Documentation of App Moderation for Detection of Mature Content

Diogo Daniel Soares Ferreira

August 2016

Contents

1 Installation Guide

1.1 Requirements

To run this software, the minimum recommended requirements are:

- Linux, Windows or Mac OS
- 4 GB of RAM
- Intel Core i3 Dual-Core or similar
- 8 GB of Free Disk Space

You will also need:

- Python 2.x (version tested: 2.7.12)
- Django (version tested: 1.9.8)
- BS4 (Beautiful Soup) (version tested: 4.4.1)
- Numpy (version tested: 1.11.1)
- Scipy (version tested: 0.17.1)
- Pillow (version tested: 1.1.7)
- Chainer (MANDATORY version: 1.11.0)
- NLTK (version tested: 3.2.1)
- MatplotLib (version tested: 1.5.1)
- Scikit-Image (version tested: 0.12.3)
- Scikit-Learn (version tested: 0.17)
- LibAtlas (version tested: 3.8.4)

The recommended commands for install the software on an Ubuntu Linux Operative System are:

- Python 2.x - Already on the system
- Django - pip install django
- BS4 (Beautiful Soup) - pip install beautifulsoup4
- Numpy and Scipy and MatplobLib - sudo apt-get install python-numpy python-scipy python-matplotlib ipython ipython-notebook python-pandas python-sympy python-nose
- Pillow - pip install Pillow

- Chainer - apt-get install g++ && pip install chainer==1.11.0
- NLTK - sudo pip install -U nltk
- Scikit-Image - pip install scikit-image
- Scikit-Learn - pip install -U scikit-learn
- LibAtlas - sudo apt-get install libatlas-base-dev

It is also needed to download libraries from NLTK, after all requirements. Best way is run:
import nltk
nltk.download('all')

1.2 Run the server/Analyse an app

For run the server, you need to go to the path aptoide_mature_app_detector/aptoide_mature_app_detector/explicit_content_detector from the command-line and run the default django command: *python manage.py runserver*

The server will start running on the localhost(127.0.0.1:8000). If you want to analyse an app by it's ID, you open the url: 127.0.0.1:8000/detect_mature/id=id_of_app. If you want to analyse an app by it's MD5 Sum, you open the url: 127.0.0.1:8000/detect_mature/md5sum=md5_of_app

The ID of an app can be found on the app page. For example, for the Facebook app, given it's url (http://norwegianstore.store.aptoide.com/app/market/com.facebook.katana/35540499/20347153/Facebook), the ID is 20347153.

MD5 can be not that easily found. If you want to know the MD5 Sum of an app, open the source code of the app url and search for "MD5". However, in some particular cases, it can be more than one "MD5" in the source code, but only one has the right MD5 Sum.

The result is described in the next image.

Figure 1: 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"
}
```


The final result is also saved on a cache database. If you want to refresh the cache with new results, you can also add */reload=1* in front of your url. For example, a valid url could be *127.0.0.1:8000/detect_mature/id=id_of_app/reload=1*.

1.2.1 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.

1.2.2 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).

2 File Organization

The source project starts at the outer file *aptoide_mature_app_detector*. It's where the package files are. These files do not need extensive reading, so we will skip them. We just need to mention *config.json*, which contains the directories for the files. Let's open the file *aptoide_mature_app_detector*. From there, let's open *explicit_content_detector*. This is the directory of the django project. The

server can be run from here. If we open the file *explicit_content_detector*, we can see the default django files. We only changed the *urls.py* to define the url of our project. Let's go back and open the file *API*.

There we can see two of the most important files for this project. First, the *urls.py* with the url's for the views. Then, the *views.py* defines the content of our views.

If we open the file *Explicit_detector*, we can see the main scripts of this project. The scripts that analyse and test the final *mature content app detector* are here, as well as the database for caching the results. From here, we can access the three parts of our project. The *Text_categorization* is the file with the scripts about the text categorization (api, tests and training). The *Illustration2Vector/illustration2vec_master* is the file with the scripts about the image categorization (api and tests). The *API to download database* is the file with the essential scripts to download a new database and to get the metrics of the misclassified information.

3 Most common scenarios

3.1 Analysing mature content

3.1.1 Analyse an Image

If you want to analyse an image, open the path aptoide_mature_app_detector/ aptoide_mature_app_detector/explicit_content_detector/API/Explicit_detector/ Illustration2Vector/illustration2vec_master.

Then, from the command-line, you need to run the script *analyse_image.py* with the image path as command-line argument. For example, if you have an image on the same path called "image.png" and you want to analyse it, just run from the command-line: *python analyse_image.py ./image.png*

3.1.2 Analyse Textual Information

If you want to analyse textual information, open the path aptoide_mature_app_detector/ aptoide_mature_app_detector/explicit_content_detector/API/ Explicit_detector/ Text_categorization.

There, you will find a script called *"Text_categorization.py"*. Open it and, on the last two lines, change the call to the function *text_cat*. The parameters passed by the order are: description, size of description, category and age. You can freely change them. Finally, for seeing the result, you can change the probability on the last line: 'exp' is the explicit probability, 'non' is the non-explicit probability.

When you have everything set up, from the command-line, run: *python Text_categorization.py* and you will see the percentage.

3.2 Training

3.2.1 Text Categorization Training

If you want to train textual information, open the path aptoide_mature_app_detector/ aptoide_mature_app_detector/explicit_content_detector/ API/Explicit_detector/ Text_categorization.

There, you have a script called *"Text_categorization_full_training.py"*. Open it.

On that file is the algorithm for training the text categorization. By default, it connect to a database on the path aptoide_mature_app_detector/ aptoide_mature_app_detector/explicit_content_detector/API/ Explicit_detector/API to download database called *"app_info_big.db"*. Feel free to change it.

The columns used from the database are title, description, category, age and mature. From there, all the pre-processing and feature choosing is pre-defined. In the end, the final model is saved under the name *"model_info.pickle"*: The word features are also serialized on a file with the name *"word_features.pickle"*.

If you want to do the training all over again, just run from the command-line: *python Text_categorization_full_training.py*

3.2.2 Mature Content Detector training

If you want to train the mature content detector, open the path aptoide_mature_app_detector/ aptoide_mature_app_detector/explicit_content_detector/API/ Explicit_detector.

There, you have a script called *"analyse_app_training.py"*. Open it.

By default, it connect to a database on the path aptoide_mature_app_detector/ aptoide_mature_app_detector/explicit_content_detector/API/ Explicit_detector/ API to download database called *"app_info_big.db"*. Feel free to change it.

The columns used from the database are title, id, description, category, age and mature. From there, all the process involves seeking for all the images for a determined ID in the default folders and use scripts to analyse images and text information and train the classifier. In the end, the final model is saved under the name *"model_apps_info.pickle"*: The feature sets are also serialized on a file with the name *"featuresets.pickle"*.

If you want to do the training all over again, just run from the command-line: *python Text_categorization_full_training.py*

A full training of a model with 10.000 apps can take up to one day.

3.3 Tests

3.3.1 Image Categorization Tests

If you want to run the tests on image categorization, open the path aptoide_mature_app_detector /aptoide_mature_app_detector /explicit_content_detector/API/ Explicit_detector/ Illustration2Vector/illustration2vec_master.

The script *"test.py"* will go through all the images in the path aptoide_mature_app_detector/ aptoide_mature_app_detector/ explicit_content_detector/API/ Explicit_detector/API

to download database/Big_Database/images and, one by one, will check what is the category of the images and the result of the analysis. In the end, it will print the true positives, true negatives, false positives and false negatives for icons and screenshots. It will also print the time processing all the screenshots and icons.

If you want to do test the images again, just run from the command-line: *python test.py*

3.3.2 Text Categorization Tests

If you want to run the tests on text categorization, open the path aptoide_mature_app_detector/ aptoide_mature_app_detector/ explicit_content_detector/API/ Explicit_detector/ Text_categorization.

There, you have one script called *"Text_categorization-tests.py"* and other called *"Text_categorization_kfold.py"*.

Both display different tests. The first one, trains the classifier with all the data, except a defined number for testing. It can be changed in the variable *number_testing*.

By default, it connect to a database on the path aptoide_mature_app_detector/ aptoide_mature_app_detector/ explicit_content_detector /API/ Explicit_detector/API to download database called *"app_info_big.db"*. Feel free to change it.

The columns used from the database are title, id, description, category, age and mature.

The classifiers not used are on the bottom of the code. If you want to see their result, just uncomment them.

In the end, for each classifier, the script will print: Time training, True Positives, False Positives, True Negatives, False Negatives, Explicit Precision, Explicit recall, Explicit F-Score, Non-Explicit Precision, Non-Explicit Recall, Non-Explicit F-Score and Accuracy Percent.

If you desire to run the first test, just type in the command-line *Text_categorization-tests.py*.

The other script, *"Text_categorization_kfold.py"* does k-folding cross validation tests. The parameter "k" can be changed at the beginning of the script. For every test, it will display the results mentioned above on the command-line. The process is the same as the single test above.

3.3.3 Mature Content Detector Tests

If you want to run the mature content detector tests, open the path aptoide_mature_app_detector/ aptoide_mature_app_detector/ explicit_content_detector/API/ Explicit_detector.

There are two scripts for testing. First, the *analyse_app_tests.py* will train the classifier with all the data, except a pre-defined number parameter used for testing. It can be changed on the variable *number_testing*.

By default, it connect to a database on the path aptoide_mature_app_detector/ aptoide_mature_app_detector/ explicit_content_detector/API/ Explicit_detector/API to download database called *"app_info_big.db"*. Feel free to change it.

The columns used from the database are title, id, description, category, age and mature. From there, all the process involves seeking for all the images for a determined ID in the default folders and use scripts to analyse images and text information and train the classifier.

The unused classifier are commented. If you want to test one of them, just uncomment it.

In the end, for each classifier, the script will print: Time training, True Positives, False Positives, True Negatives, False Negatives, Explicit Precision, Explicit recall, Explicit F-Score, Non-Explicit Precision, Non-Explicit Recall, Non-Explicit F-Score and Accuracy Percent.

The other script, *"analyse_app_kfold_tests.py"* does k-folding cross validation tests. The parameter "k" can be changed at the beginning of the script. For every test, it will display the results mentioned above on the command-line. The process is the same as the single test above.

4 Detailed information about the scripts

4.1 Django

4.1.1 Views

The Django project is on the file *explicit_content_detector*. From there, if you open *explicit_content_detector/urls.py*, you can have access to the url used to the api (*detect_mature*). If you open *API*, you can see all the default django files created. We are only going to analyse *urls.py* and *views.py*.

On the *views.py*, we can see the class *Model*, used to save variable across requests. It saves the *Illustration2Vec* model and Boolean value to indicate if the images should be parsed locally or remotely.

The two views used are *getById* and *getByMD5*. Both start by loading the configuration file and access the web services to parse the json content. If they fail, they will return a json message.

Then, they will check if the user wants a synchronous or an assynchronous request. If is set to synchronous the method used will be *get_data_sync*. If not and the cache_reload is not set, first it will check to parse the cached result of the analysis. If the app was already analysed, it will fetch the result. If it was not, will send a json message with the status "Request Submitted" and it will analyse the app simultaneously in another thread with the method *get_data_async*.

The methods *get_data_sync* and *get_data_async* start by loading the configuration model and check if the images are set to be fetched locally or in the web. Then, gets all the information about the app (title, id, description, icons, screenshots, ...).

Finally, analyses the app with the method *analyse_app* and saves the result. On the synchronous method, it will return True/False. On the asynchronous method, it will open a webpage specified in the configuration file and it will add "md5sum=md5_of_app" or "id=id_of_app", depending on which one was used. It will also add to the url the result, as "/mature=True" or "/mature=False".

4.2 Mature Content Detector

4.2.1 analyse_app.py

The analyse app has two main methods. The *get_model* returns the model of *Illustration2Vec* training. The main module is the *analyse_app*.

It begins by opening the config file to load all the configurations and connecting to the database used to save the results. If the app was already analysed, just return the result. If not, we will have to analyse it again. To analyse it, we need to load the serialized classifier for text. Then, we will try to get the analysis result for the images. If we do not have them, we will have to analyse them again. The same applies for the textual information. In the end, we will do the same to get the final result and return it.

If the option cache_reload is set, everything will be analysed again and inserted or updated to the database.

4.2.2 analyse_app_training.py

This script starts by loading the model for the *Illustration2Vec* module. Then, it connects to a database and fetches all the content divided in explicit/non-explicit. For all the apps, it will create a list with its the screenshots and icons and, for every one of them, it will find the feature sets. For convenience, the script first tries to read from file the feature sets. If it did not find any serialized object, he will start creating the feature sets. This process can take up to 8 hours with 4000 apps.

After having all the features, it will train the classifier with all of them at it will serialize the model to be used by other scripts.

4.2.3 analyse_app_tests.py

The process of getting the feature sets is the same as the script *analyse_app_training.py*. After having all the features, they will be divided in two sets: training and testing. The number of sets for testing can be defined in a global variable. After having the sets divided, it will train a set of classifiers and, with the testing set, print a number of metrics for the classifiers. The metrics printed will be: Time training, True Positives, False positives, True Negatives, False Negatives, Explicit Precision, Explicit Recall, Explicit F-Score, Non-Explicit Precision, Non-Explicit Recall, Non-Explicit F-Score and Final Accuracy Percent.

4.2.4 analyse_app_kfold_tests.py

This script is identical to *analyse_app_tests.py*. Instead of dividing the feature sets in testing set and training set, all the features will be used in the training and testing. For a k defined as a global variable, the script will analyse k times the classifiers, every time with a different testing set.

4.3 API to download database

4.3.1 get_list_id.py

This script receives as command-line argument the query to search on the Aptoide Web-Services. Optionally, it can also receive the maximum number of apps to be fetched.

For example: *python get_list_id.py games 20* will retrieve the first 20 apps with the query "games".

The scripts starts by connecting to a database "app_info.db". Then, it will search for the query given and it will save the id's on a table called *"crawl_list"*, as well as the current date, the query and '0' in "is_processed" column. While there are more results and the maximum limit of apps still isn't over yet, it will continue saving the content.

If you want to delete everything from the table, just add the option "-r" to the script as the only argument.

For example, *python get_list_id -r*.

4.3.2 get_store_info.py

This script will download the information of the apps whose id has not been processed yet. Optionally, you can pass as command-line argument the maximum number of id's you want to fetch information.

For example, *python get_store_info 20* will only download the content of 20 apps.

It begins by connecting to a database and selecting all the app id's that were not processed. Then, id loads all the textual information from the Web Services saves them on a database. Also downloads the images and screenshots and divide them as explicit/non-explicit by his minimum age.

If your desire is to delete all the textual content of the processed apps, just run the script with the only argument "-r".

For example, *python get_store_info -r*.

4.3.3 Misclassified Information

There are three scripts (*misclassified_image, misclassified_text, misclassified_apps*) that measure on a database the error classification with the minimum_age and explicitness. They print the following metrics: True Positives, False Positives, True Negatives, False Negatives, Precision, Recall, F-Score, Accuracy Percent.

4.4 Image Categorization

4.4.1 analyse_image.py

The method *analyse_explicit* analyses the explicitness of an image and returns a tuple ("explicit": percentage of explicitness, "safe": percentage of safeness).

If you want to test it, you can run it from the command-line with the images path as arguments. It begins by loading the neural network model for the images and, for every image, it will analyse its explicitness with the image model.

4.4.2 test.py

The testing for analysing the images starts by opening the model with the trained classifier for the image categorization. Then, based on the image directory, it will analyse them all and print the true/false positives/negatives, divided by icons and screenshots.

4.5 Text Categorization

4.5.1 Text_categorization.py

The text categorization has as main method *text_cat*, which receives as a parameter the description, the size of description, the category and the minimum age of an app. It begins by loading the model for text categorization and the word features.

Then, it will find the feature in the app content retrieved and it will return the classifier probability.

4.5.2 Text_categorization_full_training.py

This script saves the classifier with the full training made. It has a global variable, the number of commons words used, set to 10000 by default based on experimental tests.

It retrieves the information from a database and divides it in explicit/non-explicit. Then, all the descriptions are tokenized by words, the stop words and the punctuation is removed, as well as the uppercase letters, and the words are stemmed. For the final part of the pre-processing, it is created a frequency distribution with the most common words.

It is created the feature set for all the apps, using the description and the category (features chosed based on experimental tests).

At the end, the model is trained with all the features and serialized on a file, as well as the features.

4.5.3 Text_categorization-tests.py

All the pre-processing and the feature sets are find using the same techniques as on the script *Text_categorization.py*.

The number of testing set length is defined in the beginning as a global variable. What's left of the feature sets will be the training set. Then, for the classifiers chosen, the script will print the time training, true/false positives/negative, Explicit/Non-Explicit precision, recall and F-Score and the final accuracy percent.

4.5.4 Text_categorization_kfold.py

This script is similar to *Text_categorization-tests.py*, but instead of dividing the feature sets in training set and testing set, he technique used is different.

For a global variable k, the script will test the classifiers k times, printing all the metrics measured. The testing and training set will vary between the tests.