APPeal: Android's way to success

Erica Cau
e.cau@studenti.unipi.it
Student ID: 545126

Andrea Failla
a.failla@studenti.unipi.it
Student ID: 627098

Federico Mazzoni
f.mazzoni6@studenti.unipi.it
Student ID: 524324

1 INTRODUCTION

Since the release of the first iPhone in 2007, smartphone apps have become more and more ingrained into everybody's everyday life, with a growing digital market that lead to the launch of competitive platforms, most notably Android. Nowadays, Android is one of the most popular operating systems in the world, and most people get access to new apps through Google's Play Store, which features applications built both by indie developers a.w.a. consolidated corporations. Of course, not every app on the Play Store is successful and it is therefore legitimate to ask whether there are some key features that such successful apps have in common.

2 DATA UNDERSTANDING

This study was conducted using the *Google Play Store Apps* dataset, made available on Kaggle by Gautham Prakash[1]. The dataset included more than 2.3 million of records and 24 features. For this reason, such a project couldn't have easily been done locally and the computations had to be distributed, which led us to use the PySpark library.
The 24 features can be roughly defined as:

- **App-identifiers**, such as *App Name* and *App Id*;
- **Information about the App**, including *Category*, *Rating* (from 0 to 5 stars), *Rating Count* (i.e. how many times the App has been voted by the user-base), *Content Rating* (i.e. the suggested age rating, e.g. "10+", "Adults Only" etc.), *Editor's Choice* (whether the app has been hand-picked by the Play Store editors as a quality app), *Installs*, *Minimum Installs* and *Maximum Installs*;
- **Information about the Price** (and the business model of the app), including the *Price* itself and its *Currency* of the price and whether the app is *Free*, *Add Supported* or has *In App Purchases*;
- **System Requirements**, including the *Size* of the app and the *Minimum Android* version required for the installation;
- **Informations about the Developers**, including the *Developer Id*, the *Developer Email* and the *Developer Website*. Additionally, the *Privacy Policy* feature links to the privacy page of the developer site;
- **Temporal information**, i.e. when the app was *Released* and *Last Updated*. An additional feature, *Scraped Time*, gives a meta-information on when the particular data point was scraped.

In the first part of this analysis we studied the distribution of the features and checked for the existence of missing values, in order to address and solve this problem in the *data preparation* section (see Section 3). We created a subset of numeric columns (including Installs, Rating Count, Rating, Price and Minimum Android) and calculated their mean, the standard deviation, as well as the max and min values. In addition, we also extracted the quartiles. It is necessary to outline that *Installs* and *Rating count* have a high standard deviation, due to extremely high values of some apps in those attributes, which is in contrast to the high number of apps with few device installations or reviews.
We also found out that most of the dataset features have missing values. In particular the highest numbers were registered in *Developer Website* (760.826 values), *Released* (710.55 values) and *Privacy Policy* (420.947 values). After this preliminary analysis, we moved onto the visualization of the features distribution. As it can be seen in Fig 1, the dataset has a very large population of *Educational* apps (i.e., apps having *Education* as their *Category*), while *Productivity* and *Fitness* app are relatively scarce.

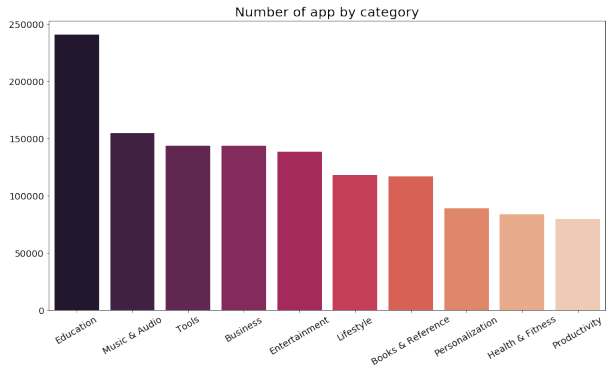

Figure 1: Number of app per Category

We also studied the average rating and found out that most categories have an average rating of 2 stars, with only 5 categories, all of them gaming-related, exceeding the 3 stars ranking: *Role Playing, Casino, Simulation, Weather* and *Card* (see Table 1). In addition to this, we noticed that most of the apps require Android version 4.0 (Ice Cream Sandwich), or 4.4 (KitKat), or higher; note that it is still uncommon to find apps that need at least Android 8 (Oreo).

Category	Average rating
Role Playing	3.38
Casino	3.27
Simulation	3.21
Weather	3.12
Card	3.08

Table 1: Top 5 rated app categories on Play Store

[1]Google Play Store Apps on Kaggle: https://www.kaggle.com/gauthamp10/google-playstore-apps

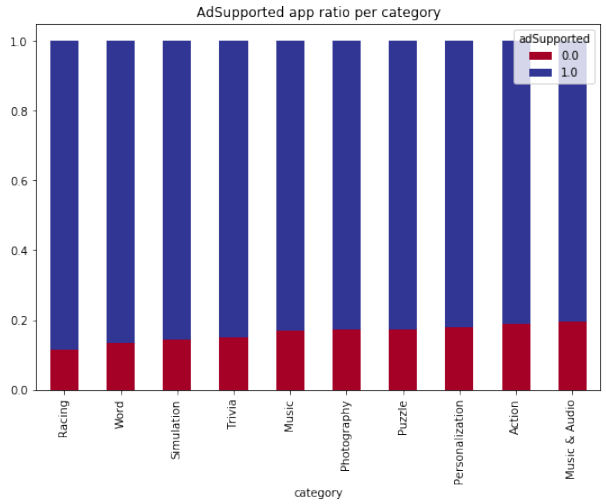

Figure 2: *AdSupported* ratio between categories

Speaking of the business model, according to the dataset, on Google Play Store there is an even split between apps with and without ads (1.162.150 apps that does not support ads, against 1.150.750 that actually do support them). Going into further details, we visualized the ratio of the *AdSupported* feature for each category into a stacked bar chart. The categories with an higher ratio are mostly games that accept in-game transactions. The only exceptions are represented by *Music & Audio*, *Personalization* and *Photography*. As for the price, we analyzed only the average price of paid apps per category ignoring free apps with a price equal to 0: the most expensive category turned out to be *Video Player & Editors* (see Fig.3), with an average price of 5 dollars, followed by other categories of games, such as *Board* (4.9 dollars) and *Casino* (4.75 dollars).

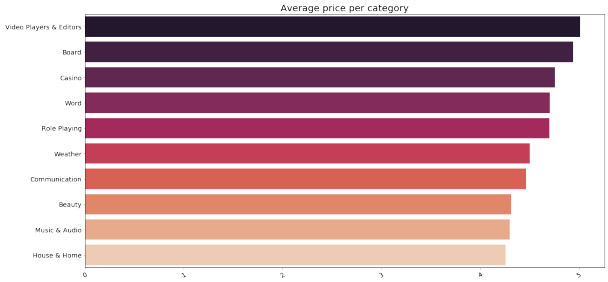

Figure 3: Average apps price per category

3 DATA PREPARATION

Of those features, *Minimum Installs*, *Maximum Installs*, *Free*, *Developer Website*, *Developer Email*, *Privacy Policy* and *Scraped Time* were deemed redundant or useless for the aim of this project and were therefore removed.
Out of the remaining features, some of them were used to clean the dataset or create new features:

- The *Size* feature had the value expressed in MB, GB or kB. The different values were converted to MB for the entire dataset;
- We only kept data having *USD* as *Currency*, because of overwhelming majority. The feature was then removed;
- textitRating Count and *Content Rating* were renamed *nReviews* and *Age* for the sake of clarity. The values of *Age* were converted into a 0-5 numerical scale (e.g. "Everyone" = 0, "Everyone 10+" = 1, "Teen" = 2, etc.). The value *Unrated* was replaced with *-1*.
- The value *Varies with device* of *Mimimum Android* was replaced by *0.0*.
- We created the features *Year of Release*, *Quarter of Release* and *Year of Last Update* out of *Released* and *Last Updated*, in order to have a more general information. The original feature were then removed;
- We created the feature *Reviews_Installs_Ratio*, from *nReviews* and *Installs*;
- We computed the standard deviation from the mean of the *Category* feature for *nReviews*, *Installs*, *Price* and *Size*.

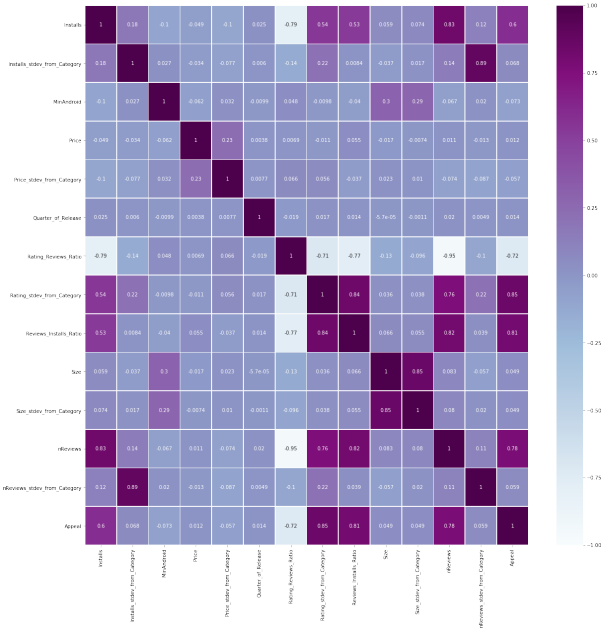

Figure 4: Correlation matrix of the dataset after preprocessing

At the end of the data cleaning phase, we also removed the records with missing values, ending up with 2089852 records and 24 features. In the end we also plotted the correlation matrix using the Spearman coefficient (see 4.

4 CLUSTERING

PySpark offers multiple kinds of clustering algorithms. For this paper, we decided to use the regular *K-Means* and its *Bisecting* variant, a.w.a. the *Gaussian Mixture Model*.

4.1 K-Means

At first, we scaled the dataframe using the *MinMaxScaler*. We decided to use *Size, Price, Reviews_Installs_Ratio* and *Rating* as clustering features. The Silhouette score suggested that the best value for *K* was 3, as shown in figure 5.
Cluster 1 was the the most unique, while Cluster 2 and Cluster 0 were closer to each other. The average *Rating* and *Review Rating Ratio* of the apps in Cluster 1 were significantly lower (0.005 and 1.29E-4) than the one of the other clusters (3.16 and 0.06 for Cluster 2, 0.5 and 4 for Cluster 3). Those results confirm the linear relation between *Review Rating Ratio* (created from *Installs* and *nReviews*) and *Rating*, already seen in the Data Understanding phase. Interestingly, Cluster 1 has an higher percentage of apps without Ads (58% vs. 47% and 48%) and In App Purchases (98% vs. 88% and 87%).

4.1.1 Clustering as a means for outlier detection. Since PySpark does not feature any outlier detection algorithm, we also decided to use the output of the K-Means to detect outliers. We posit that the outliers equal to 1% of our dataset; we then ordered the records according to their distance from the respective centroid and removed the farthest 1%.

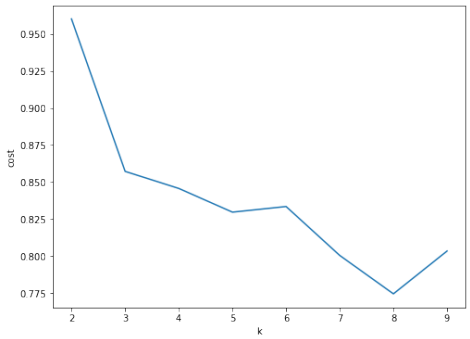

Figure 5: Silhouette scores for K-Means

4.2 Bisecting K-Means

Results obtained by using the *Bisecting* variant of the K-Means were comparable to those already discussed.

4.3 Gaussian Mixture Model

The results obtained by using the *Gaussian Mixture* Model greatly differ from those of the K-Means, starting from the number of clusters suggested by the Silhouette score (7, as seen in Figure 6). Of those seven clusters, Clusters 2, 4 and 6 included apps with a low *Reviews Installs Ratio* but with a relatively high *Rating*. Apps included in Cluster 4 also had a higher price. By contrast, Clusters 3 and 5 had a very low *Rating* and *Reviews Installs Ratio*. Clusters 0 and 1 both had the *Reviews Installs Ratio* greater than *1*, thus including a great percentage of apps with the *nReviews* value higher then *Installs* value. Those clusters diverged in the *Rating* value: Cluster 0 had a higher *Rating* than Cluster 1 (4.12 vs. 1.74).

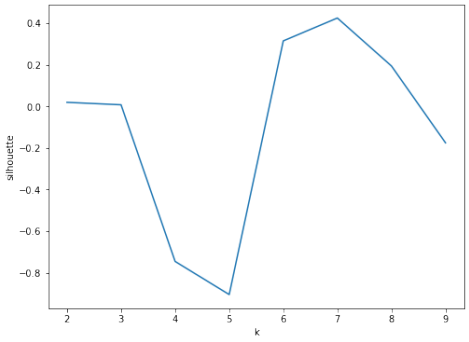

Figure 6: Silhouette scores for GMM

5 CLASSIFICATION

For the classification task, we tried to automatically recognize the apps according to their appeal through the means of different classification algorithms already implemented in PySpark. This decision led to the creation of a new variable, called *Appeal*, extracted from the pre-existing feature *Rating*. For a *Rating* value of 3.5 or more, *Appeal* is *1*, otherwise it is *0*.

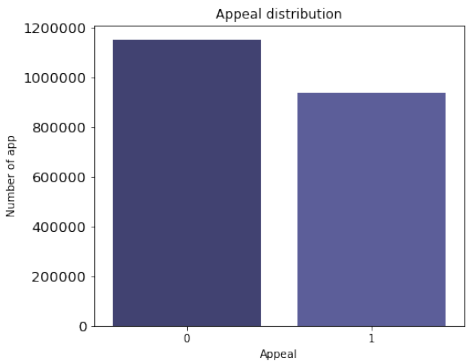

Figure 7: Appeal distribution

Moreover, for the classification task we opted to use a subset of the original features, including *Year_of_Release*, *Age*, *EditorsChoice*, *AdSupported*, *Quarter_of_Release*, *InApp*, *Year_of_Last_Update Price* and *Price_stdev_from_Category*. At first, we also tried to use *Installs* and *Reviews_Installs_Ratio*, but since the classifiers assigned those features too much weight (e.g., the results of the Decision Tree were for 98% due to the value of *Installs*), we preferred not to use them. The dataset was also split into two different segments: the 70% of the records became the *training set*, the remaining 30% was hold out as a *test set*.

5.1 Decision Tree

First of all, we looked for the set of the best parameters by using a *GridSearch*. We then trained the model setting *entropy* as impurity measure, the *maxDepth* of the tree to 5 and the *maxBins* parameter

Classification models					
	Decision Tree	**Random Forest**	**SVM**	**Logistic Regression**	**Naïve Bayes Classifier**
Accuracy	0.63	0.64	0.58	0.65	0.59
Precision	0.61	0.65	0.53	0.65	0.70
Recall	0.48	0.43	0.59	0.48	0.17
F1 Score	0.43	0.51	0.56	0.55	0.27
AUC-ROC	0.51	0.68	0.66	0.68	0.46

Table 2: Classifiers' results on the test set of the models

to 2. The model was later evaluated using the test set: the Precision obtained was fine (0.61), but the Recall problematic, with a value lower than 0.5 (0.48). From the confusion matrix it can be noticed that, due to the higher number of FN than TP, the classifier does not correctly recognize the *Appeal*.

5.2 Random Forest

As above, we trained the model after running a GridSearch. From the evaluation of the model we noticed that the Decision Tree performed better than the Random Forest, not only in terms of Precision, Recall and F1-score but also in terms of recognizing the *appealing* apps (120.150 app vs 135.230). In one field, it fares better than the Decision Tree – it has a higher AUC score (see Table 2).

5.3 Support Vector Machine

The SVM classifier, after the parameter tuning and MinMax scaling, gave the most interesting results, returning the highest scores for Recall and F1-Score. Overall, it is the best *Appeal* predictor, with the highest number of *appealing* apps correctly classified and a relatively lower number of false negatives (see Fig 8).

5.4 Logistic Regression

The model, after the parameter tuning and training, was the second best, after the SVM. It is affected by low Recall (0.48) and F1-score, but on the other hand all the other scores are equal or higher than 0.65.

5.5 Naïve Bayes Classifier

The Naïve Bayes Classifier was trained using the following features: *Age*, *EditorsChoice*, *AdSupported*, *InApp*, *Price*, *Size*, *Year_of_Release*, *Year_of_Last_Update*. It turned out to be the worst model for recognizing the *appeal*, since it recognizes only the 7.51% of successful apps; furthermore, it also has the lowest scores in term of recall and F1-score.

5.6 Feature importance

We also studied which features play an important role in finding the *appeal* of the applications, through the extraction of the weights assigned to each feature by the *Decision Tree* and the *Random Forest* models. With a weight score of around 0.4, both our models had the *Year of Release* feature playing a key role in determining whether the app is *appealing* or not. Another notable attribute is *AdSupported*: 0.41 for the *Decision Tree* model, 0.29 for the *Random Forest*. The lower weight given by *Random Forest* might explain the fewer

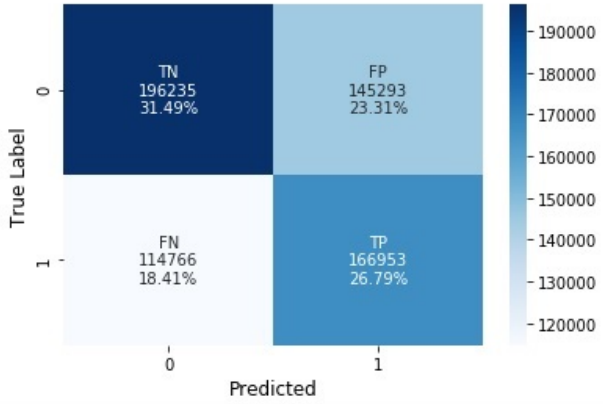

Figure 8: SVM confusion matrix

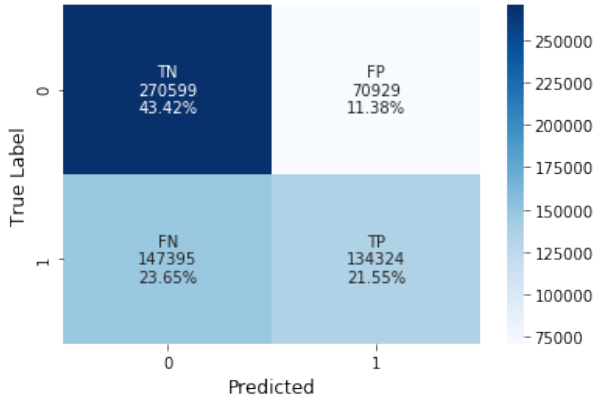

Figure 9: Logistic Regression confusion matrix

amount of TP recognized by the ensemble version of the *Decision Tree*.

Every other features were not considered important to find the *appeal*. The only other relatively "high" weighted feature of both classifiers is *InApp*, but its weight was only 0.12 for the *Decision Tree* and 0.17 for the *Random Forest*.

6 CROSS-CATEGORY CLASSIFICATION

We performed a cross-category classification using the best model emerged from the tests on the overall dataset, i.e. the SVM. We tested it both on the two most common categories, as per the *Data*

Understanding phase (i.e. *Music & Audio* and *Education*), and on *Arcade* and *Casual*, two similar categories both in terms of the apps included (usually very simple games) and the overall amount of *apps*. Before retraining the model, we performed a *GridSearch*, looking for the best parameter for the new, smaller dataset.
By training the model on the *Arcade* subset and then using it on the *Casual* we obtained similar results to the model used on the *Arcade* dataset itself. Those results were, however, quite poor (and a lot worse than the resuls obtained by SVM on the complete dataset) – notably, they failed to classify most of the *appealing* apps, as it can be seen in the Confusion Matrix of the *Casual* subset (Fig 10). On the other hand, the SVM model trained on the *Education* subset gave very different results when used on the on the *Education* subset itself and on the *Music* subset, as can be seen in Fig. 11 and 12. Surprisingly, the classifier performed better in the recognition of *Music & Audio* apps: this seems to confirm that the effective *appeal* is not bound to the category, but rather lies in other features.

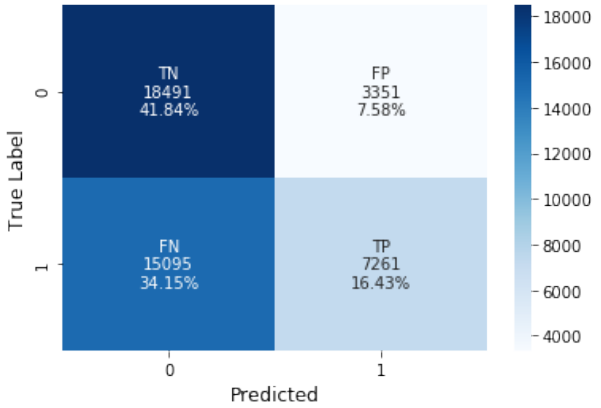

Figure 10: Confusion matrix of SVM trained on the *Arcade* subset and used on the *Casual* subset

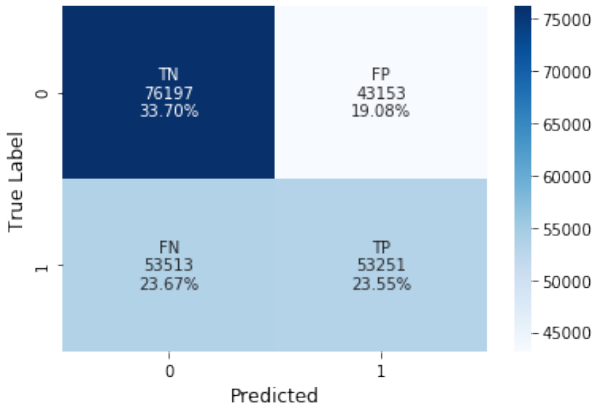

Figure 11: Confusion matrix of SVM trained and used on the *Education* subset

The underwhelming results obtained by using the subsets compared to the those obtained with the entire dataset suggest a lack

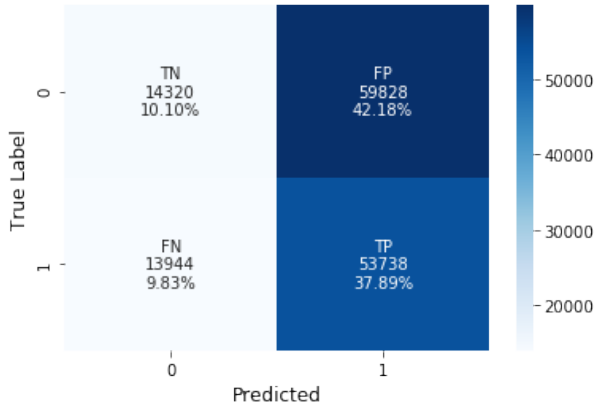

Figure 12: Confusion matrix of SVM trained on the *Education* subset and used on the *music* subset

of data, while the similar results of *Casual* and *Arcade* suggest that the vectors of those apps are, in fact, quite similar, while the same cannot be said for *Education* and *Music & Audio*.

7 DISCUSSION

If the aim of this project was trying to predict the *appealing* apps, it cannot be said to have been successful — while the results of SVM and Logistic Regression were relatively good, they left a lot of room for improvement. Still, the classification task we chose is far from being an easy one, and it is perhaps better to look at everything else we have discovered in this paper. For instance, the results of the *Data Understanding* were interesting in their own right (with the five specific gaming apps category having an higher rating, the majority of apps being educational, some considerations on the business model etc.) and the *Decision Tree* showing the importance of *Year of Release* confirms that *First-mover advantage* often discussed as a vital marketing strategy holds truth also on the Google Play Store.
Of course, as a benchmark for PySpark, the project shows how much mature the library is, allowing to complete a whole Data Mining project in a distributed context, while processing more than 2 millions of records.

8 ACKNOWLEDGEMENTS

The authors wish to acknowledge their friend Pippo for his continuous support, especially for debugging.