can be observed the values of the key metrics performed by the model, validated on both training and test set, and in Fig. 9 the relative 'simpler' tree. Also a slight improvement in the KS was achieved compared to the previous model.

Table 3. Decision Trees results after features selection

Criterion	Max-Depth	Min SS	Min SL	MAE	MSE	RMSE	Pearson (r)	KS
MSE	8	2	100	0.34	0.25	0.50	0.37	0.25

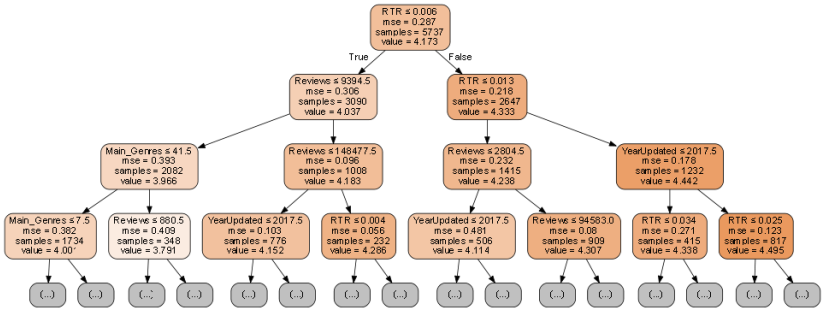

Fig. 9. Decision Tree

9.2 Support Vector Machine Regressor

For the **Support Vector Machine Regressor (SVMR)** implementation the sklearn library sklearn.svm.SVR [2] have been used. First of all, the dataset has been prepared with some preprocessing techniques:

1. All the nominal variables values have been encoded through the LabelEncoder.
2. All values in the dataset have been scaled through the StandardScaler.

Kernel function evaluation The SVR model provides different Kernel functions to be used in the machine learning algorithm, so we decide to implement a GridSearchCV search in order to estimate the best Kernel function for our dataset. The results shows that the best Kernel function is RBF.

Hyperparameters tuning After the Kernel function selection, another GridSearchCV has been implemented for Hyperparameter tuning and the best parameters returned have been chosen for the model.

- gamma: 'auto'
- tol: 1e-2
- epsilon 0.1
- C: 1
- kernel: 'RBF'

[2] https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVR.html

Feature selection First we tried to remove Installs and Reviews attributes because already included in the new engineered RTR, but the model produced the same results. Therefore, in order to get an objective and more efficient features importance, it was decided to use a features selection approach, implementing the sklearn RFECV [3] algorithm. The selected features are: Rating, Reviews, Price, Content Rating, Last Updated, Main_Genres, Compatibility and RTR. Using the above features the GridSearchCV produces the same hyperparameters of the previous model, however when testing the model, the performance on the test set have been increased. Then, another features selection has been implemented splitting Last Updated in two columns: Year and Month. The ranking results are the following:

- Rank 1: RTR, Last Updated, Year, Month
- Rank 2: Content Rating
- Rank 3: Price
- Rank 4: Compatibility
- Rank 5: Current Ver
- Rank 6: Reviews
- Rank 7: Main_Genres
- Rank 8: Secondary_Genres
- Rank 9: Is_Family
- Rank 10: Installs
- Rank 11: Size
- Rank 12: Both_Stores
- Rank 13: Android Ver
- Rank 14: Is_Game

According to the ranking, we considered only features up to Rank 7 (included). Last Updated was not taken into account because considered redundant.

Other experiments In order to increase the performance of the model other experiments have been done:

- **splitting the problem:** the prediction problem has been split in two different predictive ones: classification and regression. In particular, the classifier (SVC) has been implemented to predict if the rating of an App is above or below the mean and then two different SVMRs were used to predict the ratings in the two classes estimated by the classifier. The split model was implemented in a sort of pipeline for the test set.
- **oversampling the dataset:** the dataset has been integreted with synthetic data in order to balance the ratings distribution, but the results were not satisfactory.

All the results are summarized in Tab. 4.

Best model According to the results in Table 4, we chose the model with the lowest values for error metrics and KS and the highest value of Pearson r. The corresponding predictions are shown in Figure 10.

[3] https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.RFECV.html

Table 4. SVMR performances

Metrics	Original DS	RFECV	RFECV Month - Year	Problem split
MSE	0.275	0.271	0.269	0.325
RMSE	0.524	0.520	0.518	0.570
MAE	0.343	0.341	0.338	0.404
Pearson (r)	0.35	0.376	0.384	0.265
KS Test	0.256	0.285	0.258	0.355

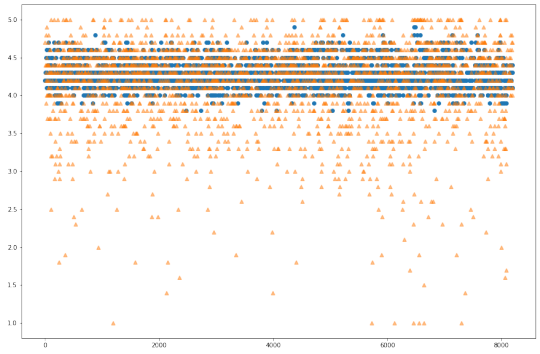

Fig. 10. Results predicted from the best SVM model

9.3 Ordinal Regressor

Since our analysis aims at the prediction of an ordinal variable also an **Ordinal Regressor (OR)** have been implemented.

Implementation In order to perform the OR algorithm, two regression-based models of the Mord library [4] have been used: LAD (Least Absolute Deviation) and OrdinalRidge. Before applying the model to the data, the categorical variables have been transformed into numerical ones, while Last Updated have been divided into YearUpdated and MonthUpdated. Then, a Grid search approach with MAE as the scoring object have been implemented in both models in order to find the ones with the best parameters in terms of optimization of this metric. The best parameters chose by the algorithm are the following (Tab.5):

Table 5. LAD & OrdinalRidge Best estimators

Model	C	epsilon	loss	tol	alpha	solver
LAD	1	0.0	squared_epsilon_insensitive	0.0001		
OrdinalRidge				0.0001	0.001	svd

The two models have been evaluated in terms of key metrics and the OrdinalRidge turned out to be significantly the most performing:

[4] https://pythonhosted.org/mord/

Table 6. LAD & OrdinalRidge Evaluation

Model	MAE	RMSE	MSE	r	KS
LAD	0.75	1.17	1.39	-0.143	0.704
OrdinalRidge	0.44	0.66	0.32	0.095	0.69

9.4 RandomForestRegressor

With the purpose of dealing with the regression problem also a **RandomForest Regressor (RFR)** have been implemented, trying to improve the performance of the DTR.

Implementation Before implementing the model, the categorical variables have been encoded through a Label Encoder and a Grid search was used to find the best parameters for the model, in terms of optimization of MAE value.
The performance of the model are the following in Table 7.

Table 7. Random Forest Regressor performance

Model	MSE	RMSE	MAE	r	KS
Results	0.24	0.48	0.33	0.44	0.29

Furthermore, the ten most important features for the model have been extracted and represented through a bar chart (Fig. 7). As it can be seen, as for the DTR, RTR and Reviews are the most important features, while all the others have similar values below 0.10.

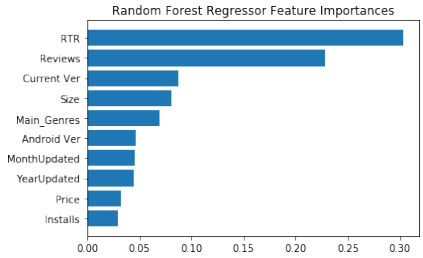

Fig. 11. RandomForestRegression Features Importance

10 Convolutional Neural Network

Convolutional Neural Networks (CNN) is often used to solve multi-class and ordinal classification problems, even if they were born to model visual data. In order to predict the target variable, they were implemented considering three different point of view:

1. Multi-class Classification;
2. Ordinal Regression;
3. Stacked-Ensemble.

The first step was involved in adding four new numerical attributes: Last Updated was split into YearUpdated, MonthUpdated and DayUpdated, because these kinds of variable may give some additional information; the last variable created was SentimentPolarityMean and derives from the 'googleplaystore_user_reviews' .csv dataset. In particular, we applied the mean of the sentiment polarity per app and we added it in the dataset, in order to have more quantitative information.

10.1 Multi-class Classification

The multi-classification consists of reducing the target variable as a set of binary output for preserving the natural order of the classes and avoiding to have outputs that are not exactly in the domain of the target. In this context, we decided to investigate feature importance computing the ANOVA F-value and to remove only variables with F-value ≤ 1.0 to avoid underfitting.
All the models we found have been built considering 'ReLu' as activation functions in all the layers except the last one, where we consider a 'SoftMax' activation function since can return output as the probabilities of belonging to one class. In Tab. 8 we summarized the performance of the fourth best multi-classification models and we can say that *Model_1* is the best, because RMSE and MAE are the lowest one and Pearson (r) is the highest one, even if KS coefficient is the worst among all the models. The CNN architectures found are slightly different from each other: Model_1 and Model_4 were obtained by splitting the whole dataset according to the type of the attributes (categorial, ordinal and numerical) and then they are merged to result target classes. In particular, we built CNN only for the subset containing numerical attributes. On the other hand, Model_2 and Model_3 were obtained considering simpler architecture: we passes the whole dataset in a single convolutional network.
The differences between these two pairs of models is based on the SMOTE over-sampling: since our classes are very imbalanced (see Fig. 3) and we always obtained most classes prediction between 4.0 and 5.0, we thought that could be a good solution over-sampling minority classes, but this operation did not improve the performance of the models.

Table 8. Performance CNN multi-classification

Models	RMSE	MAE	Pearson (r)	KS
Model_1	0.57	0.37	0.19	0.33
Model_2	0.73	0.52	0.01	0.04
Model_3 (SMOTE)	0.84	0.59	0.01	0.06
Model_4 (SMOTE)	0.65	0.59	0.17	0.07

10.2 Ordinal Regression

The choice to implement an ordinal regression considering the real type of the target variable was based on the fact that the multi-class classifiers performed not well and we hypothesized that the reason could be the excessive number of target classes. In fact, we can notice in Tab. 9 that RMSE, MAE and Pearson (r) improved considering each of these models. In particular, Model_7 was built only on the attributes with F-score *leq* 1.92 and consists in two convolutional layers with a max pooling; Model_6, instead, was built using three convolutional layers using also BatchNormalization and max pooling. So, *Model_7* could be considered the best one because it is simpler than Model_6

Table 9. Performance CNN ordinal regression

Models	RMSE	MAE	Pearson (r)	KS
Model_5	0.54	0.41	0.20	0.56
Model_6	0.50	0.34	0.34	0.24
Model_7	0.50	0.24	0.34	0.24

10.3 Stacked-Ensemble

This kind of architecture is useful when we want to use the output of a neural network as input of another one and this may often improve the performance of the overall model [5]. It seems like what we did in the Model_1 and Model_4, but in this case we used an ensemble method in which the model can learn combining several models generated from the same input (in this case). In order to test the model capability to predict target classes, we decided to define 11 models using the best ones we explained before: Model_7 and Model_1 (only the CNN branch). The result, in terms of performance, is shown in Tab. 9. As we expected, the performance improves, even if only a little bit, but now we can definitely confirm that *Model_8* is the best CNN model found for the classification task and it can be seen in Fig. 12

Table 10. Performance CNN Stacked-Ensemble

Model	RMSE	MAE	Pearson (r)	KS
Model_7	0.57	0.39	0.27	0.41

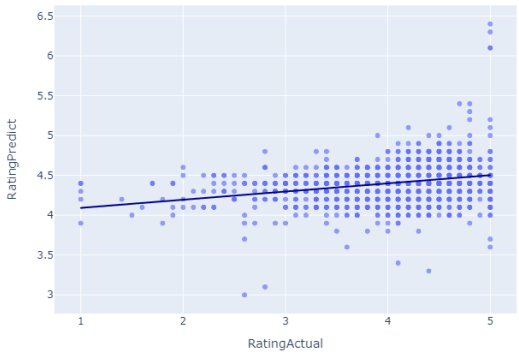

Fig. 12. Actual vs. Predict rating

11 Conclusion

We investigated and evaluated several kind of models, but two of these stood out: the Decision Tree Regressor and the Random Forest Regressor. All their evaluation metrics are comparable and better than the ones of the other models considered. In addition, we know that these two models are very simple and require less computational costs. So, since the performance are very very similar, we decided to take into account the simplest one, which is the **Decision Tree Regressor**.

References

1. CPIDroid 'How Google Displays the App Installs Count on Google Play Listing?', https://tinyurl.com/yxcsl5mg. Apr 20
2. Google Operating System 'Varies With Device', http://googlesystem.blogspot.com/2013/08/varies-with-device.html. Aug 13
3. Saedsayad 'Support Vector Machine - Regression (SVR)', https://www.saedsayad.com/support_vector_machine_reg.htm.
4. Fabian Pedregosa 'On the Consistency of Ordinal Regression Methods', https://jmlr.csail.mit.edu/papers/volume18/15-495/15-495.pdf. 2017
5. ML From Scratch 'Stack Machine Learning Models - Get Better Results', https://mlfromscratch.com/model-stacking-explained/#/. Jan 20