Aptoide

Summer Internships

2016

App Moderation of Mature Content

Author:
Diogo Ferreira

Menthor:
Ehsan Nia

August 12, 2016

Contents

1 Introduction

The main goal of this project on Aptoide is to write an API that is able to detect mature content on apps in the Aptoide store.

With the images (icons and screenshots) and information about the app (title, age, description, category, ...) this program should detect if the app has mature content or not, with at most 6% of accuracy error.

Along this report, we will show you what was our method to solve this problem. We will also show some tests done with our databases. If you want to read the whole description of the tests, the excel file attached has all the detailed results.

2 Literature Review

2.1 University Papers

There are some University papers with theoretical algorithms or some research about nudity detection in images. From those, *"An Algorithm For Nudity Detection"* ([Ap-Apid, Rigan, De La Salle University]) has an algorithm that is used by many softwares to detect nudity content. Briefly, this algorithm goes through all the pixels and gets his RGB, HSV and normalized RGB values. Then, according to a skin distribution model, determines if the pixel color satisfies the parameters of being skin. With all the pixels labelled, calculates the percentage of skin pixels and connects them in regions. Then, all the processing is done based on those regions. *Nude.js* ([WIED Patrick, 2010]) uses this algorithm as starting point for it's analysis.

"Detection of Pornographic Digital Images" ([Jorge Basilio, Gualberto Torres]) also has an extensive study about the color models used for skin detection, but it fails short on the results and on the algorithm used (just based on the percentage of skin that an image has).

"Machine Learning Application On Detecting Nudity In Images" ([Yong Lin, Yujun Wu]) has more tests about the regionization of skin but, most importantly, it uses machine learning to do so, with different classifiers.

"Automatic detection of images containing nudity" ([Andreas Carlsson, Andreas Erikson]) uses an artificial neural network in extensive testing to detect not only nudity on

images, but also face detection.

Finally, *"Illustration2Vec"* ([Masaki Saito, Yusuke Matsui]) it's an open-source software based on a convolutional neural network that detects illustrations and returns semantic vectors with it's percentage. It was trained with over 1.3 millions of illustrations using *Nearest Neighbour Search*. Since two of the vectors are "explicit" and "safe", could also be used for analysing the explicitness of images.

2.2 Software research

Initially, the desired was to find an open-source algorithm that could recognize images with nude bodies and improve it. After some research, we found it really hard to find an algorithm that could satisfy our needs.

At the end, we had two different open-source algorithms for image categorization. The first one, *Nude.js* ([WIED Patrick, 2010]), was built upon algorithms of image detection without any neural network. The second one, *Illustration2Vec* ([Saito, Masaki and Matsui, Yusuke, 2015]), was a convolutional neural network system for illustrations (explained thoroughly in chapter 4). We also compared them with three of the best web api's (not open-source) for image nudity classification (Table 1).

Table 1: Initial comparison of algorithms for nudity classification. As we can see, Nude.js is clearly the worst performer algorithm. The other four are more balanced, with a slightly advantage of *PicPurify* and *SightEngine*, followed close by *Illustration2Vector*. We can see that even though *Illustration2Vector* is not the best algorithm that we tested, is comparable with the best ones available.

	True Positives	True Negatives	False Positives	False Negatives	Accuracy % on explicit data	Accuracy % on Non-explicit data
Nude.js	4	31	0	28	12,5	100
Illustration2Vector	23	29	2	9	71,9	93,5
PicPurify	25	31	0	7	78,8	100
#IsItNude	31	18	13	1	96,9	58,1
SightEngine	25	31	0	7	78,8	100

To improve the final accuracy, we decided to also analyse information about the text inserted (Chapter 5).

3 Creation of Dataset

To train and test the software, we would need a database with images and information about the apps. Initially, we created a hand-picked database with easily recognizable images divided by explicit/non-explicit content.

When we moved to automated tests, we decided that we would have to have a bigger database, with images and information of the *Aptoide Store*. So, we wrote automated tests to download information and images based on the *Aptoide Web Services*.

The first database we build with the automated tests had 693 apps with explicit content (17%) and 3363 apps with non-explicit content (83%). The big database that we build to do the final training and tests had 1772 apps with explicit content (17%) and 8153 apps with non-explicit content (83%).

Unfortunately, the final results with the big database resulted on a lower accuracy, which make us repeat all the tests to choose the best classifier and the best parameters, as we had done to the previous database.

3.1 How was it done

3.1.1 Get List of Id's

The script *get_list_id* receives as a parameter the search you want to make on *Aptoide Store* and the number of apps that you want to search (optional). For example, if you want to search for car games, you can pass as a parameter: *car games*. If you only want the first 50 results, you can pass parameters: *car games 50*.

The script will connect to the *Aptoide Web Service* and to a database named *"app_info.db"*. Then, it will save on a table called *"crawl_list"* the id, the search query, the (Unix) date and a '0' on the row "is_processed".

If you want to erase all the content on the *"crawl_list"* table, just call the script with '-r' as an argument.

3.1.2 Get Information of App's

This script receives as an optional parameter the number of app's you want to download. If no parameter is given, it will download them all.

The script will connect to a database name *"app_info.db"*, and it will fetch from the table *"crawl_list"* all the id's that were not processed. For each one of them, it will save on the table *"app_info"* the app information (id, title, age, description, wurl, category and repository). It will also download all the images (icons and screenshots) and it will divide them as explicit content/non-explicit content, given its minimum age.

If you want to erase all the content on the *"app_info"* table, just call the script with '-r' as an argument.

3.1.3 Other scripts and important data

After the databases are done, it is needed to add another column to the table *"app_info"*, called *"mature"*. The only way to check if an app really has mature content or not is to check it one by one. The same for the images, that should be divided in explicit and non-explicit content, for both icons and screenshots.

There are two databases already divided, one with 4000 apps and other with 9925 apps. They were used for training and testing. They are not in the repository because they were too large to fit.

There are also other scripts called *"misclassified_text"*, *"misclassified_images"* and *"misclassified_apps"*. They connect to a database count how many apps are misclassified, between others metrics, for future reference (Table 2).

Table 2: Misclassified Apps. More than 10% of our sample are misclassified as non-explicit content (from a sample of 8153 non-explicit and 1772 explicit apps)

True Positives	True Negatives	False Positives	False Negatives	Accuracy % on explicit data	Accuracy % on non-explicit data
1591	7985	168	181	89,7	97,9

4 Detection of mature images

Our choice relied on an open-source software with machine learning, *Illustration2Vec* ([Saito, Masaki and Matsui, Yusuke, 2015]). The model came pre-trained with 1.3 million illustrations of Japanese animes and illustrations. On the first tests, the total average percent was really high, what encouraged us to choose this software. Besides the final test results in accuracy percentage on explicit data were

of only 25%, it is still by far the best software that we tested to detect explicit images.

4.1 Tests

The tests done are represented on the table 3.

Table 3: Image categorization tests. As we can see, the number of false negatives is still high, but the false positives are lower.

	True Positives	True Negatives	False Positives	False Negatives	Accuracy Percent
Icons	373	17327	253	724	94,27
Screenshots	1616	57094	491	3075	94,77

On the script *"analyse_image"*, the function *"analyse_explicit"* receives as an argument the *"Illustration2Vec"* model and the image directory for analysis. It return a tuple with two elements: (("explicit", explicitness rate), ("safe", safeness rate)). The sum of the two rates are NOT equal to one. The rate left is the questionable rate.

5 Detection of mature text

To improve the precision on explicit data we had to build a machine learning software for the textual information present on the apps. We relied on the python libraries *NLTK* ([Steven Bird, Ewan Klein, and Edward Loper, 2000]) for text processing and *Scikit-learn* ([Pedregosa, F. and Varoquaux, G. and Gramfort, 2011]) for classify our language as explicit content or non-explicit content.

On the final clasifier choosing (Table 5), the features used to classify our language were the description and the category. We tested other features, as the title or the description size, but the accuracy was lower with both of them. We also tried to add the minimum age as a feature, which gave us excellent results, but after all the tests we had to take a step back and realise we were overfitting the model (If the age was a feature, every app with minimum age as 18 would return as explicit. On a real world application, we would only check if it was mature content all the apps where the minimum age was under 18. So, most of them would return false) (Table 4).

Table 4: Feature Choosing. As we can see, every time we add age to the result, the accuracy percent is higher, but we had to remove the age as a feature. The biggest accuracy without the age as a feature is the description and the category.

Description	**Description and Age**	**Description, size of description, title, age and category**	**Description and Category**	**Description, Category and Age**
92,2	94,8	92,4	93,2	95,2

5.1 Classifier Choosing

Table 5: Initial tests with all the classifiers (from a sample with 8153 non-explicit and 1772 explicit apps. 500 were used for testing, all the others for training). Random Forest and Logistic Regression had the best accuracy, so they were used for extensive tests.

Classifiers	True Positives	True Negatives	False Positives	False Negatives	Accuracy %
Naive Bayes	54	372	30	44	85,2
Random Forest	69	382	20	29	90,2
Ada Boost	49	386	16	49	87
Multinomial	48	376	26	50	84,8
Decision Tree	71	371	31	27	88,4
Bernoulli	43	385	17	55	85,6
Logistic Regression	59	387	15	39	89,2
SGD Classifier	70	365	37	28	87
SVC	0	402	0	98	80,4
Linear SVC	72	373	29	26	89
Nu SVC	57	368	34	41	85
Extra Trees	67	376	26	29	89

Table 6: 5-Fold Cross Validator tests on Ada Boost Classifier for the text categorization.

Ada Boost	True Positives	True Negatives	False Positives	False Negatives	Accuracy %
1	280	1605	25	74	95
2	271	1589	42	83	93,7
3	270	1581	49	85	93,2
4	281	1593	38	73	94,4
5	270	1587	44	85	93,5
Average	274,4	1591	39,6	80	93,96

Table 7: 5-Fold Cross Validator tests on Logistic Regression Classifier for the text categorization.

Logistic Regression	True Positives	True Negatives	False Positives	False Negatives	Accuracy %
1	278	1608	22	76	95,1
2	265	1592	39	89	93,5
3	266	1588	42	89	93,4
4	277	1599	32	77	94,5
5	268	1594	37	87	93,7
Average	270,8	1596,2	34,4	83,6	94,04

After extensive parameter otimization and other tests, *Logistic Regression* was the chosen classifier.

5.2 Scripts

There are two test files for the text categorization. The *"Text_categorization-tests"* was used to test all the classifiers and compare them one by one. The *"Text_categorization_kfold"* script uses the *k-fold cross validation* technique to measure with more accuracy a classifier result.

The script *"Text_categorization_full_training"* does all the training for the model and saves it in a python serialized object using *Pickle*, named "model_info.pickle".

The script *"Text_categorization"* has a function *"text_cat"* that receives as argument the description, the size of the description, the category and the age and it will return a classifier tuple with the probability of being explicit content or non-explicit content

6 Detection of Mature Apps

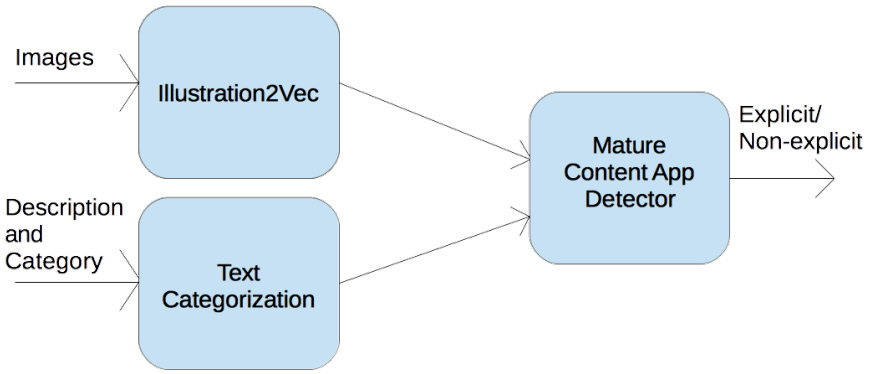

Figure 1: Example of the process for detection of mature apps.

Finally, the mature detector for the apps is by itself a machine-learning algorithm just like what we used on the text information, but with a different classifier. The features are the percentage of explicitness and safety that the text detector and the image detector return (Figure 1).

6.1 Classifier Choosing

Table 8: Initial tests with all the classifiers (from a sample with 8153 non-explicit and 1772 explicit apps. 500 were used for testing, all the others for training) for the mature app detector.

Classifiers	True Positives	True Negatives	False Positives	False Negatives	Accuracy %
Naive Bayes	143	304	539	14	44,7
Random Forest	122	823	20	35	94,5
Ada Boost	123	834	9	34	95,7
Multinomial	64	838	5	93	90,2
Decision Tree	122	790	53	93	91,2
Bernoulli	0	843	0	157	84,3
Logistic Regression	125	830	13	32	95,5
SGD Classifier	124	3834	9	33	95,8
SVC	125	835	8	32	96
Linear SVC	235	829	14	32	95,4
Nu SVC	115	794	49	42	90,9
Extra Trees	127	817	26	30	94,4
Ridge	155	340	503	2	49,5
Lasso	157	0	843	0	15,7
Elastic Net	157	0	843	0	15,7
SGD Regressor	155	316	527	2	47,1

Table 9: 5-Fold Cross Validator tests on SGD Classifier for the mature app detector.

SGD Classifier	True Positives	True Negatives	False Positives	False Negatives	Accuracy %
1	277	1612	18	77	95,2
2	227	1617	14	127	92,9
3	264	1587	43	91	93,2
4	254	1603	28	100	93,5
5	252	1609	22	103	93,7
Average	254,8	1605,6	25	99,6	93,7

Table 10: 5-Fold Cross Validator tests on Ada Boost Classifier for the mature app detector.

Ada Boost	True Positives	True Negatives	False Positives	False Negatives	Accuracy %
1	280	1605	25	74	95
2	271	1589	42	83	93,7
3	270	1581	49	85	93,2
4	281	1593	38	73	94,4
5	270	1587	44	85	93,5
Average	274,4	1591	39,6	80	93,96

Table 11: 5-Fold Cross Validator tests on Logistic Regression Classifier for the mature app detector.

Logistic Regression	True Positives	True Negatives	False Positives	False Negatives	Accuracy %
1	278	1608	22	76	95,1
2	265	1592	39	89	93,5
3	266	1588	42	89	93,4
4	277	1599	32	77	94,5
5	268	1594	37	87	93,7
Average	270,8	1596,2	34,4	83,6	94,04

Table 12: 5-Fold Cross Validator tests on SVC Classifier for the mature app detector.

SVC	True Positives	True Negatives	False Positives	False Negatives	Accuracy %
1	277	1617	13	77	95,5
2	262	1600	31	92	93,8
3	263	1593	37	92	93,5
4	270	1602	29	84	94,3
5	267	1599	32	88	94
Average	267,8	28,4	1602,2	86,6	94,22

Table 13: 5-Fold Cross Validator tests on Linear SVC Classifier for the mature app detector.

Linear SVC	True Positives	True Negatives	False Positives	False Negatives	Accuracy %
1	279	1608	22	75	95,1
2	266	1591	40	88	93,6
3	266	1590	40	89	93,5
4	276	1599	32	78	94,4
5	267	1592	39	88	93,6
Average	270,8	1596	34,6	83,6	94,04

In the end, SVC was the classifier chosen for the mature app detector.

7 Django API and Packaging

At the end of this project, for faster integration with our present system, we decided to advance on building a Django API.

The Django API, once the server is running, is listening at the url 127.0.0.1/detect_mature/id=id_of_app or 127.0.0.1/detect_mature/md5sum=md5_of_app. Once a request is made, the API analyses the app information and returns JSON information about the explicitness of the app.

The analysis result is saved on a database for caching the results. If your desire is to reload the cache with new information, even if the app was already analysed, just add */reload=1* to the url. For example, a valid url could be: *127.0.0.1:8000/detect_mature/id=id_of_app/reload=1*.

The format of the JSON result can be found on the image below.

Figure 2: The result shows the id and md5 of the app processed, its result, the time it took to analyse it and the status


```
{
    app_id: 20323499,
    app_md5: "cdfe7e6dbdaac8b6831a9d63112ef654",
    mature_content: "no",
    status: "OK",
    time: "0:00:00.092123"
}
```


8 Configurations

There is a file in the project directory called *"config.json"*. That file has some configurations to be changed if necessary. It has the directories of the static files used (the models and the database), as well as the location of the images. If the *"local_or_web_images"* is set to "local", the script will try to parse the images locally, using the path in *"local_image_path_prefix"*. If the *"local_or_web_images"* is set to "web", it will download the images from the web.

It also allows the user to choose between synchronous and asynchronous requests (*"synchronous_or_asynchronous"*). If it is set to "synchronous", the user will have to wait a few seconds to have an answer from the server. If is set to "asynchronous", the user will see a webpage with json content but without the final result. The server will be processing the request internally. When it finishes, it will open a webpage defined in *"asynchronous_dir"* and it will add to the url the id or the md5 and the result "True" or "False", depending on the result of the analysis. If the result was already cached and cache reload is not set, the server will immediately send the result.

9 Database for final results

Every time an app is analysed, the final result is saved on a database, as well as the images and description result. The database has four tables.

The "app" table identifies an app by it's md5 sum and for every app generates a database id.

The "text_results" table has four columns: "for_id" (database id of the app), "text_exp" (percentage of explicitness of the textual information), "is_mature" (yes or no) and "external_validator" (by default set to -1. When you want to mannualy validate the answer as yes/no, just change to 0/1).

The "image_results" table has seven columns: "for_id" (database id of the image), url (url or path of the image), "image_safe" (percentage of safeness of the image), "image_exp" (percentage of explicitness of the image), "is_mature" (yes or no) and "external_validator" (by default set to -1. When you want to mannualy validate the answer as yes/no, just change to 0/1) and "icon_or_screenshot" (indicating if the image is an icon or a screenshot).

The "final_results" table has five columns: "for_id" (database id of the app), date (UNIX datetime of the request), "exp_per" (percentage of explicitness of the app), "is_mature" (yes or no) and "external_validator" (by default set to -1. When you want to manually validate the answer as yes/no, just change to 0/1).

10 Testing on the server

On the server, we not only installed the project and its dependencies, but also other tools like *"Nginx"* and *"gunicorn"* for improved web responses.

While testing on the server, we ran into some issues. The major one is the time that it takes to analyse an image. While on a local PC it takes about 0.5 seconds to analyse an image, on the server it takes up to 2 seconds. Because of this, we had to increase the timeout to prevent from a "Bad Gateway" error. We believe that is because the server was running on a virtual machine, since we did some testing on a virtual machine and the results were very similar.

11 Improvements

The image analysing is the biggest bottleneck of this software. Not only because of the time spend, but because the inaccuracy on explicit images is too high. To solve the time problems in short-term, I would recommend using it on a local PC, since we have proven that it runs much faster than on a virtual machine.

For long-term, the better approach would be to build a model similar to the *Illustration2Vec*, but that we could train and modify the source-code. For that, it is needed a high computing power, but the final accuracy and the time spent analysing the images would improve much more after that rearrangement.

12 Conclusion

On the mature images detector there are two big flaws. The first one is that we can't train the model. We can only use it as it is presented to us, which lead us to the second flaw. The model was trained with illustrations, so it is optimized for illustrations recognition, which can cause the low accuracy percent on explicit data. Besides that, the total accuracy is good and it fits our goal.

The mature text detector has a high accuracy percent, which is satisfying. At anytime, we can train again the model with a larger database or change the features, since it was done top-to-bottom by us.

The same can be said about the final mature detector. The only negative point is that on a regular computer, it can take up to a day to find the feature set and train the model with all the data.

At the end, our model has an estimated accuracy percent of 94,4 %, which is much higher than what we expected to achieve at the begining of the project.

References

[WIED Patrick, 2010] https://www.patrick-wied.at/static/nudejs/ nude.js is a JavaScript implementation of a nudity scanner based on approaches from research papers.

[Saito, Masaki and Matsui, Yusuke, 2015] http://illustration2vec.net/ Illustration2Vec: A Semantic Vector Representation of Illustrations

[Pedregosa, F. and Varoquaux, G. and Gramfort, 2011] http://scikit-learn.org/ Scikit-learn: Machine Learning in Python

[Steven Bird, Ewan Klein, and Edward Loper, 2000] http://www.nltk.org/ NLTK is a leading platform for building Python programs to work with human language data.

[Ap-Apid, Rigan, De La Salle University] An Algorithm for Nudity Detection

[Jorge Basilio, Gualberto Torres] Detection of Pornogrphic Digital Images

[Yong Lin, Yujun Wu] Machine Learning Application on Detecting Nudity in Images

[Andreas Carlsson, Andreas Erikson] Automatic detection of images containing nudity

[Masaki Saito, Yusuke Matsui] Illustration2Vec: A Semantic Vector Representation of Illustrations