APPIC: Finding The Hidden Scene Behind Description Files for Android Apps

Yingyuan Yang, Jinyuan Stella Sun, Michael W. Berry
Dept. of Electrical Engineering and Computer Science
University of Tennessee
Knoxville, TN, 37996 USA
Email: yyang57@utk.edu
{jysun, berry}@eecs.utk.edu

***Abstract*—Today one of the greatest abundance of applications (apps) are those found on mobile devices. The procedure to download an app has never been easier. The user will first find the desired app within an app market such as the Google Play. Once the user finds the desired app normally a description is given. This description will give the user information on what they are trying to purchase. The user will then download this new piece of software based off of the trust that the description given is correct. This brings to light the new threats that there are chances the descriptions could be misleading. This paper will present the idea of APPIC to solve such issue. APPIC is a framework that can automatically extract main theme (tags) from both the description pages and permission file. By further checking the similarity of these tags, we can verify the validity of the description. In the APPIC the core component is topic modeling. Using different topic models, the APPIC can verify different contents within an app market such as comments and screenshots. For this paper, we will discuss the two main methods used in the APPIC which are latent dirichlet allocation (LDA) and partially labeled dirichlet allocation (PLDA). During the testing phase the APPIC will use different parameters, topics, models and categories. We have achieved on average 88.1% precision of the app description classification, and on average a 76.5% accuracy of the app permission classification. Therefore, the APPIC can largely reduce the number of misleading descriptions within the app markets.**

I. Introduction

More and more researches tend to focus on analyzing the description file [1] because the permission label can only provide very limited information to the user. Unfortunately, the method of verifying the validity of the description is unseen.

In both Android and iOS, a developer is responsible for selecting the category[1]for his application and writing a description for it. The user then searches for the desirable application based on the category and name. In both Google Play and the Apple Store, there are only policies about how to write a morally correct description, which means a malicious developer can still publish an app with a misleading description. Thus after the app is published there is not a program to check the validity of the description. Also, there is not a verification mechanism to check the category either. Only the user will be able to verify whether the application downloaded is correct. In this paper, we will assume two properties for a valid description: First, a well written description must be in the correct category. Secondly, a well written description has to be the same as the hidden features of the permission labels.

Currently the mobile operating systems which include the Android and iOS have user involved malware detection mechanisms besides the traditional inspection done by either employee or system [2]. In this way, a permission label provides some information for the user to judge the quality of an unknown application. Most of the time the permission labels can only provide a limited of information [3]–[5] for the user. Thus, the user may not understand the permission message [6]. In such a case, the user may need to further consider more informative materials such as an application description [1]. An example would be reading other user's comments to decide whether an application is malware or generally not needed. Moreover, compared to the very limited information provided by permission labels, the description is more understandable and may be more reliable.

Most (application) descriptions can provide better information than a permission based mechanism with limited size labels [7]–[9]. The problem is now the descriptions can also mislead or hide some of the critical pieces of information from the user. A good description provides accurate information about the application. A poor description, however, may mislead the user or describe something not related to the application. If we can provide a reliable and well written description, this will improve the searching efficiency.

The users have played an important role in an application market in both the feedback and security especially in the Google Play Market. In the Android operating system, a permission-based security model only has to be responsible to the user for the evaluation of the application. They will need to check to see if there exists any unusual permissions for that app [10]. Later the user may refer to the descriptions of such app for more detailed information. After used such app, the user can post feedback or report to suggest Google Play keeping or banning this app. In the App Store, the user will only need to read the accompanying descriptions of the app to decide which one he needs. The most tedious work of

[1]The term "Category" is the same as the original class in Google Play, such as "Book", "Game", etc. The term "Auto generated tags" is the same as the first two most related "topics" in PLDA. It is also the human calibrated tags in LDA. In short, we will call the "Auto generated tags" as "Auto Tags" or ATs.

verification already has been done by an Apple employee.

From the developer's point of view, once they have completed their app they may need to first upload this app to either Google Play or the App Store in order to provide downloadable content for other users. During the upload period, they have been asked to provide a description for their app and select a category for it. These description and category tags will be read by other users in the future. The description may or may not reflect the real value of the app. The developer can also mislead the user by selecting the wrong category or writing a misleading review. In this way, both the Google permission based system and Apple iOS provides no automatic protection. The developer can then write anything they want in the description and even something completely different from the actually purpose of the app. In this paper, the potential dangers are noted for the previously stated scenario. For example: Malicious developer "Alice" may generate some malware and write a description for an app claiming that it is affiliated with certain banks. It then prompts the user to enter in personal information and steals it for malicious purposes. User "Bob", trusting the security within Google Play or the Apple Store, will then read the description which does not state the truth and further download the app.

If the description can be checked to see whether or not it describes the real purpose of the app as it is uploaded, we can reduce the chance of a mal-write description. This paper will demonstrate the tested solution for this issue. Similarly, it is also possible for the developer to select a wrong category either intentionally or by mistake when they are uploading the app. We can check this as well using our system.

This paper makes the following main contributions:

- To demonstrate and analyze the potential security issue within the description which can be misleading for the user when they are selecting apps and to demonstrate the issues with verifying the descriptions on the app markets.
- Will provide a solution for solving this issue using topic models. We are using both LDA and PLDA for detecting hidden topics in each application for both the description and permissions file while generating an auto-tag for them as well. Through testing we have discovered that PLDA is much more suitable for this problem.
- Evaluated the APPIC system on 207865 apps from Google Play. Also have achieved 88.1% precision within the app description classification, and 76.5% accuracy in app permission classifications.
- Filling in the gaps between category and security in the app market. Pointing out that a well defined category system can provide the user with a correct understanding of the app, and at the same time improve the reliability of the app market.

Using topic models we can extract the main topics for each description and permissions which are hidden from us. The name APPIC is from using both the words "application" and "topic". The APPIC is a framework and system that can grab the description for a listed app name within a database and can use a topic model to find the topics. Once we have some topics from the description and permission, we can further generate ATs and compare these two ATs to see if they are same. If not, we may claim that this description is misleading. Furthermore, we can also check the category in Google Play[2]to see if the developer has selected the correct category.

The paper is organized as follows. In Section 2, we discuss the general structure of APPIC and how does it verify the app. We further describe the main components of APPIC in Section 3. In Section 4, we present the result of using LDA and PLDA to extract ATs. We will also compare these two methods to check their suitability for our problem. Section 5 discusses related works. Finally, Section 6 concludes our work.

II. APPIC Overview

APPIC is a framework that can extract tags[3]from descriptions and permissions. It can then compare both of these tags to see if they are matching and will further decide whether the description is correctly describing the apps. The methodology for the APPIC can be shown in Fig. 1. This chart begins with the assumption that the user wants to download an app. In order to find the correct app, the user will try to search the app market to locate apps which are suitable for his requirements. Once the apps are located, he can then read the description for each app. Unfortunately, he does not know whether the description is accurate. Next with the APPIC server he can now check the description for correctness. The APPIC will then tell him the validity of the description. The information provided by APPIC can either be an embedded tag (correct sign) attached to a description, or a message from the APPIC server.

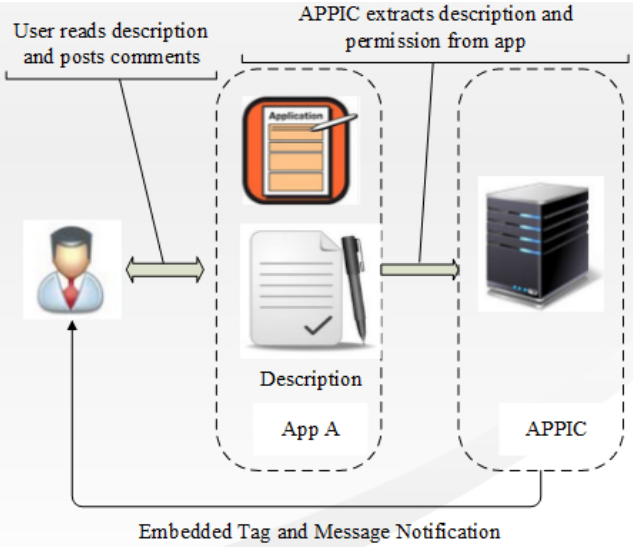

Fig. 1. APPIC Overview: User finds app "A" which is suitable for his requirement by reading the description but he is not sure about the correctness of description for such app. APPIC will check it for him and forward the result to user or embed the result beside the description

[2]We have not tested the app within the Apple Store but the method is still the same

[3]Generally, the topics generated by LDA is not same as Tags on Google play, but these topics can be converted to similar tags (ATs). We will further talk about this process in the next section.

The total time for the evaluation process can be very short - testing and comparing a specific app description takes less than 1 second even if the description contains 1000 words. Furthermore, the whole process can be precalculated before the user reads the description and can embed the result tag beside the description. In such conditions, the user can read the tag before the description and save themselves valuable time. (If they see a misleading sign provided by the APPIC, they can simply pass the app). APPIC can largely reduce the chance of mal-written descriptions which can mislead the user. It may also purify the app market environment to encourage developers to write an understandable and truthful description of their apps.

III. APPIC

From a flexibility standpoint - the APPIC is using the topic models as embedded tools for generating auto-tags but it is not limited to any specific topic model. By selecting a different topic model the APPIC can achieve different goals. For instance, it can do a screenshot check or user comments attitude check and so on.

The APPIC contains several components: raw data crawling, data filtering, training, auto-tag generation, app description and permission testing and comparison of the auto-tags. These processes can use different programming languages and platforms. For the demonstration, we are using a .NET platform, C# , and a Stanford Topic Model [11] package, though the framework is not limited to just these tools.

We will discuss training and testing in the following section.

A. Data Filtering Before Training

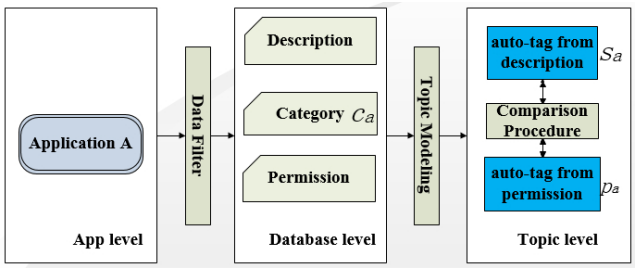

Fig. 2. APPIC flow chart

From Fig. 2 we can see that the raw data has been filtered before it enters into the database. This process is necessary because some of the descriptions are a non-English document or too short (less than 10 characters) to be considered valid. Some documents have poorly written sentences or an inaccurate translation[4], which we can not detect, leading to inaccuracies in future training process. The above processes have been done using C# .NET and MS-SQL stored procedures.

After we have purified the data, we store it in our MS-SQL database and further divide it into different categories based on years and category tags. The information is coming directly from Google Play. From the database, we can send training data to the topic model. The topic model can either be embedded in the database server or in other servers (More servers mean more security but less efficiency. In here, we are using only one server that has both database and topic models). After training, we can use the model for testing. The training process may take 2.5 minutes (34 topics on 2.4-GHz quad-core, 6GB machine) to 30 minutes (84 topics) depending on how large the training set is and how many topics were needed. Because of the special property of LDA - refer section III-C, to calculate each per-description topic distributions and per-topic word distributions will take an exponential time when the number of topics increases. Furthermore, the training uses the Gibbs-sampling approaches [12]. In order to converge, the total iteration should be large enough. The testing phase, however, is faster than training phase (less than 1 seconds).

Training and testing happen in both the description and permission. The results have been compared in the server using the stored procedure. After testing the result can either be attached to description or directly sent to the user.

B. Topic Model

The topic model is a probabilistic modeling tool that can extract various topics from different corpus (description). The topic model generally uses what is called the machine learning method - a training method using large set corpus and then predicting hidden topics from testing dataset. After training, given an unknown app description the model can tell to which category the app belongs. Another feature is that given a set of permission labels of a specific app, the topic model can tell to which class the app belongs.

Suppose we have an app which we have named "A" (see Fig.2). This app first is taken through the testing procedures with the data filtering mentioned earlier. After this testing phase, we have a topic category p_a by analyzing a set of permission labels. Next we can analyze the description file, and gain the topic s_a . We can also have topic c_a by reading the developers chosen category tag. By comparing p_a, s_a, c_a we can easily find out if this description is correctly describing the application.

C. LDA and PLDA

Since each description has a category, we can choose partial labeled latent Dirichlet allocation (PLDA) [13] to find the hidden category of a specific application. PLDA differs from LDA because it is partially supervised which means there are human added tags ("tags" are related categories for each application) for each description. Since each description may exhibit several different topics, we have only chosen the top two largest topic rates in which to verify our model. We will also compare the precision rate of traditional LDA and PLDA in the evaluation section.

The LDA model is one of the chosen topic models [14]. It is a multidimensional distribution model [15] with the total

[4]Google Play provides translation service to developers who are from other countries.

joint distribution as,

$$p(\varphi_{1:K}, \theta_{1:D}, z_{1:D}, w_{1:D}; \alpha, \beta) = \prod_{i=1}^{K} p(\varphi_i; \beta) \prod_{d=1}^{D} p(\theta_d; \alpha) \prod_{n=1}^{N} p(z_{d,n}|\theta_d) p(w_{d,n}|\varphi_{1:K}, z_{d,n}) \quad (1)$$

The φ_k indicates a distribution over each word in kth topic. θ_d indicates the topic proportions for the dth app description. $\theta_{d,k}$ is the topic proportion for topic k in description file d. $z_{d,n}$ indicates the topic assignment for the nth word in the description file d. $w_{d,n}$ is the nth word in description d, which is an element from the fixed vocabulary. α is a concentration parameter which controls the sparsity of the topic that is assigned to each description. β is a smoothing vector similar to α which controls the sparsity of per-description word distribution.

It is easier to understand the model from the training and testing aspect. The training process is used to find a list with the most possible words for several similar descriptions. It should list these words based on their probability. While, on the other hand, the testing process is assigning topics to a specific description based on probability. One description can be assigned several topics, but only the first two topics are important to us.

PLDA [13] uses a partially supervised (tagged) procedure with humans to generate topics. It is very suitable for our problems because each app must have a category - this is required by many of the app markets including Google Play and the Apple store. These categories can be used as human annotated tags for training purposes which have made the testing phase more accurate.

D. Using the Topic Model

We are using the Stanford topic model package to train and infer description and permission. This package is open source, and we first calculate the count of each word in each and every document. Then we add some meaningless words into the stop list because they are too common to be useful for representation of any topic. Some examples are "the", "and", "you", "your", "for" and so forth. There are about 40 kinds of words in the description and 60 in the permission.

Next we need to select a reasonable α parameter in function 1. α is the parameter of the Dirichlet shown prior on the per document topic distributions. This parameter controls the topic sparsity in each document. For this paper, we need no more than 2 topics to be assigned to each document. After testing, we find the $\alpha = 0.05$ fits our requirements the best. After this, we set the training iteration to 4000 with the average change in the expected log likelihood to be less than 0.001%.

By end of the testing phase, each document has been assigned two topics. Topics will contain the most co-occurring words in similar documents and are arranged based on the probability order. An example of the topic would be, "Topic2: music, player, play, guitar, audio, video, album, media, song, mp3, sound". This is a string of words arranged based on probability (with the largest first). In the former example, we can easily see that it is about music and media which will correspond to "media" category in Google Play.

E. Generating Auto-Tags

The topic generated by our model is a distribution of words in the description in which all of the words in each topic are derived from a word in the description with the order of highest probability first. An example of these would be: "Topic1: gps, speed, distance, google, while, track, direction, map, ...".

In LDA, the topic words may or may not match the original category in Google Play. Two or more topics may belong to one category. For example, the topic "gps, speed, distance, google, while, track, direction, map" and topic "location, map current, maps, places, address" are all words that belong to the "Travel and Local" category. One of the topics may also belong to two or more categories, but after we select more than 64 topics this case is seldom seen. We will need to rematch these topics to each category. This process only happens once, and we call it human calibration. After the calibration, the program can now easily match each topic to a category. In PLDA, because most of the auto-tag areas are generated from the original category tags, we do not need to rematch the topics to tags with the category. We can instead select the top 2 words in each topic and use them directly as an auto-tag. For example, the words "tools", "productivity", and "communication" can be placed in one topic and we can select the words "tools" and "productivity" as our auto-tags. This category will have the same words.

F. Comparison

The last step is comparing the permission auto-tag with the description auto-tag and the categories. Once we have the auto-tags in the permission and description this step becomes easier to manage. Next we will once again use the stored procedure within a database to compare each of the auto-tags. The final testing result can be published to a website using Javascript, or a message can be sent to the user who can trigger the APPIC service.

IV. Evaluation

We will be evaluating the APPIC in this section. After the training phase, the APPIC can now classify a new app and assign it a category tag based on both description and permission categories. By comparing these two tags we can deduce whether the description is correct for the app. The evaluation will focus on two main parts: correctness and perplexity. We can also compare PLDA with LDA to see which program can give us the best solution to our problems. The following experiments try to solve the problems:

- How many topics/tags do we need to classify apps? (i.e., more topics/tags means more accuracy but lower efficiency. We need to find the best number of topics that can achieve the highest efficiency and accuracy).

• What is the accuracy of the predicting category given a specific description or a set of permissions?
• What is the difference between LDA and PLDA in an app classification?

A. Dataset

The dataset we will be using is from Google Play 2011 to 2012 provided by [16]. The dataset has also been cleaned, since there could have been multiple applications developed by same author [8]. The 2011 dataset contains 71,331 apps and the 2012 dataset contains 136,534 apps. We are also crawling all of the descriptions from each individual app and will redo the cleansing again. The recleansing will keep the English descriptions with more than 10 characters at the newest version (in Nov, 2013). Finally, there are only 26703 apps left with the number of words totaling 1100715. Since the LDA does not suffer from the same overfitting problem that the supervised models have [14], we use perplexity(2) to find the best number of parameters. We then randomly slice the dataset into two parts. The first half has 13339 apps will be used for training[5], and 13364 apps will be used for testing - no overfitting as mentioned above. All of the training and testing datasets will have two pairs of a description and permission.

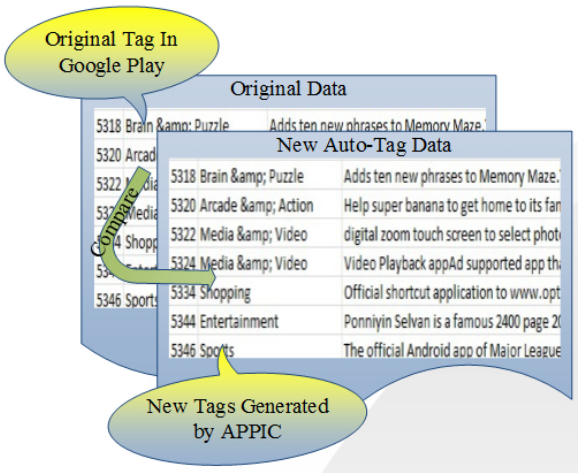

Fig. 3. Evaluation Process

The Figure 3 illustrates the process of inspection. The new Auto-Tag dataset will be data generated by the APPIC. On the other hand, the original dataset is data that has been filtered from Google Play. In both the original data table and the new auto-tag data table, the first columns are the same which can be used to uniquely identify an app. The second columns are generated from the APPIC which have been used in comparing the original data table to verify the correctness. For completeness, we also have left both of the raw description and permission tables untouched. Generally, we only need to check the second columns to see whether the auto-tags are correct match or not.

[5]We will assume all of the data from Google Play is accurate, but in reality they are not. During the training phase, we find that there are less than 1% of the data that either have the wrong category or have the wrong description.

B. Evaluation Setup

During the testing period we randomly choose 1000 apps to test. The results from this testing phase have been inspected by three different individuals. Each of them independently evaluated each individual tag to check for correctness.(i.e., given a description "A cool app to track your class grades and instantly calculate your GPA through the semester." The model will generate a tag to classify it. Based on Google Play, the correct tag is "Education". If the auto-tag is not "Education", we mark it as incorrect; if it is "Education" we can mark it as correct.) It is also possible that the generated topics have a very close probability which means APPIC will generate two or more different auto-tags for such app. For this special case, we will mark the app as "ambiguous". It is also possible that the three individuals all reach a different conclusion. If so, we will also mark the app as "ambiguous".

Another problem we have faced is how many different topics/auto-tags do we need to properly classify different apps. The method that Google Play uses is to separate the apps into 34 different categories. Within these 34 categories, they further divide them into 98 smaller categories. The problem with this method is we do not know how many categories specifically we will really need to classify the different apps. We use perplexity [17] to approximate calculate the number of topics. The perplexity is monotonically decreasing in the likelihood of the test data. It measures how well a probability model predicting the test data. The lower the perplexity score, the better the generalization performance [14]. Generally, it will be stable at some number of topics. However, it is just an approximation - may or may not reflect the best number of topics. For a dataset that contains M apps descriptions, the perplexity is:

$$perplexity(D_{test}) = exp\{-\frac{\sum_{d=1}^{M} logp(w_d)}{\sum_{d=1}^{M} N_d}\} \qquad (2)$$

$w_d$is a specific word in a document d. N_d is the total number of words in a document d. The perplexity is algebraically equivalent to the inverse of the geometric mean per-word likelihood.

C. Result

In this section we will demonstrate the effectiveness of the APPIC in predicting the app tag based on the description and permission.

1) Topic/Auto-Tag number selection: The number of topics directly affect the efficiency of our framework. For instance, too many topics are unnecessary and inefficient while too few topics may not be enough to classify the data.

Because perplexity can only be viewed as a simple but necessary guess for the trend of a topic number, it may not be an accurate enough guess for our problem. We may instead need to do a more accurate human inspection which will be discussed in the next subsection. From Figure 4, we can see that the two lowest points happened between 50 to 100 and

150 to 200 [6]from the description. On the other hand, in the permission, the lowest points are between 1 to 50.

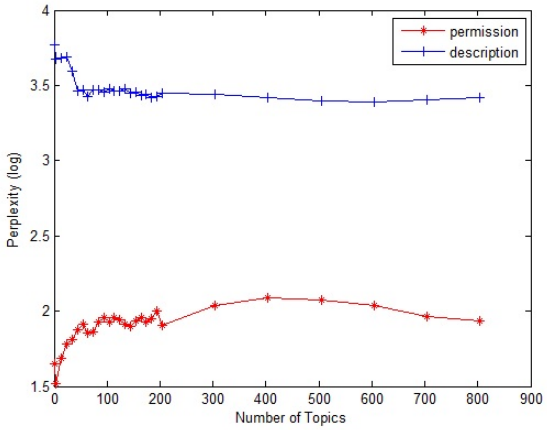

Fig. 4. Perplexity in different number of topics (the lower the better). y-axis is the logarithm of perplexity to base 10. x-axis is the number of topics.

2) Why PLDA: We have discovered that the best topic number will fall into three different intervals from the last subsection. In these intervals we may need to find a specific number for the topics. At the same time, we also will take this chance to further compare PLDA and LDA based on the varying numbers of the topic. A better model will outperform the other in most of these cases.

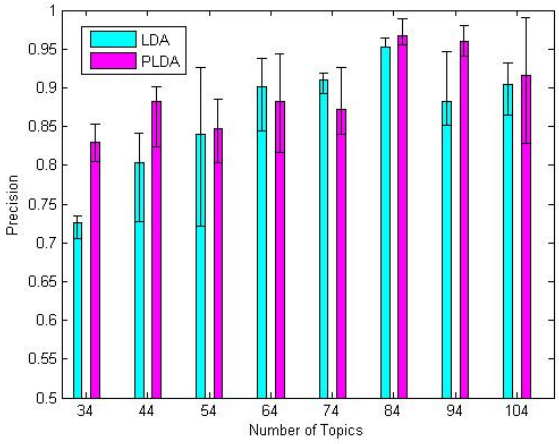

Fig. 5. Comparison between LDA and PLDA in different number of topics

Using Figure 5 we have found that after using human inspection of 34 to 104 topics, PLDA is better than LDA within our model but in some cases such as 64 and 74 topics LDA is better. This is because of a special property of PLDA [7]. Both PLDA and LDA will reach there peek when a topic number is 84. Now we can check the correctness of the auto-tags as well as further check the suitability of LDA and PLDA in the final tag generation.

3) Description Classification Using Both PLDA and LDA: In this experiment, we will check the correctness of an auto-tag for each individual app based on a given description. We will also take this chance to further compare the suitability of LDA and PLDA as a solution to our problem.

The process shown in Figure 3 will be demonstrated as we compare the auto-tag with the original tag to see if they are a match. Please note that in Figure 5, we are verifying the correctness of a topic distribution table such as ”music, player, play, guitar, audio, video, album, media, song, mp3, and sound”. Thus we can verify the correctness of the topic that is assigned to each document. From table I we can see PLDA is more suitable than LDA from the correctness and ambiguity aspect. In this testing phase, the total precision using PLDA achieved 88.1% precision in the category reference.

TABLE I
PRECISION USING PLDA AND LDA IN CATEGORY REFERENCE

Model	Correct Inference	Incorrect Inference	Ambiguous Inference
PLDA	88.1%	11.1%	0.08%
LDA	78.4%	16.6%	0.5%

*The precision rate of PLDA and LDA (randomly choose 1000 apps from testing dataset).

4) Permission Classification Using PLDA: From the result of the last subsection, we have found that PLDA has better performance than LDA. In this section, we will complete the testing phase for the permissions to see how well the PLDA performs on different categories. We randomly selected 1500 apps from the testing results found in the earlier subsection. Next, we separate all 1500 apps into different categories (priori) based on their original categories in the Google Play. Once again the apps will be inspected by three different people. We now will define the following measurements: true positive (TP) is the correct tag generated by the APPIC for a specific category of an app based on permissions; false positive (FP) is the incorrect tag generated by the APPIC of an app category; true negative (TN) will be an app that the APPIC correctly identifies but it does not belong to a defined category; false negative (FN) will be an app that the APPIC incorrectly identify and it does not belong to a defined category. For example, the defined category is ”tools” and given the app the APPIC correctly tags it as ”tools”. So within the APPIC server we count TP +1 because if the defined category is not ”tools” this shows the APPIC has incorrectly tagged it as ”tools”. Thus the APPIC server will add FP +1. Finally if the given app is not ”tools” and the APPIC has tagged it as ”something else” then TN +1. Otherwise if the given app is tagged as ”tools”,

[6]We did not consider 150 to 200 topics because if we compare these numbers to Google Play’s classification numbers (34), the 150 topics are too high to be valid.

[7]In PLDA, the words in each topic are generated based on a category tag as well as the words in the description. Some of the tags only contain general concepts(i.e. Lifestyle or Shopping; Game or Entertainment, etc.). It is hard to decide the correctness of a specific topic, especially when the topic contains too many of such words. We will mark all of these topics as ambiguous in PLDA, though they are essentially correct. In topic 64 and 74, APPIC happens to select much of these words.

but the APPIC incorrectly tags it as "something else", we will add FN +1. Once all of the tags have been collected we can calculate the accuracy of each category using the following equations:

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} \quad (3)$$

From Figure 6, we now know that within the category "tools" we have an average accuracy of 97.3% from a total of 371 apps. The tag "game" has a 80.3% from a total of 224 apps. The tag "lifestyle" has a 89.5% from a total of 173 apps. The tag "video" has a 63.8% from a total of 105 apps. The tag "travel" has a 88.2% from a total of 204 apps. The tag "education" has a 20.5% from a total of 107 apps. The tag "finance" has a 10.6% from a total of 94 apps, and the other categories have a 77.4% total from 222 apps. Overall, the average accuracy is 76.5% for permission classification. The lowest point exists in "education" and "finance". The reason is that in both of these two categories the app tends to requests different permissions. There are very few co-occurring special permissions that can precisely identify these two types of apps.

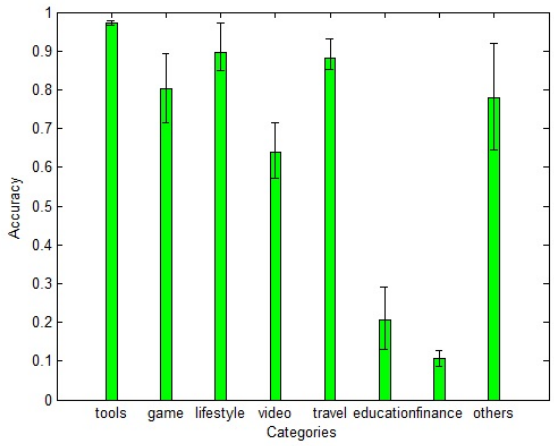

Fig. 6. Accuracy of different categories in permission file. We assume that one document (permission file) can match only one category. The sizes of each category are listed as follows: tools 371, game 224, lifestyle 173, video 105, travel 204, education 107, finance 94, others 222.

D. Summary

We have fully tested our framework using data filtering, auto-tag generating, and a category comparison. We also have compared different topic models like LDA and PLDA and found that PLDA is a better solution to our problems. In addition, we have calculated the perplexity in a various range of topics. We have now found that the best number of topics is 84 which has an accuracy of 88.1% using PLDA. Next we compared LDA and PLDA in suitability. The results for (PLDA) look promising for most of the topics. We then checked the performance of PLDA using various categories. Finally, we have listed the top 7 categories. From the results, we can see that in most categories PLDA has a 76.5% accuracy rate on average for permission classification. We also have a 88.1% accuracy rate on average for description classification.

V. Discussions and Future Work

With the current setup, we have only built a n-tiers [18] framework that can automatically extract a desired app description from Google Play using C#.NET and Javascript. In the database tier, we have several stored procedures that can further trim and format the raw data from the cowling results. By using the topic modeling tool box provided by Stanford University [19], we can achieve PLDA and LDA training and testing. They are java packages that are open source which are very easy to embed and implement. We have already written several stored procedures to further annotate the result. After the human calibration of our result set, we can further complete the program to inspect an unknown app without any human supervision in the future.

We have not covered all of the possible range of topics in this paper. The framework we have so far is only to consider that the topic numbers are less than 104. It is possible that a topic number in between 150 to 200 may generate a better result, which can be considered for further investigation.

Another issue that we have acknowledged is the low inference accuracy in some categories, especially in the education and finance tags. This low accuracy is due to a lack of sufficient symbolic permissions for these categories. It has proven to be very hard to separate them. This issue could be solved by using better topic model or using a more reasonable classification method. For example, a education app may physically seem similar to a game app from the permissions view. In such case, we should consider it as a category between a game and education. Unfortunately, Google Play does not provide such a category. We do not know how many of these cases exist within the market, or how well the prediction could perform if we generated an additional and more appropriate catagory. We believe that it should improve the overall accuracy.

The training dataset we are using may not be accurate for some parts of the app since we have made the assumption that all the data from Google Play is accurate. In actuality there is about 1% data that is incorrect in Google Play. However, to further trim and correct the inaccurate data may take long periods of time. We believe the data in the App Store could be better, but we have not used their data. For future work we could consider adding this dataset to the APPIC Server.

Indeed, the APPIC is not only for Android but is also available for various products including Apple iOS and Windows. We can verify their description and category if given an app with permission file. We can further verify whether the permission requirement matches the description tags.

Finally, the APPIC can not only be used as detection system for mal-written descriptions, but it can also be used to evaluate an app from user comments or even evaluate the images the developer has uploaded. This is possible because APPIC can be used as an extraction tool for common words based off of user comment and then decide if the comments are positive

or negative. The App Store and Google Play all require a developer to upload at least 2 to 6 screenshots of their app. These screenshots can also be misleading. Using a different topic model, the APPIC can process the image [20] and check the category of said image and compare with the permission to see the validity of the image.

VI. Related Work

Our framework has crossed several areas: topic modeling, information security, text mining, machine leaning, and software evaluation. In this section, we will discuss the relevant research that is pertinent to our work.

In 2012, Harman et al. [21] first applied the data mining approach to the App Store. They used data mining to extract certain feature information from the app and combined it with some business information. Later Pandita et al. [1] has discussed the important relationships between description and permission when the user is trying to understand permission tags. They are using the traditional First-Order-Logic (FOL) to analyze the sentences in the description. The flaw of such approach is: it is hard to analyze the sentence with complex grammar. The APPIC uses a topic model that is grammar free and more accurate especially when dealing with unsupervised descriptions with no human selected tags. The APPIC can also achieve very high accuracy (please refer evaluation section).

The topic model has first been described by Papadimitriou et al. [22]. They have proven that Latent Semantic Indexing (LSI) can capture the underlying semantics of the corpus under certain conditions. Later Hofmann and Thomas [23] improved LSI in probabilistic way. In 2002, David Blei et al. [14] developed the Latent Dirichlet Allocation (LDA) and proved that it is a better tool than PLSI from a perplexity view because LDA can be easily embedded into different models. Lots of types of topic modeling tools have been developed in last decade. One of them is the Partially Labeled Dirichlet Allocation (PLDA) [13] developed by Ramage et al. The APPIC uses these two traditional topic models (PLDA and LDA) for description and permission classification. A plethora of ideas have been developed between NLP and the software industry [24]–[26]. Due to the flexibility and high precision rate, some of the cross area NLP models have achieved very successful implementations such as Hidden Markov Model using face recognition [27] and network regularization [28]. The topic model also be used as a classification tool and has achieved impressive results [29]. Unfortunately, very few of these tools have been able to be properly used with topic modeling in the app market. Our work has properly used this tool to demonstrate the effectiveness of using the topic model.

With respect to mobile software verification there are several different types of work done being done on permissions analyzing [8], [30]–[32]. Among them, Felt et al. [30] has stated the relationship between malware detection and permission. Zhou et al. started using a probabilistic method to analyze permission classifications. In addition, Enck et al. [33] suggested using the dynamic analysis technique to detect the potential misusage within private information.

The works mentioned above are mainly focused on either NLP for specific problems or a permission security issue on an Android system. A true analysis of the validity of a description based off of a permission is so far unseen. The APPIC uses both the topic modeling and the android permissions to find the scene behind the app. It has also filled the gap for description verification.

VII. Conclusion

We have described the APPIC framework. From the test results, we can see that the APPIC can be used as a backend system that verifies the validity of a description with a very high average accuracy rate (76.5% accuracy rate for permission classification,and an 88.1% average precision rate for the description classification). The APPIC can become a very useful tool for the user who is reading an app description. It can largely reduce the number of misleading descriptions, and it is also an important framework to purify the environment of the app market. This market is not just limited to Google Play, but can also include the App Store and the Windows Store. After the APPIC tagged the user can truly trust the validity of a description for different apps.

VIII. Acknowledgements

We gratefully acknowledge the thoughtful review and invaluable suggestions from Wong, Nina Mei-Yee. We also thank Xueli Huang for her help with data inspection.

References

[1] R. Pandita, X. Xiao, W. Yang, W. Enck, and T. Xie, "Whyper: towards automating risk assessment of mobile applications," in *Proceedings of the 22nd USENIX Security Symposium, Washington DC, USA*, 2013.

[2] H. Lockheimer, "Android and security," 2012. [Online]. Available: http://googlemobile.blogspot.com/2012/02/android-and-security.html

[3] A. P. Felt, E. Ha, S. Egelman, A. Haney, E. Chin, and D. Wagner, "Android permissions: User attention, comprehension, and behavior," in *Proceedings of the Eighth Symposium on Usable Privacy and Security*, 2012.

[4] W. Enck, M. Ongtang, and P. McDaniel, "On lightweight mobile phone application certification," in *Proceedings of the 16th ACM conference on Computer and communications security*, 2009.

[5] D. Barrera, H. G. Kayacik, P. C. van Oorschot, and A. Somayaji, "A methodology for empirical analysis of permission-based security models and its application to android," in *Proceedings of the 17th ACM conference on Computer and communications security*, 2010.

[6] A. P. Felt, K. Greenwood, and D. Wagner, "The effectiveness of application permissions," in *Proceedings of the 2nd USENIX conference on Web application development*, 2011.

[7] A. P. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner, "Android permissions demystified," in *Proceedings of the 18th ACM conference on Computer and communications security*, 2011.

[8] H. Peng, C. Gates, B. Sarma, N. Li, Y. Qi, R. Potharaju, C. Nita-Rotaru, and I. Molloy, "Using probabilistic generative models for ranking risks of android apps," in *Proceedings of the 2012 ACM conference on Computer and communications security*, 2012.

[9] B. P. Sarma, N. Li, C. Gates, R. Potharaju, C. Nita-Rotaru, and I. Molloy, "Android permissions: a perspective combining risks and benefits," in *Proceedings of the 17th ACM symposium on Access Control Models and Technologies*, 2012.

[10] Barrera, David and Kayacik, H Güneş and van Oorschot, Paul C and Somayaji, Anil, "A methodology for empirical analysis of permission-based security models and its application to android," in *Proceedings of the 17th ACM conference on Computer and communications security*, 2010.

[11] H. Lockheimer, "Stanford topic modeling toolbox," 2013. [Online]. Available: http://nlp.stanford.edu/software/tmt/tmt-0.4/
[12] C.-J. Kim and C. R. Nelson, "State-space models with regime switching: classical and gibbs-sampling approaches with applications," *MIT Press Books*, vol. 1, 1999.
[13] D. Ramage, C. D. Manning, and S. Dumais, "Partially labeled topic models for interpretable text mining," in *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2011.
[14] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," *the Journal of machine Learning research*, vol. 3, pp. 993–1022, 2003.
[15] D. M. Blei, "Probabilistic topic models," *Communications of the ACM*, vol. 55, pp. 77–84, 2012.
[16] R. Potharaju, A. Newell, C. Nita-Rotaru, and X. Zhang, "Plagiarizing smartphone applications: attack strategies and defense techniques," in *Engineering Secure Software and Systems*. Springer, 2012.
[17] S. Nakagawa, "Relationship among perplexity word accuracy and phoneme accuracy, and drawback and modification of perplexity," in *Proc. First Int. Workshop East Asian Language Resources and Evaluation*, 1998.
[18] V. P. Mehta, "N-tier architecture," *Pro LINQ Object Relational Mapping with C# 2008*, pp. 311–345, 2008.
[19] D. Ramage, E. Rosen, J. Chuang, C. D. Manning, and D. A. McFarland, "Topic modeling for the social sciences," in *NIPS 2009 Workshop on Applications for Topic Models: Text and Beyond*, 2009.
[20] L. Cao and L. Fei-Fei, "Spatially coherent latent topic model for concurrent segmentation and classification of objects and scenes," in *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*, 2007.
[21] M. Harman, Y. Jia, and Y. Zhang, "App store mining and analysis: Msr for app stores," in *Mining Software Repositories (MSR), 2012 9th IEEE Working Conference on*, 2012.
[22] C. H. Papadimitriou, H. Tamaki, P. Raghavan, and S. Vempala, "Latent semantic indexing: A probabilistic analysis," in *Proceedings of the seventeenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, 1998.
[23] T. Hofmann, "Probabilistic latent semantic indexing," in *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, 1999.
[24] A. Sinha, A. Paradkar, P. Kumanan, and B. Boguraev, "A linguistic analysis engine for natural language use case description and its application to dependability analysis in industrial use cases," in *Dependable Systems & Networks, 2009. DSN'09. IEEE/IFIP International Conference on*, 2009.
[25] A. Sinha, S. Sutton, and A. Paradkar, "Text2test: Automated inspection of natural language use cases," in *Software Testing, Verification and Validation (ICST), 2010 Third International Conference on*, 2010.
[26] V. Gervasi and D. Zowghi, "Reasoning about inconsistencies in natural language requirements," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 14, pp. 277–330, 2005.
[27] A. V. Nefian and M. H. Hayes III, "Hidden markov models for face recognition," in *Acoustics, Speech and Signal Processing, 1998. Proceedings of the 1998 IEEE International Conference on*, 1998.
[28] Q. Mei, D. Cai, D. Zhang, and C. Zhai, "Proceedings of the 17th international conference on world wide web," in *Proceedings of the 17th international conference on World Wide Web*, 2008.
[29] T. Hastie and R. Tibshirani, "Discriminant adaptive nearest neighbor classification," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 18, pp. 607–616, 1996.
[30] A. P. Felt, M. Finifter, E. Chin, S. Hanna, and D. Wagner, "A survey of mobile malware in the wild," in *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices*, 2011.
[31] Y. Zhou, Z. Wang, W. Zhou, and X. Jiang, "Hey, you, get off of my market: Detecting malicious apps in official and alternative android markets," in *Proceedings of the 19th Annual Network and Distributed System Security Symposium*, 2012.
[32] S. Chakradeo, B. Reaves, P. Traynor, and W. Enck, "Mast: triage for market-scale mobile malware analysis," in *Proceedings of the sixth ACM conference on Security and privacy in wireless and mobile networks*, 2013.
[33] W. Enck, P. Gilbert, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. Sheth, "Taintdroid: An information-flow tracking system for realtime privacy monitoring on smartphones." in *OSDI*, 2010.