Introduction

Online reviews are an important and inevitable aspect of e-commerce. Before purchasing products or services, customers usually examine and rely on previous customers' or online reviews. Therefore, online reviews have a substantial effect on customers' decisions.

App stores are one of the common use of online reviews. The reviews include the users' feedback and experience related to the application. Users' reviews can lead to the revision of the application by users' needs or manipulate the potential users' decisions before downloading the application. Besides, research shows that positive reviews promote the download and sales of applications. Unfortunately, some application developers mislead users by posting fake and high-rating reviews. They can show their applications better or rival companies' applications worse than they are with fake reviews. As a result, they misguide users into making wrong decisions and cause economic damage to other application owners, so detecting fake reviews is highly necessary.

In this project, I aim to develop a model to detect and eliminate fake application reviews to prevent fake reviews from misdirecting users.

Dataset

The Apple App Store reviews dataset is created by Martens and Maalej (2019). There is a total of 16000 reviews. In addition, this dataset has three .json files related to users, reviews, and applications (for details, please see Martens & Maalej, 2019).

Features Data

Features data includes the extracted machine learning features related to users. There are 19 attributes, such as users' total number of reviews, review ratings, and account usage features. A total of 8696 unique users have posted the reviews. Also, in this data, reviews' label ("real" or "fake") and length, the applications' total number of reviews, and ratings are provided. These reviews belong to 5624 unique applications.

Reviews Data

Reviews data contains information about fake (n=8000) and real (n=8000) reviews. There are ten attributes: review body, title, posting time, vote, etc.

Applications Data

Applications data includes meta-data (e.g., price, version, genre) of applications related to the reviews. There are 31 attributes of 5563 unique applications in this dataset. The information of 61 applications that are in the features dataset is missing.

Data Wrangling

After loading .json files, I used essential functions (.shape, .dtypes, etc.) to investigate dataframes. The followings are the important points of the data wrangling stage.

- In the features data, users' frequency is defined as the average time in seconds between all reviews provided. A total of 1,734 cases are undefined because the user provided only a single review. The creator of the dataset set the frequency to the lifetime of the app store, which is nine years. We don't want to use this feature because it would create bias, so we dropped this attribute.
- I decided to create a new feature from the posting date of each review. The App Store was opened on July 10, 2008, with an initial 500 applications available. For creating a new feature, I found the time difference between the App Store release time and the review posting time.
- In the review data, each user has a name and id. I dropped the user id column because there are 761 NaNs.
- The application data has a feature that shows in which languages the application is. However, it is more important how many languages are rather than what the languages are. Therefore, a new column has been generated from the number of languages. There are 15 NaNs. I fill them with "1" because each app must address at least one language.
- The advisories feature in the reviews dataframe is related to warnings. It could investigate the similarity between advisory text and review body, but there are so many NaNs (67%). I dropped this column because there is no way to impute these NaNs.
- I got the major versions of applications from the version column. Some versions are higher than usual and seem related to the year. I checked ten applications' version history from

App Store. I changed some of them with 4 (the average version of all apps) or the version I got from version history, or I did not change them.

- A new feature that shows whether the application is paid or free has been created.
- The columns (i.e., currency, kind, wrapperType) with only one unique value have dropped.
- In "supportedDevices" column, there are many device names in a cell. New devices are added to these yearly, and their names are unimportant. Therefore, we decided not to use the device's name and created a new column for the number of devices.
- A new feature has been generated by finding the difference between the application release date and the last version's release date. That can be an indicator of how the application is well-set.

Exploratory Data Analysis

In the dataset, there are labels for reviews as fake or real. I have data about unique users and applications, but they don't have labels. Therefore, I investigated the fake review percentages of users and applications. After labeling users and applications, I conducted the exploratory data analysis using review, user, and app labels. This document shows all the research questions and the statistical methods for using answering these questions. The followings are the essential findings.

User Analysis

I have 8696 unique users. 7993 users (91.92%) haven't posted any fake reviews, while 700 (8.049%) users' all reviews are fake. Three users (0.03%) have both fake and genuine reviews. Their fake reviews percentage is more than 85%, so the likelihood of being a fake user is so high. Surprisingly, the users are gathered in two groups. Therefore, I labeled each user as fake or real user.

- There is an interesting finding about users who have just one review. 1734 users (19.9%) have just one review. 98.2% of these users are real, and 1.8% are fake. This is a statistically significant difference (X^2 *(1, N = 1734) = 92.31 , p < 0.05*) and important indicator of not being fake. I have created a new feature for users who have just one review. It can be said that fake users usually employ the same account more than once for posting fake reviews.

- While the average number of reviews posted by real users is 6.85, fake users are 31.5. According to the two-sample t-test result, there is a statistically significant difference in the number of reviews between real and fake users (t = -29.3, p < .05). Figure 1 shows the distribution of total reviews of fake and real users. Not surprisingly, the fake users' account usage (in seconds) is more than the real users (t = 4.27, p < .05).

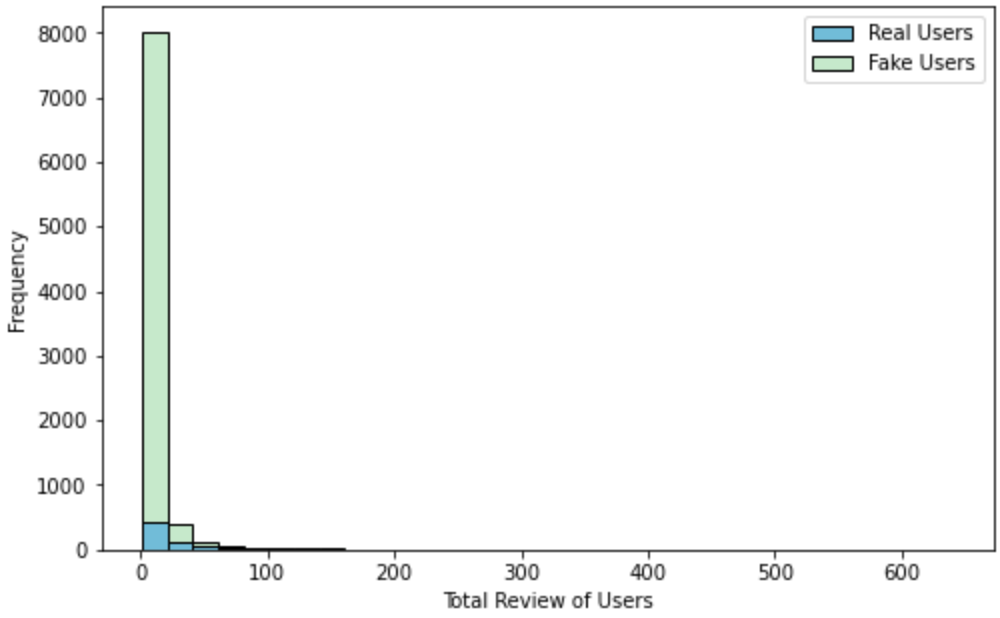

Figure 1. The distribution of fake and real users' total reviews

- Real users' ($\bar{X} = 0.06$) vote count is more than fake users ($\bar{X} = 0.04$), but this difference isn't statistically significant ($t = 0.50, p = .61$).
- Fake users give applications more ratings ($\bar{X} = 4.60$) than real users ($\bar{X} = 4.19$), which difference is statistically significant ($t = -8.05, p < .05$).
- As known, users have to pay for some applications to use. When the prices paid by the users are compared, it is seen that real users have paid ($\bar{X} = 0.69$ *(in dollars)*) for applications than fake ones ($\bar{X} = 0.14$ *(in dollars)*). This average is not statistically significant ($t = -0.91, p = .35$).

Review Analysis

The review labels' were predetermined, so I haven't changed anything. There are a total of 16000 reviews, and the followings are the findings about reviews.

- The reviews' posting year changes between 2008 and 2017. As seen in Figure 2a, the number of reviews has increased over the years. The total reviews in 2017 are less than in 2016 because the last review was posted in March 2017. While there were no fake reviews in 2008 and 2009, the number of fake comments in 2010 and 2011 was 1 and 2. There is a significant difference between posting years of reviews (X^2 (9, N = 16000) = 5821.22, p < .05). 2014 is a threshold for fake reviews, and there has been a dramatic increase this year. As seen in Figure 2b, the percentage of fake reviews is more than %65 after 2014.

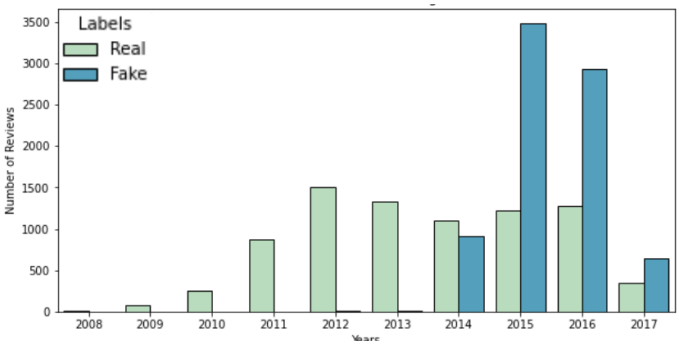

Figure 2a. The number of reviews according to years

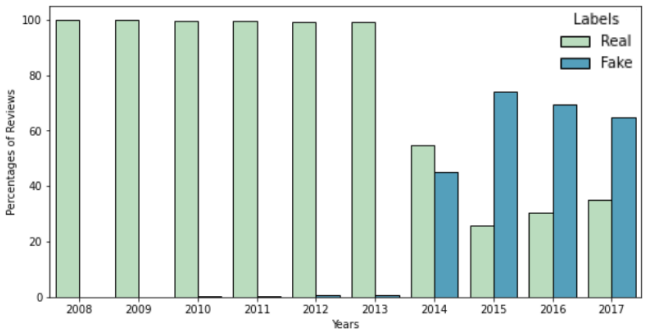

Figure 2b. The percentage of reviews according to year

- In each review, users give a rating between 1 and 5 for applications. Figure 3 shows the distribution of fake and real reviews according to ratings. As seen in Figure 3, the users like the application review and give it 5-stars.

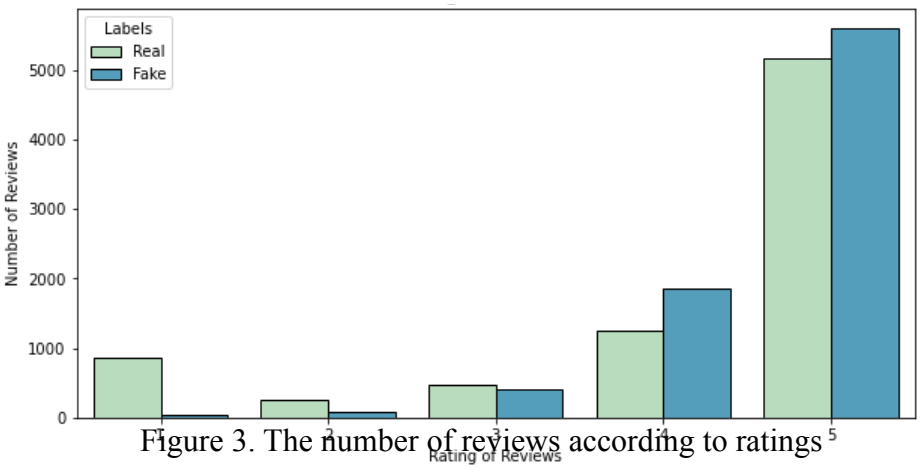

Figure 3. The number of reviews according to ratings

- 52.1% of the 5-star reviews are fake, while 94.7% of 1-star reviews are real (See Figure 4). Except for three stars, there is a statistically significant difference between fake and real reviews. Real reviews have fewer stars, and fake reviews have more stars. Therefore, it can be said that fake reviews give more stars to applications than real ones.

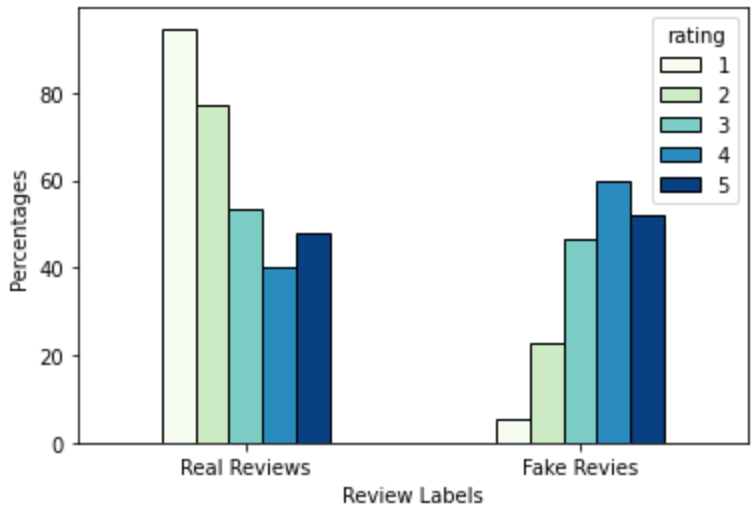

Figure 4. The percentages of reviews according to rating

- For two reasons, applications can get fake reviews: they want to promote their application or sabotage rival applications. For this reason, we want to explore fake review distribution each year after 2014. As seen in Figure 5, fake reviews are usually used to promote the application.

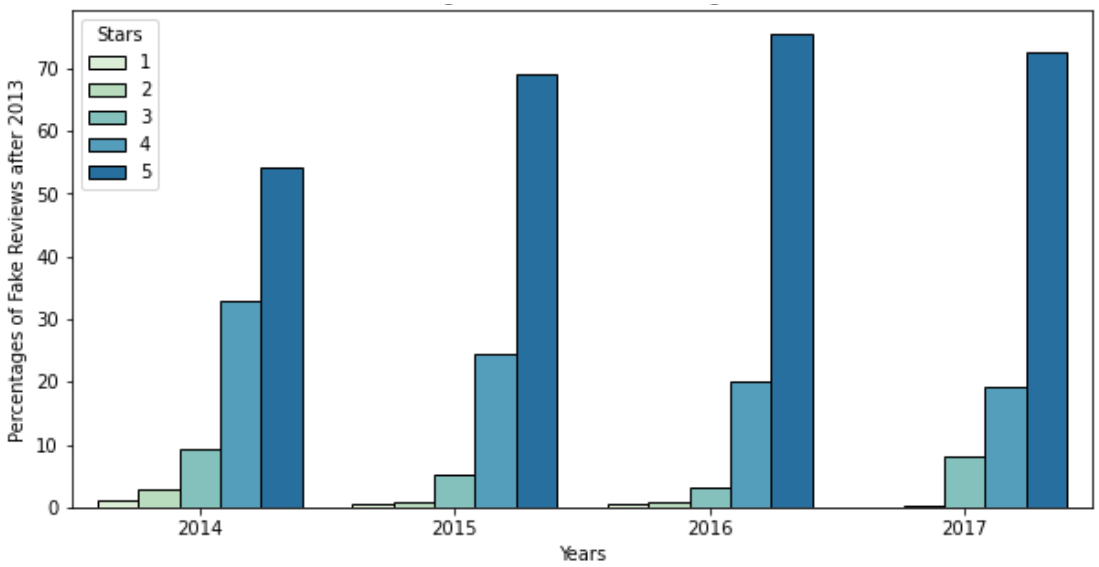

Figure 5. Percentages of fake reviews after 2013

- As seen in the user analysis section, there is no significant difference between the average length of the reviews of fake and real users. However, the review body length is longer in fake reviews ($\overline{X}$=121.18) than in real ones ($\overline{X}$=111.83). Also, the lengths' averages are close; however, this difference is statistically significant ($t = -5.19, p < .05$). Figure 6 shows both reviews' distribution is skewed to the right. The range of real reviews is more than fake ones.

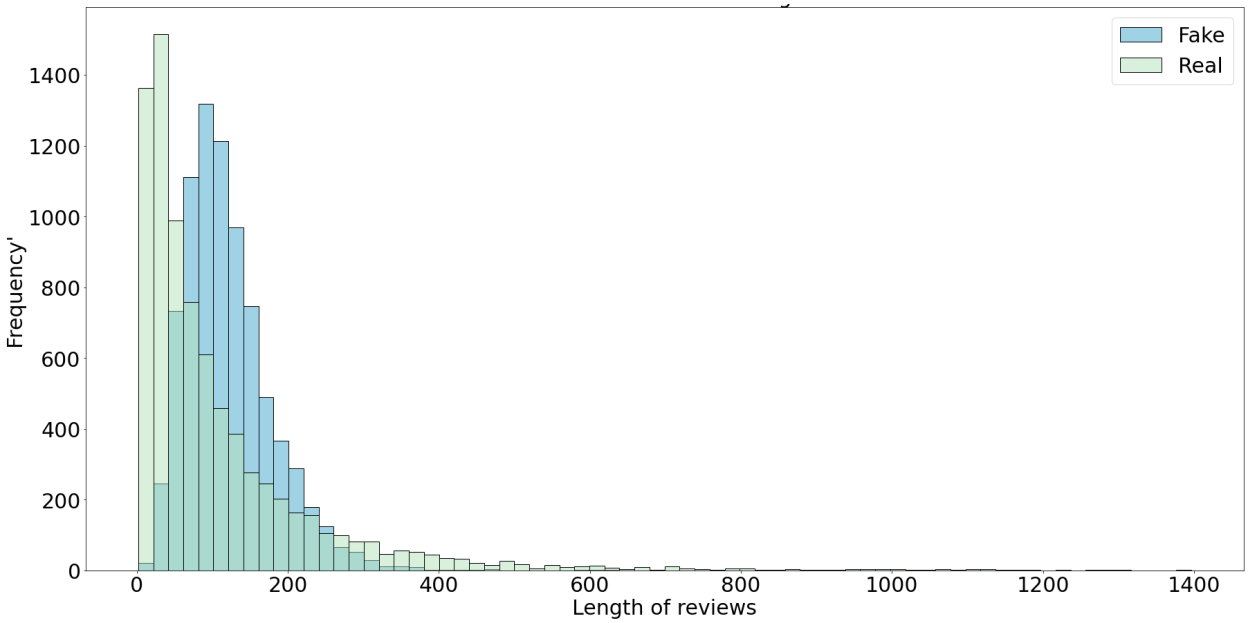

Figure 6. Distribution of review lengths

- Interestingly, the number of real reviews (n = 2809, 35.1%) is much higher when the review length is 40 or less. The 3.15% of fake reviews' lengths are 40 or less. The number of real reviews is more than 11 times fake ones. Also, this difference is statistically significant (X^2 (1, N = 3161) = 2135.98, p = .00). It can be said that fake reviews may be written longer to be more convincing, but in real life, reviews are shorter.

Application Analysis

Firstly, the fake review percentages are calculated for all applications (n=5624). 1822 apps' fake review percentage is 100%, whereas 3734 applications' fake review percentage is 0%. These two groups of apps can be easily labeled as reliable and unreliable. The rest 61 app's percentages vary between 3.8% and 97.1%. I determined the criteria for labeling apps: "If an app has at least one fake review, this app is unreliable." As a result, 3734 (66.3%) of 5624 apps are

reliable, and the other 1890 (33.6%) apps are unreliable. The followings are the main findings of the application analysis.

- According to rating percentages of applications, more than half of applications' reviews have 5 stars. 66.7% of unreliable applications' reviews have 5 stars, % and 6.3 have 1 star. (Figure 7). Also, reliable applications (12.5%) have more 1-star reviews than unreliable applications (6.3%).

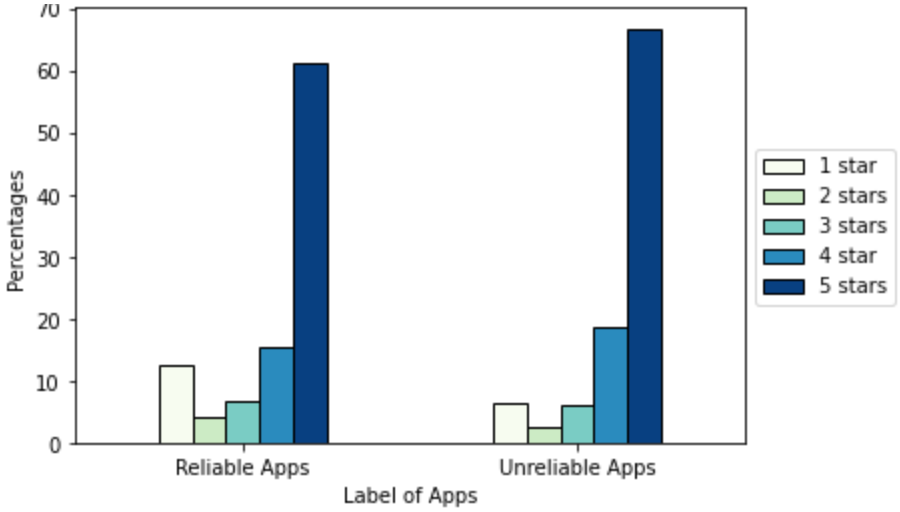

Figure 7. The average percentage of ratings according to applications

- Reliable applications ($\bar{X} = 10519$) have more reviews than unreliable applications ($\bar{X} = 1418$). This difference is statistically significant ($t = 15.7$, $p < .05$).

- The distribution of applications according to the price is presented in Figure 8. The average price of reliable applications is $0.87, and the average price of unreliable applications is $0.14. This difference is not statistically significant ($t = 1.91$, $p = .055$).

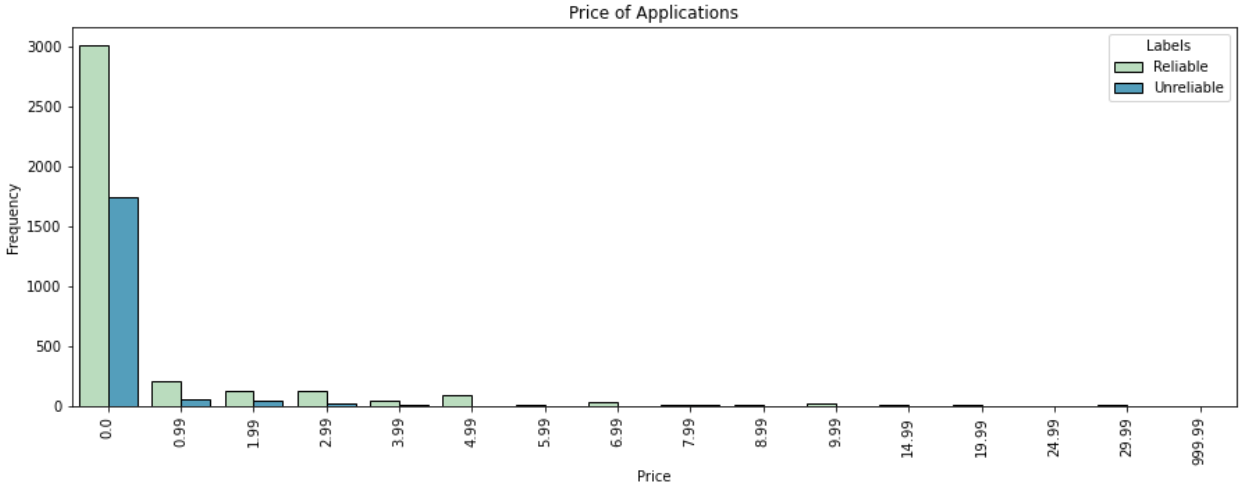

Figure 8. The frequency of applications according to the price

- As seen in Figure 9, most of the applications are free. The number of reliable and unreliable free apps is 3012 (54.1%) and 1746 (31.4%), respectively. Also, 14.4% of the apps are paid. This difference is statistically significant (X^2 (1, N = 5563) = 138.92, p < .05))

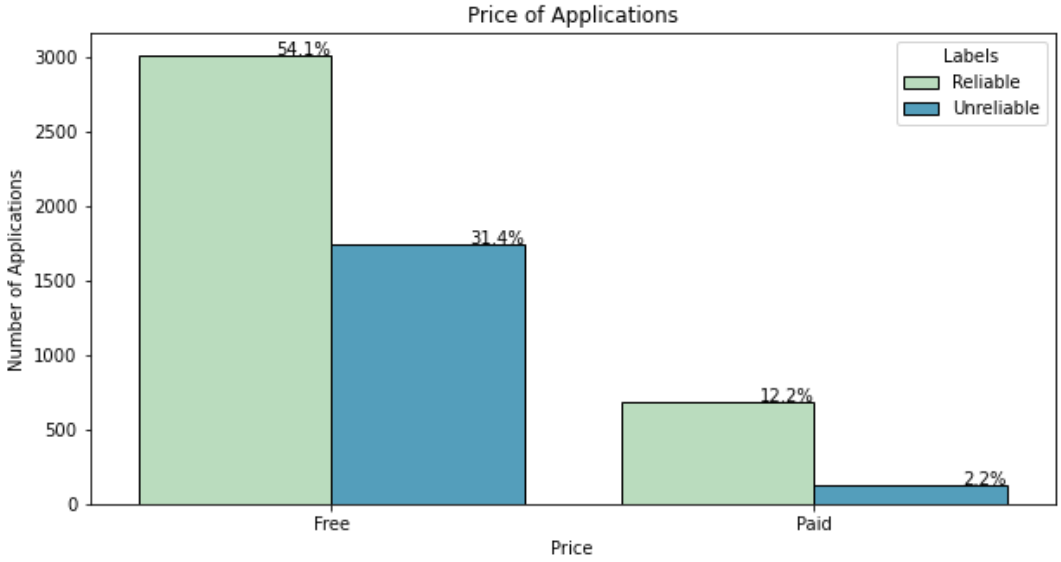

Figure 9. The frequency of free and paid applications

- The most common primary genre of applications can be listed as follows: Games (n = 2576), entertainment (n = 354), photo & video (n = 350), utilities (n = 256), social networking (n = 233), health & fitness (n = 223), productivity (n = 212), lifestyle (n = 192), education (n = 180) and etc. There are two genres whose app frequency is so low. Calculating their unreliable percentage is not fair. Therefore, the 'Magazines & Newspapers' and 'Stickers' genres are excluded.
- Figure 10 shows the top 10 genres with the highest percentage of unreliable applications. The education genre has the most (50.6%) unreliable applications. Sport (47.3%), navigation (43.3%), food & drink (39.6%) and games (38.3%) follows. For these five genres, the unreliable app percentage is higher than the total unreliable app percentage (33.6%). These differences in the top five apps are statistically significant.

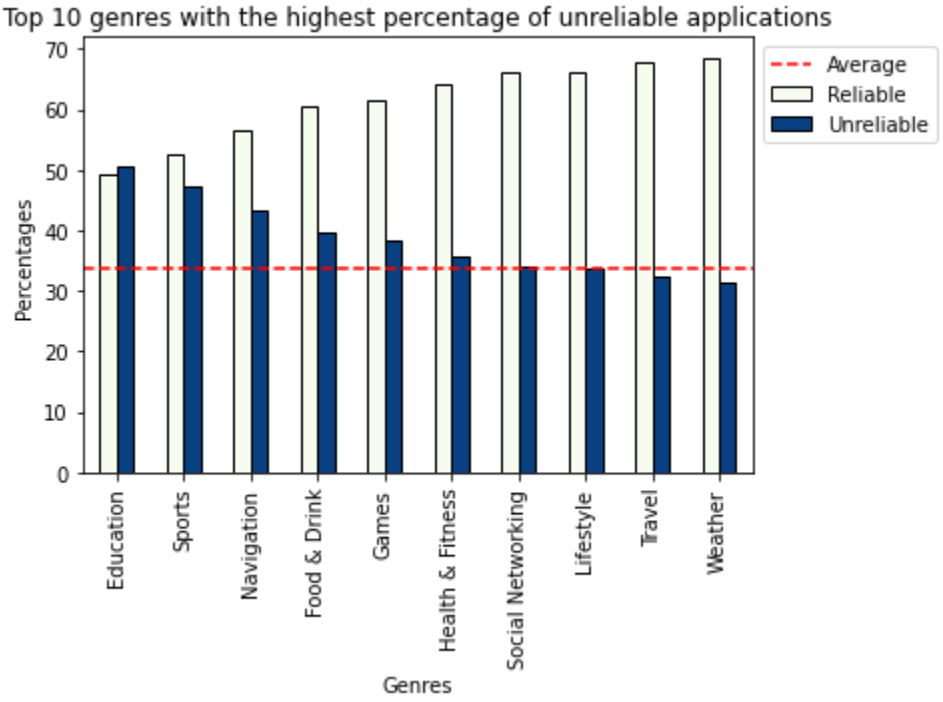

Figure 10. Top 10 genres with the highest percentage of unreliable applications

- As seen in Figure 11, as the version -except 0.0- increases, the number of applications decreases. In all versions, there are fewer unreliable applications.

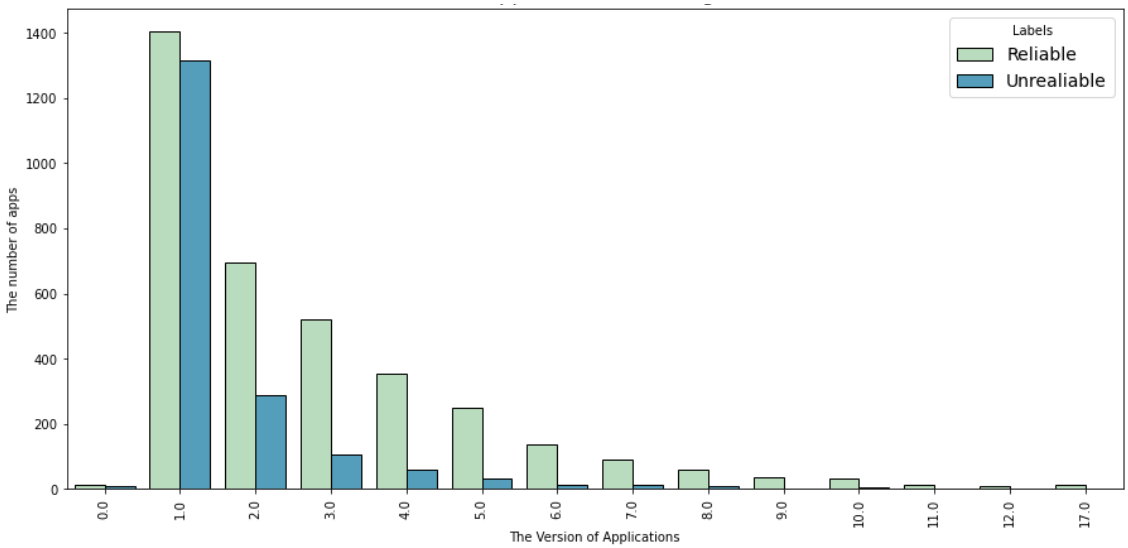

Figure 12. The number of applications according to versions

- Applications' versions' unreliable app percentages were examined, and Figure 12 shows each version's percentages of reliable and unreliable apps. In version 1.0, the rate of unreliable apps is the highest. Also, in 0.0 and 1.0 versions, the unreliable app percentage is higher than average.

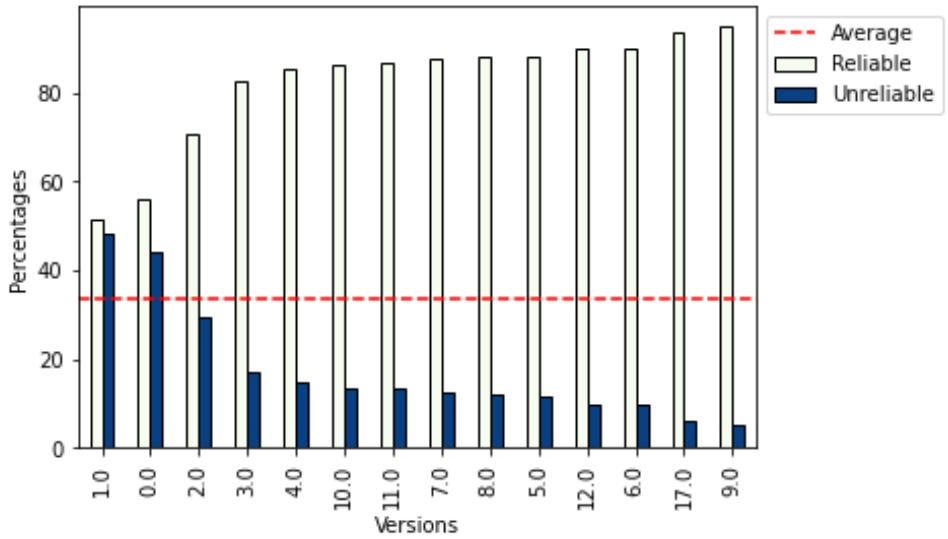

Figure 12. App versions with the highest percentage of unreliable applications

- The initial and current release time difference can indicate how well the application is established. Figure 12 shows the distribution of applications according to the time difference. While the time difference in reliable applications is 1213.9 days, the average of unreliable applications is 354.4. This difference is statistically significant ($t = 41.56$, $p = 0.00$). Therefore, it can be said that reliable applications are long-running.

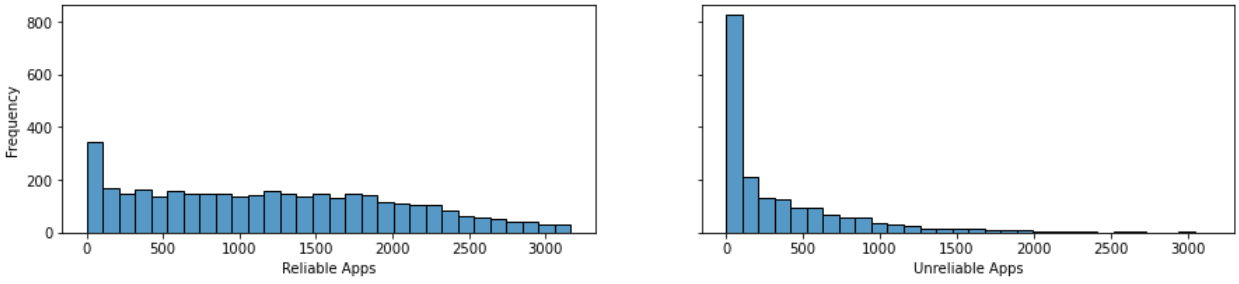

Figure 13. The distribution of applications according to time difference

- Besides review ratings, all rating averages of applications are in the dataset. The average rating differs from 0 to 5 with 0.5 intervals. The distribution of apps' average ratings is different from review ratings. While real users give more 1-star reviews, unreliable apps have fewer average ratings (Figure 14). It can be said that after a while, their app rating starts to show its defects.

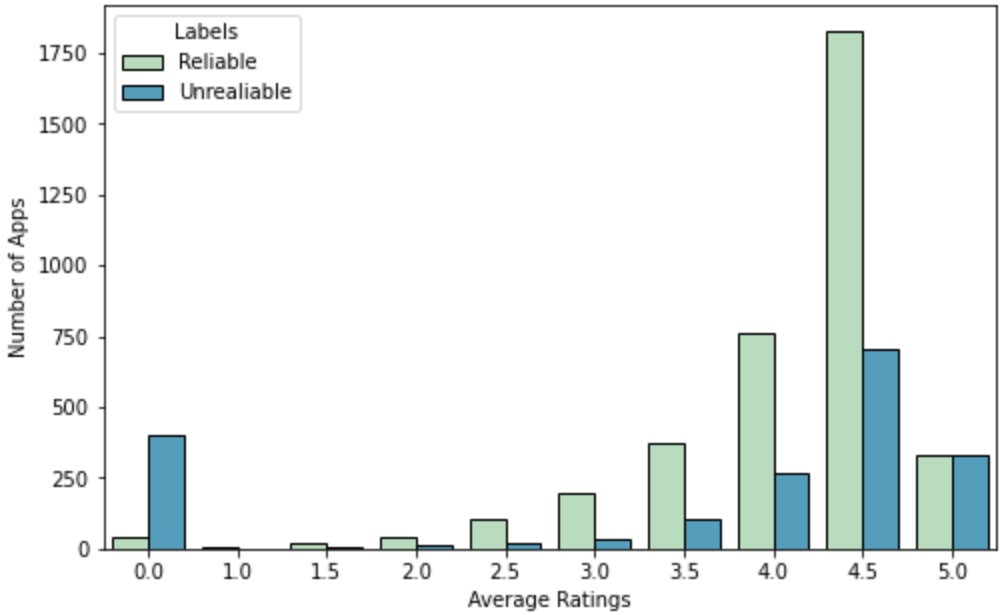

Figure 14. The number of applications according to apps' average rating

Pre-Processing

The dataset has two textual features which can be used in models: Review body and App description. The review title and body have been combined into one feature. There are 148 NaNs in the app description feature. These NaNs have been accepted as blank text (‘ ’).

Firstly, additional features related to textual data, such as the number of words, punctuation, hashtags, etc., have been generated. The language of the review text and descriptions has been detected. 94.2% of review text and 98.7% of descriptions are written in English. The followings are used for text cleaning:

- Converting to lowercase,
- Punctuation, HTML, emoji, URL, white spaces, text in the square brackets, and numbers have been removed,

After cleaning, the textual data in other languages have been translated into English. Also, stopwords and short words have been removed. Tokenization and stemming have been used for the TF IDF vectorizer. The cosine similarity has been calculated to detect the similarity between the review text and app description. There are two cases in which both text and description are blank. For these cases, cosine similarity has been defined as 0.001.

Word cloud has been generated for the review body (See Figure 15). As seen in Figure 15, words such as app, game, really, love, and use, have come to the forefront in the word cloud.

Figure 15. Word cloud of all review text

For encoding textual data, we are implementing a pre-trained BERT model, which is about the context of a word in a sentence. It is a good idea to use so because Bert's embeddings have been trained on huge text data beyond what we could accomplish with this small dataset of reviews. It is, therefore, much more efficient and accurate. Bricken (2021) has found that heavy text data cleaning works worse when input into a BERT model because this contextual information is lost. Therefore, we have used the raw form of textual data (‘text’ and ‘description’). After converting sentences into vectors, we downloaded them as .csv files.

Besides sentence embedding, we want to use word embedding to compare the results of models with text. So we used Glove embedding to convert words into vectors.

Finally, some features have been cleaned before using them in modeling. For instance, the 'time_diff_release_post' feature is in DateTime, only days have been extracted from this feature. The 'user_account_usage_days' feature in seconds. This feature has been converted into days. After the final check, data has been downloaded as a .csv file.

Modeling

Four dataframes will be used in the model: (i) Review text vectors - BERT, (ii) Review text vectors - Glove, (iii) app description vectors, and (iv) features related to reviews, users, and apps. After loading data, these have been split (80% train and 20% test) with the same seed.

Machine learning algorithms require the input data as numeric. Therefore, we have to encode some features. Several categorical variables need to be encoded in features related to reviews, users, and app data. For instance, the ‘reviews’ feature has been encoded by get_dummies, which means we have 0 and 1 for each rating. The following categorical variables have been encoded by using target encoding.

- review_year
- version_major
- primaryGenre
- lang_text
- lang_desc
- trackContentRating
- minimumOsVersion

Also, most machine learning algorithms do not accept null values, so imputing comes into play. Before imputing, we kept a copy of the data. We used three impute methods:

- **Genres of apps:** In raw data, besides the primary genre feature, there was a feature showing all genres related to applications. In the data wrangling notebook, new binary columns were generated for all genres. There are 148 NaNs in these columns. These NaNs are imputed with 0 because there is no information about app genres.

- **Continuous variables:** A total of 7 features with NaNs have been imputed with mean. Some means (counts, days, numbers) convert into integers.

● **Discrete variables:** Four features are imputed with the most frequent value.

We have built different models with three different inputs:

(i) Merge of three dataframes: Review text (BERT) + Description text (BERT) + All features,

(ii) Review text: Review text (BERT),

(iii) Review text: Review text (GloVe)

This is because we may not always have other features related to users, reviews, and apps. We want to see the model results with just review text. Therefore, we will use review text with both BERT and GloVe embedding. Table 1 shows the evaluation metrics of all models.

Table 1. Performance of models

#	Model	Input	AUC	Accuracy	F1	Precision	Recall	Training Time (s)	Prediction Time (s)
1	XGBoost classifier	all_features	0.987	0.987	0.987	0.987	0.987	41.80	41.88
2	Random Forest	all_features	0.984	0.984	0.984	0.984	0.984	65.06	65.20
3	Gradient Boosting	all_features	0.980	0.980	0.980	0.980	0.980	638.67	638.88
4	AdaBoost Classifier	all_features	0.980	0.980	0.980	0.980	0.980	125.39	125.98
5	Decision Tree	all_features	0.967	0.968	0.967	0.968	0.967	96.63	96.68
6	Multinominal Naive Bayes	all_features	0.891	0.891	0.891	0.893	0.891	0.08	0.09
7	Logistic Regression	all_features	0.863	0.862	0.862	0.867	0.863	1.55	1.64
8	Logistic Regression	review_Bert	0.824	0.824	0.824	0.825	0.824	1.47	1.49
9	XGBoost classifier	review_Bert	0.813	0.812	0.812	0.813	0.813	33.29	33.32
10	Gradient Boosting	review_Bert	0.799	0.799	0.799	0.801	0.799	359.68	359.80

11	Random Forest	review_Bert	0.786	0.786	0.786	0.787	0.786	45.02	45.16
12	AdaBoost Classifier	review_Bert	0.774	0.773	0.773	0.775	0.774	72.76	72.99
13	Random Forest	review_GloVe	0.756	0.756	0.756	0.757	0.756	8.52	8.60
14	Gradient Boosting	review_GloVe	0.753	0.753	0.752	0.756	0.753	12.02	12.03
15	XGBoost classifier	review_GloVe	0.747	0.747	0.746	0.748	0.747	1.92	1.93
16	Naive Bayes	review_Bert	0.743	0.742	0.741	0.749	0.743	0.20	0.25
17	Naive Bayes	all_features	0.739	0.737	0.719	0.823	0.739	0.91	1.04
18	Multinominal Naive Bayes	review_Bert	0.732	0.731	0.729	0.740	0.732	0.02	0.03
19	AdaBoost Classifier	review_GloVe	0.722	0.721	0.721	0.724	0.722	2.50	2.53
20	Naive Bayes	review_GloVe	0.689	0.688	0.682	0.706	0.689	0.01	0.01
21	Decision Tree	review_text	0.683	0.683	0.683	0.683	0.683	23.06	23.07
22	Logistic Regression	review_GloVe	0.678	0.678	0.678	0.679	0.678	0.07	0.07
23	Decision Tree	review_GloVe	0.636	0.636	0.636	0.636	0.636	0.70	0.70
24	Multinominal Naive Bayes	review_GloVe	0.615	0.614	0.613	0.616	0.615	0.00	0.00

Overall, there are 4 models with all features which give more than 98% accuracy on the unseen test data: Random Forest, Gradient Boosting, Ada Boost Classifier, and XGBoost Classifier. These performances are great for detecting fake reviews. Figure 16 shows the 30 most important features in the Random Forest - All Features - model. The three most important features are the total number of reviews an app receives, the total number of reviews provided by a user, and the total number of app ratings. Also, 29 out of 30 features come from data related to users, reviews, and apps. Therefore, it can be said that these features play a significant role in detecting fake reviews.

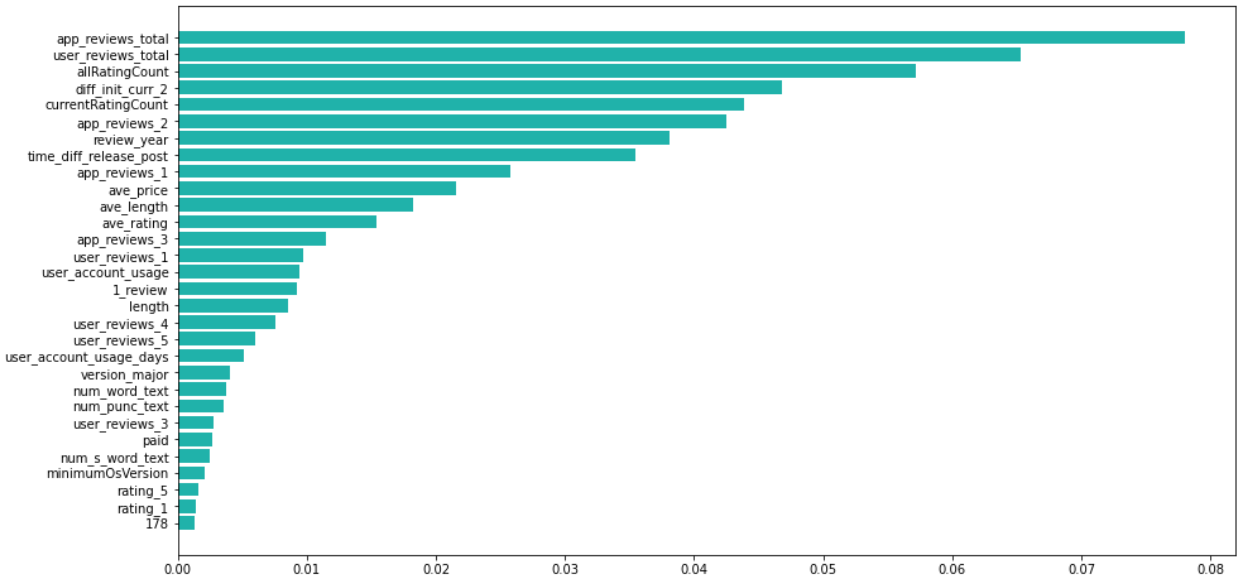

Figure 16. Random Forest -All Features - Model's Feature Importance Plot.

Unfortunately, models with just review text gave worse metrics than models with all features (see Table 1). Assuming we do not have any data about reviews, users, and apps, we have just reviews in the worst-case scenario. Working on optimizing these models is necessary and useful. For this reason, instead of optimizing models that already predict well, we tried to optimize models with review text.

Hypermeter Tuning for Models with Review Text

Table 2 shows the metrics after hypermeter tuning. Compared to the results of the default models, hypermeter tuning cannot improve models' performance much.

Table 2. Results of optimization

Model	Optimization Type	Mean AUC	Performance on Unseen Test
Logistic Regression	Grid Search	.901	.824
XGBoost Classifier	Random Search	.884	.811
Gradient Boosting		.879	0.811
Random Forest		.858	.784

Conclusion

We have aimed to build a model that filters the App Store reviews as fake or real in this project. After experimenting with different datasets, we found that the XGBoost classifier identifies fake reviews with an AUC/ROC value of 98%. That score is too good for real-world data. In the project, the models with different inputs indicate essential points in detecting fake reviews.

Firstly, the models with additional features perform better than those with text input. The information such as total reviews, ratings, and how long a user has been subscribed to App Store are good indicators of fake or real reviews. It can be said that information about users, applications, and reviews should be considered in fake review detection.

Secondly, the models with just review text data do not perform as well as models with additional features. Therefore, it can be said that fake reviews seem authentic, and it is tough to recognize fake reviews without user and app behavior. Also, the vectorization method plays an essential role in model performance for these models. We have used both GloVe word embedding and BERT sentence embedding. BERT performed better because it considers the context of a word in reviews. Also, BERT vectorization has taken many different elements in reviews, such as punctuation and emoji, into account. This may have helped identify genuine items in reviews. Consequently, factors such as the business problem, the aim of the model, and text type should be considered when deciding which vectorization method to use.

Deep learning algorithms could be deployed with the same dataset in further studies. Also, the winner XGBoost model could be tested on review data from e-commerce platforms like Amazon, Yelp.

References

Martens, D., & Maalej, W. (2019). Towards understanding and detecting fake reviews in app stores. Empirical Software Engineering, 24(6), 3316-3355.

Credits

Thanks to Ajith Patnaik for being an amazing mentor with his support, guidance, and advice, and Walid Maalej for sharing the dataset.