PriApp-Install: learning user privacy preferences on mobile apps' installation

Ha Xuan Son, Barbara Carminati, and Elena Ferrari

University of Insubria, Varese, Italy
{sha,barbara.carminati,elena.ferrari}@uninsubria.it

Abstract. It is undeniable that smartphones play a vital role in our lives, as their applications (apps) can be used to access various services anytime and anywhere. Despite the benefits provided by mobile apps, there are risks connected to the release of personal and sensitive data. Understanding the potential privacy risks of installing an app based on its description or privacy policy could be challenging, especially for non-skilled users. In this paper, to assist users in their app selection process, we propose PriApp-Install, a privacy-aware app installation recommendation system. It leverages semi-supervised learning to learn individual privacy preferences w.r.t mobile app installation. Learning is done based on a rich set of features modelling both the app behavior w.r.t. personal data consumption and the benefits a user can get in installing the app. We tested four learning strategies on a real dataset by exploiting three participant groups: security and privacy experts, IT workers, and crowd workers. The obtained results show the effectiveness of our proposal.

Keywords: Mobile apps, Privacy preferences and policies, Static analysis, Semi-supervised Learning

1 Introduction

Nowadays, smartphones are an integral part of our lives, and, as a result, we heavily rely on them and their apps. Several apps exist to support us in almost all our activities, whether these activities are related to our digital life, social life, real-life or working time. The result is that, on average, users install from 20 to 60 apps on their mobile devices.[1] This trend has relevant advantages: users benefit from different services almost everywhere. However, apps may collect sensitive information about users (e.g., users' profiles and habits) and their devices (e.g., locations), which may not be all needed for the app's execution.

To mitigate privacy risks, different privacy laws have been approved to limit service providers' requests for unnecessary information, such as GDPR[2] for the European countries. Also, mobile platforms have started to care more about individual privacy. For example, Android OS introduced the `permission` and

[1] https://www.statista.com/statistics/267309/number-of-apps-on-mobile-phones/
[2] `https://eur-lex.europa.eu/eli/reg/2016/679/oj`

`permission group` principle, by which developers can indicate the type and scope of collected data, and apps have to request such permissions from end-users before getting access to sensitive resources.[3] These efforts reduce the set of permissions required by apps to some extent [25], but users are still largely unaware of the privacy implications of some permissions they grant to apps.

For this reason, several approaches have recently been proposed to develop recommendation systems supporting users in making more privacy-aware decisions on app installation. However, most of them (a) assign a risk score to target apps and (b) suggest not installing apps with a high-risk score. Therefore, their main focus is the definition of the risk score metric. For instance, some approaches suggest determining the risk by comparing the app's requested permissions with the app's description [3,23]. Other studies focus on static analysis of the app's code to capture the app's behavior w.r.t personal data collection (see Section 6 for more details).

However, we envision these approaches not fully supporting end-users in the app installation process. Focusing only on the risk of an app w.r.t personal data consumption, they ignore other relevant factors to be considered when installing an app. First, the possible risk must be compared with the benefits that an app can bring to a specific user. For instance, a user may decide to install a risky medical app since it collects sensitive information because it is essential for him/her as it monitors a disease he/she suffers. The other important aspect is that privacy is subjective by nature, so one-size-fits-all solutions do not work properly. Users might have different opinions w.r.t. the installation of apps with the same risk level.

To cope with all these issues, this paper proposes PriApp-Install, a tool whose goal is to learn user preferences on app installation to suggest whether to install a new app or not.[4] Learning exploits three main dimensions: the app's behavior w.r.t. personal data usage; the benefits that installing the app could bring to target users; and the sharing of collected personal information with third parties. All this information is collected through (a) the app market, (b) the static analysis of the app's code, and (c) the analysis of the app's privacy policy.

To balance users' effort and the prediction quality, we opt for semi-supervised learning [20]. In particular, we have experimentally tested four different learning strategies over the selected features, and we show that, among different semi-supervised approaches, the ones that better fit the considered scenario are those based on ensemble learning [12]. We also show that semi-supervised learning provides better accuracy than supervised approaches (i.e., random forest, linear regression, and SVM) even with a small training dataset.

We have tested our learning approaches with three different groups of participants. First, we considered the feedback of 51 cybersecurity and privacy experts (S&P experts) who work in academic institutions. We then collected feedback from 90 IT workers, i.e., junior and senior developers in industrial institutions from different countries. Finally, we used a crowd-sourcing platform to have a

[3] https://developer.android.com/guide/topics/permissions/overview
[4] Implementation is available at `https://github.com/SonHaXuan/PriApp-Install`

more significant (i.e., 300 participants), more balanced (i.e., gender, ages), and more heterogeneous (i.e., different nationalities and education levels) dataset. The obtained results show that around 94.51% of S&P experts, 86.39% of IT workers, and 80.69% of crowd workers were satisfied with the app installation decisions suggested by our tool.

The remainder of this paper is organized as follows. Section 2 describes the overall architecture of PriApp-Install. Sections 3 and 4 present the adopted learning approaches and dataset, respectively. The evaluation and related work are presented in Sections 5 and 6, respectively. Section 7 concludes the paper. Finally, the appendix includes information about the adopted metrics.

2 Overall architecture

PriApp-Install aims at assisting users in deciding whether to install an Android app or not. The decision is mainly based on the privacy risks related to the information collected and shared with third parties and the benefits the users might get from the app. Moreover, in developing our tool, we acknowledge that users might have different perceptions of privacy risks implied by apps' data collection [8], or more in general of privacy, as well as, on possible benefits. To consider this subjective dimension, PriApp-Install leverages semi-supervised learning to build a classifier for predicting app installation decisions based on users' feedback.

The learning process relies on a rich set of features describing different aspects of the target app relevant to the decision process. First of all, thanks to the information available in app store,[5] we obtain some *app's basic information*, including the app's name, developers' name, app's category, app's description, and so on.

Additionally, we consider the *app's privacy policy* to collect other relevant information. In particular, we parse the privacy policy to determine the declared purposes for data collection and third-party usage. We focus on these information because they are always present in the analyzed apps' privacy policies. In contrast, other information, such as the retention period (i.e., for how long the server stores the data), is present only in a few apps' privacy policies. However, this publicly available app's information is not sufficient for users to fully understand the app's behavior w.r.t personal data consumption. To overcome this limitation, PriApp-Install also considers the *app's source code* to determine which data is collected by an app.

In particular, in this paper, we leverage on the static analysis tool developed in [15] to model and capture the API usage w.r.t. the collection of personal data. [15] models the app's behavior w.r.t personal data collection through the definition of the *app's signature* which is generated through the static analysis of the app's source code. In particular, given a data type dt and an app app, [15] models its behavior w.r.t. collection of data of type dt as an array $sign^{dt}_{app}$, containing

[5] Android app store: `https://play.google.com/store/apps`

an element for each specific API able to retrieve a data of dt's type. $sign^{dt}_{app}$'s elements have value 1, if the corresponding API is present in app's source code; 0, otherwise. [15] focuses on seven types of personal data, namely: user location (e.g., city, country), media (e.g., users' image, video, audio), connection (e.g., activity on Bluetooth, NFC), hardware (e.g., camera, USB devices), telephony (e.g., contacts info, phone number), user profile (e.g., gender, name), and health & fitness (e.g., heart rate, body fat). Therefore, the app's signature consists of seven arrays $sign^{dt}_{app}$, one for each of the considered data type dt.

By analyzing 30 permissions with significant security/privacy impact, provided by Google[6], [15] has determined 66 potentially dangerous APIs that collect data of the above-mentioned data types.[7] The app signature is therefore an array of 66 elements, corresponding to: $sign^{Loc}_{app}$ (6), $sign^{Med}_{app}$ (24), $sign^{Con}_{app}$ (13), $sign^{Har}_{app}$ (8), $sign^{Tel}_{app}$ (8), $sign^{U\_P}_{app}$ (2), and $sign^{H\&F}_{app}$ (5).

Example 1. Let us consider the Google Fit: Activity Tracking app signature ($sign_{G\_F}$). For simplicity, we will focus on the Location data type only. Its corresponding array $sign^{Loc}_{G\_F}$ in $sign_{G\_F}$ can be defined as follow:[8] $sign^{Loc}_{G\_F} = 100000$, where the first element has value "1", representing that the android.location API is used in the app code; whereas, the other elements' value is "0" (i.e., 5 elements), since the corresponding APIs are not used in the app code.

Therefore, we model an app as a set of features extracted by its *basic information* provided in the app market, *source code* analysis, and *privacy policy*, as the following definition states.

Table 1. app's dimensions and corresponding fields

Dimension	Collection process	App's features
basic information	crawled from the app market	an, dn, cat
data consumption behavior	app's signature, determined via the static analysis tool proposed in [15]	$sign$
privacy policy	data collection purposes and third party usages, extracted by exploiting [22]	pp, $3pt$

Definition 1. ***App's features:*** *Given an app, we model its features as a tuple* $(an, dn, cat, sign, pp, 3pt)$*, where: an is the app's name; dn is the developer's name; cat is the app's category; sign is the app's signature; whereas pp and* $3pt$ *model the data collection purposes and third-party usages, respectively, as stated in the app's privacy policy. pp is defined as a set of pairs, each one denoting a purpose and the corresponding collected data.*[9] *Finally,* $3pt$ *is a set of triples,*

[6] `https://developer.android.com/reference/android/Manifest.permission`
[7] We presented the APIs list and dangerous permissions at `https://bit.ly/3qm5VT4`
[8] Arrays for the other personal data types can be similarly defined.
[9] pp contains $not\_specified$ if the policy does not specify any purpose

each one denoting the third party name, the corresponding shared data and the sharing purpose.[10]

Table 1 summarizes the considered dimensions, the corresponding app's features and the way we collect them.

Example 2. Let us consider Google Fit: Activity Tracking app and let $sign_{G\_F}$ be its app signature, defined similarly to its location component $sign^{Loc}_{G\_F}$ given in Example 1. According to Definition 1, its features are modelled as follow:

$GoogleFit$=(Google Fit: Activity Tracking, Google LLC, Health&Fitness, $sign_{G\_F}$, {(analysis, user profile), (marketing, $not\_specified$), (maintenance, {location, user profile})}, {(ads, Google Ads, $not\_specified$)}).

To learn user judgment w.r.t apps' installation, we build a labeled training dataset $LApp$, by asking users to assign a judgment (label) for each *app* in the training set. This judgment indicates whether the user wants to install the corresponding app or not, based on its features (see Definition 1). If the user finds the provided information insufficient, he/she may decline to respond. The corresponding labels are therefore: Yes (Y), No (N), and Maybe (M). We used $LApp$ to test different learning approaches described in the next section.

3 Learning Approaches

Literature offers several learning approaches that could be used for our goal. However, as mentioned in Section 2, we opt for semi-supervised learning [20], which has the advantage of achieving good accuracy with a small labeled training set (see subsect. 5.3). It allows us not to require users to label many apps since this would negatively impact the usability and acceptability of our solution. To select the best semi-supervised learning approach, we test four different learning strategies with different complexity (the results are reported in Section 5). These four strategies are described in the following sections.

3.1 Naive and category-based prediction models

As learning algorithm we exploit the Expectation-maximization (EM) algorithm [2]. EM is an iterative method to find maximum likelihood or maximum a posteriori (MAP) estimates of θ parameters[11] in statistical models, where the model depends on unobserved latent variables. The EM algorithm alternates between an expectation (E) and a maximization (M) steps. The first step computes the maximum likelihood estimation for θ, quantified by the log-likelihood of all the items, based on the current estimation for θ parameters. In the second step, the algorithm updates θ to maximize the likelihood. In our case, the first step trains a classifier with only the available labeled apps and uses the classifier to

[10] If shared data are not specified in the privacy policy, $3pt$ is set to $not\_specified$.
[11] θ parameters aim at optimizing the label probability.

assign probabilistically weighted class labels to each target app, by calculating the expectation of the missing class labels. The algorithm, then, trains a new classifier using all the apps (i.e., both the labeled and pseudo-labeled apps).

Our baseline is a naive approach that considers all *app*'s features (i.e., an, dn, cat, $sign$, $pp, 3pt$) to predict the pseudo-labels of the target apps.

The second approach, i.e., category-based, takes in consideration that apps belonging to the same category might have similar behavior (in terms of required personal data). At this aim, the category-based approach groups all apps with the same category to build the classifier pseudo-labeling the target apps.

Algorithm 1 EM($LApp$,$UApp$)

```
1:  input: the set of labeled apps LApp, the set of unlabeled apps UApp.
2:  output: labels for UApp.
3:  for each app_i ∈ UApp do
4:     θ = EM(LApp);
5:     Calculate log-likelihood L(θ);
6:     convergence = false;
7:     Dataset = LApp ∪ {app_i};
8:     while convergence==false do
9:        E-step: Estimate likehood on Dataset based on θ parameter values;
10:       M-step: Update θ parameter values;
11:       Calculate new log-likelihood L(θ_new);
12:       if |L(θ_new) - L(θ)| ≤ ϵ then
13:          convergence=true;
14:          lab_app_i = argmax_θ(p(app_i, lab_app_i|θ)p(θ));
15:          UApp = UApp - {app_i};
16:          LApp = LApp ∪ {(app_i,lab_app_i)};
17:       else
18:          L(θ) = L(θ_new);
19:       end if
20:    end while
21: end for
22: return LApp;
```

Algorithm 1 describes how the pseudo-labels of each app in the unlabeled dataset are generated using the EM algorithm. This algorithm is used both for naive and category-based approaches. In the case of the naive approach, Algorithm 1 receives as $LApp$ all apps. In contrast, in the case of category-based, we pass to Algorithm 1 only apps with the same category. The algorithm is thus re-executed for each category, obtaining a different prediction model.

Algorithm 1 executes EM to estimate the θ parameters using only the labeled dataset, i.e., $LApp$, and the pseudo-labeled apps from the previous iteration (see line 4). Line 5 calculates the initial value of the log-likelihood $L(\theta)$. The expectation step calculates the membership probability for each app and each label (i.e., yes, no, maybe) (line 9). In the maximization step, parameters are

optimized for each label based on the current membership probability (line 10). The expectation and maximization phases are iterated until log-likelihood does not change (i.e., the difference between values of log-likelihood computed in two subsequent iterations is less than a value ϵ (see line 12)). Then, the pseudo-label returned for each app (app_i) is associated with the highest membership probability (line 14).[12] Algorithm 1 repeatedly labels the unlabeled apps in $UApp$ until this set is empty (the for-loop from line 3 to line 21). Finally, it returns $LApp$, including the original labeled apps and the pseudo-labeled ones (line 22).

3.2 Simple and advanced ensemble-based prediction models

The first two approaches benefit from being efficient in terms of time required for building the prediction model [19]. However, they do not exploit the correlations that might exist among the considered features, which can affect the accuracy during label prediction [9]. We believe that different combinations of apps' features might impact the prediction models, and thus the app installation decisions. For instance, prior studies have found that knowledge of the data collection purposes (i.e., *pp*), as well as third party usages (i.e., *3pt*), can increase user awareness when installing an app [7,21].

For these reasons, we further apply two ensemble-based algorithms with the aim of better capturing user behavior w.r.t app's installation. Ensemble-based learning combines a group of sub-classifiers to create a more robust prediction model [11]. In practice, we use ensemble learning to define prediction models on the app's features subsets. Then, we combine the most confident one.

Table 2. Classifiers for the simple (Sim) and advanced (Adv) ensemble-based approach

	Classifier 1 features	**Classifier 2 features**	**Classifier 3 features**
Sim	*an*, *dn*, *cat*	*sign*	*pp*, *3pt*
Adv	*an*,*dn*,*cat*,*sign*	*sign*,*pp*,*3pt*	*pp*,*3pt*,*an*,*dn*,*cat*

Algorithm 2 implements the simple ensemble-based approach. It exploits the app's features corresponding to each app's dimension to define a different classifier (line 3). In particular, as reported in the first row of Table 2, the first classifier is built only on *an*, *dn*, and *cat* features; the second on the *sign* feature; and the third on *pp* and *3pt* features.

Once the classifiers have been built, given a target app (app_i), the `Ensemble` algorithm identifies to which label app_i most likely belongs to by using the bagging method [12] (see lines 6 - 11). Bagging is an effective method for ensemble learning, where the final label is assigned by computing the average of membership probabilities returned by the obtained classifiers (line 12). Algorithm 2 repeatedly labels the unlabeled apps in $UApp$ until $UApp$ is empty (the for-loop

[12] `argmax` is used for finding the label (i.e., Y, N, MB) with the highest probability.

from lines 4 to 15). Finally, the algorithm returns $LApp$, including the original labeled apps and the pseudo-labeled apps (line 16).

Algorithm 2 Simple_Ensemble($LApp$,$UApp$)

1: **input**: labeled apps $LApp$, unlabeled apps $UApp$.
2: **output**: labels for $UApp$.
3: Let $dimension$ = {(an,dn,cat),sign, (pp,3pt)}
4: **for** each $app_i \in UApp$ **do**
5: θ_{ens} = EM($LApp$);
6: **for** $X \in dimension$ **do**
7: Let $LApp_X$ be the projection of $LApp$ on X;
8: Let app_{i_X} be the projection of app_i on X;
9: θ_X = EM($LApp_X$, app_{i_X});
10: $\theta_{ens} = \theta_{ens} \cup \theta_X$;
11: **end for**
12: $lab_{app_i} = argmax_{\theta_{ens}}(p(app_i, lab_{app_i}|\theta_{ens})\ p(\theta_{ens}))$;
13: $UApp = UApp$ - $\{app_i\}$;
14: $LApp = LApp \cup \{(app_i, lab_{app_i})\}$;
15: **end for**
16: **return** $LApp$;

Implementation of the advanced ensemble-based approach is similar to that of the simple ensemble-based one (Algorithm 2), being the only difference the app's features used to build the classifiers, which are those reported in the second row of Table 2. In particular, in the advanced ensemble-based learning, we build three classifiers exploiting correlations among features belonging to different dimensions. Indeed, users might have different opinions on apps by considering features of different dimensions simultaneously. As an example, Lin et al. [6] found that users are not willing to install dictionary apps since they access their locations (i.e., data collection). However, when users realize that this data is needed to compute location-based trending words (i.e., purpose), they are fine to install them. As such, we believe that features of different dimensions should be combined together. Therefore, in the advanced ensemble-based approach we design each classifier to combine two out of the three dimensions. For instance, the first classifier is built by considering features belonging to the first and second dimension, namely an, dn, cat, and $sign$, whereas the second classifier combines features belonging to the second and third dimension (see the second row of Table 2). Figure 1 describes the architecture of PriApp-Install and the supported learning strategies.

4 Dataset

To generate the app dataset to test our learning strategies, we selected a list of the most-downloaded apps from 15 popular categories. Table 3 reports the

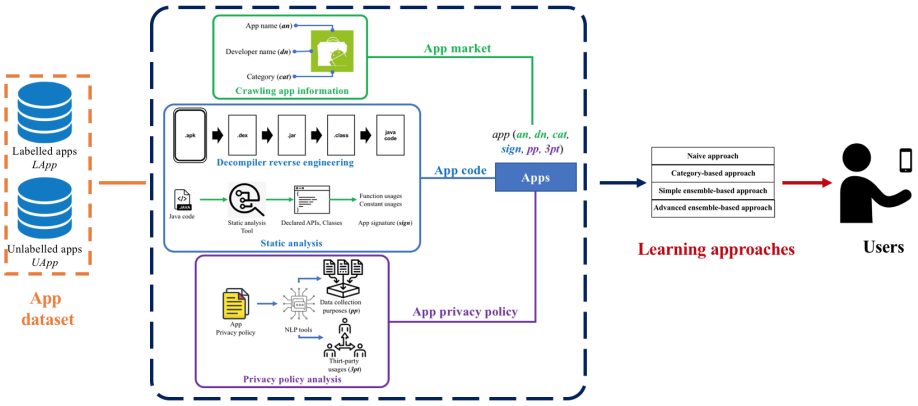

Fig. 1. PriApp-Install architecture

considered categories. In total, we crawled 32,856 apps in Feb. 2021. We crawled app's basic information from the app market, namely app's name, developer's name, and category. Also, we downloaded the APK files of the selected apps.

Table 3. App dataset (i.e., #apps in Java code and Privacy policy)

Category	Java code	Privacy policy	Category	Java code	Privacy policy	Category	Java code	Privacy policy
Beauty	1311	1061	**Food & Drink**	2115	1515	**Shopping**	2087	1537
Business	2258	1568	**Health & Fitness**	1546	1167	**Social**	1506	1158
Education	2249	1782	**Maps & Navigation**	1606	1240	**Sports**	1655	1303
Entertainment	1838	1407	**Medical**	1733	1232	**Tools**	1799	1387
Finance	2225	1747	**Music & Audio**	1425	1203	**Travel & Local**	1857	1513

To generate the app's signature, we first leveraged decompiler reverse engineering to obtain the Java code from the corresponding APK (see Figure 1). Among the 32,856 crawled apps, we successfully decompiled 27,210 of them (about 82.82%).[13] Then, we applied the static analysis approach in [15] to collect the APIs usage for the seven considered data types (i.e., 66 APIs), namely location, media, connection, hardware, telephony, user profile, and health & fitness. To determine if a declared API is exploited by an app, we checked whether the functions/constants of the target API are used in the app's source code.

To minimize the amount of information that participants to our experiments must check, we show only the information on API usage for each target app (66 APIs), rather than information on classes (nearly 1.4K classes) or functions/constants usage (more than 13.6K functions/constants).[14]

Finally, to analyze privacy policies to determine data collection purposes and third-party usage, we exploited the natural language processing (NLP) tool de-

[13] Failure to decompile was primarily due to code obfuscation.

[14] On average, in our dataset, an app uses 6 APIs, 48 classes, and 238 functions/constants to collect personal data.

fined by Wilson et al. in [22]. Among all 27,210 decompiled apps, we successfully collected privacy policies of 20,820 apps (approximately 76.52%). For data collection purposes (i.e., component *pp* of Definition 1), we considered the 18 most popular ones, such as Profiling, Analysis, Statistical, Advertisements, Maintenance. We also determined the corresponding collected data (by restricting our analysis to the seven personal data types) for each purpose mentioned in the app's privacy policy. Additionally, we captured the third-party usage (i.e., component $3pt$ of Definition 1) by extracting from the privacy policy the name of the third party with which data are shared, the corresponding usage purposes, and collected data. For usage purposes, we considered the five most common ones (i.e., Payment, Delivery, Marketing, Advertisement, and Analysis).

5 Experiments

We carried out several experiments to evaluate the effectiveness of the proposed learning approaches in capturing users' preferences regarding app's installation.

5.1 Experiments Setting

In our experiments, users have been involved both to label the training set and evaluate if a prediction model has correctly labeled a target app. For this purpose, we developed a web application through which participants label the training apps in the learning phase (i.e., training dataset), and give their feedback on the labels generated by the prediction models in the testing phase (i.e., test dataset). During the learning phase, participants are asked to associate a decision (i.e., Y, N, M) with ten apps that belong to two categories, randomly selected from the 15 categories in Table 3 (i.e., five apps/category). For each app, we show its features (those listed in Definition 1), namely: app's name, developer's name, category, its signature, data collection purposes, and third-party usages.

Once the labeling phase is terminated, the collected training dataset is elaborated by the learning algorithms illustrated in Section 3. To evaluate the learning strategies, the web application shows 16 new apps to the experiment's participants to build the test dataset for the four proposed approaches (i.e., four target apps/approaches). For each target app, the participants give feedback on the labels, i.e., agree (Y) or disagree (N) - and in case of disagreement, they provide the correct label. Finally, the web application asks the participants whether they are satisfied with the labels assigned by the four approaches. The participant can answer with yes, no, or maybe.

In the survey, we set a timer to track how long participants spend in evaluating an app. Reading basic app information takes on average 26 seconds, whereas participants spend approx. 46 seconds to (a) check information regarding API usages (i.e., app's signature) and privacy policy (i.e., third party usage, and data collection purposes) and (b) answer the proposed questions. The total average time to complete the survey is therefore approximately 30 minutes for 26 apps (i.e., 11 and 19 minutes for the training and test phase, respectively).

5.2 Participants

We selected three groups of participants to evaluate our prediction models' accuracy and app installation suggestions.

The first group includes S&P experts who work in academic institutions. For the recruitment process, we send the invitation email to the members of different S&P research groups. Specifically, we collected feedback from 51 S&P experts in different countries (e.g., USA, Italy, Switzerland, Germany, Belgium, Greece, Vietnam, Morocco). Participants have an average age of 28.9 (from 24 to 42). 84.31% of them were male (43 participants). Among the 51 participants, 14 are lecturers/professors, whereas the others are Ph.D. students and post-docs.

The participants in the first group have a strong background in security and privacy. However, not all of them work on software or mobile development. Therefore, to check if this background impacts the results, IT workers were selected as the second group to assess the predictions of our models. We collected high-quality feedback from 90 IT workers, junior and senior developers in industrial institutions from different countries (e.g., France, Singapore, Vietnam, Taiwan, Australia). IT workers have an average age of 29.6 (between 23 and 38). Among these participants, 18.88% (17 participants) were female. For the geographical distribution, 86.67% are working in Asian countries (78 participants), whereas the remaining participants are working in European countries (10 participants - 11.11%) and Australia (2 participants - 2.22%).

The third group aims at ensuring a good number of participants from different regions, nationalities, ages, and educational levels. At this purpose, we used the Microworkers crowdsourcing platform[15] for the enrollment of participants (called workers). To achieve diversity in evaluating the prediction models, we selected participants from five different regions. The regions are Latin America (e.g., Brazil, Argentina, Venezuela), USA - Western (e.g., USA, Canada, Australia), Western Europe (e.g., Italy, Spain, France), Eastern Europe (e.g., Poland, Greece, Hungary), Asia & Africa (e.g., Singapore, Vietnam, Ethiopia). For each geographic area, we try to balance gender, nationalities, ages, and educational levels. Specifically, we received 60 feedback per geographic area - a total of 300 participants - 157 - of them 52.33% were male. The participants came from 81 countries (e.g., UK, Italy, India, Spain, Portugal, USA), with an average age of 28.2, being the oldest worker 48, and the youngest 18. Also, they have different educational levels (e.g., student, bachelor) and backgrounds (e.g., accountant, teacher, manager). Moreover, nearly 60% of the participants reported to have a bachelor's degree or equivalent, whereas 10% of them had a master's degree or a Ph.D. The workers were given $1.5 for each successful feedback.

As the proposed approach depends on the participants' input on the training and testing datasets, we removed low-quality feedback. We use three characteristics to detect low-quality feedback: time to complete the survey, the number of labelled apps, and the workers corresponding answers. Therefore, we measured how much time a user devotes to give feedback for an app. If it was less than

[15] https://www.microworkers.com/

a desirable time (1 minutes/app), we suspect that the participant did not read the app's information carefully; thus, his/her feedback was removed from the dataset. As we want to analyze the accuracy and satisfaction level for all four approaches, participants' feedback was considered only when they have evaluated all 26 apps. Finally, we removed feedback from participants that gave a unique label to all apps (e.g., Maybe for all the 26 apps). In total, we removed 45 low-quality feedback from 8 experts, 7 IT workers, and 30 crowd workers.

5.3 Evaluation

To evaluate the effectiveness of the proposed learning approaches, we measure the accuracy, precision, recall and F1 (see Appendix A for more details).

In the first experiment, we compare the accuracy of the proposed prediction models, namely naive, category-based, simple ensemble-based, and advanced ensemble-based. In the second experiment, we compare the accuracy obtained by our semi-supervised approaches with the ones obtained through traditional supervised learning techniques (i.e., Linear Regression, Random Forest, and SVM).

Prediction models' accuracy In this experiment, we test the accuracy of PriApp-Install prediction models in labelling target apps. As shown in Figure 2, around 72.22%, 76.72% and 78.27% of S&P experts, **crowd workers**, and IT workers' choices in labeling app installation are correctly predicted by the naive prediction model, respectively. It is the lowest accuracy of the four prediction models. Whereas, simple ensemble-based and advanced ensemble-based have the highest accuracy, with around 89.22% and 92.89% for S&P experts, 86.24% and 84.79% for crowd workers, and 86.29% and 84.21% for IT workers. For the remaining models, approximately 78.04%, 79.63%, and 80.04% of the labeling predictions done by the category-based model are made correctly for crowd workers, S&P experts, and IT workers, respectively. This experiment shows that the performance of both ensemble approaches is better than those of the other approaches. In particular, the ensemble-based approaches capture the app installation behavior of S&P experts with outstanding accuracy (about 90%). It highlights that relationships among elements of an app play a vital role in the participants' decisions.

Satisfaction level For this experiment, the web application showed participants a set of apps with the labels generated by our prediction models and asked them if they were satisfied with the model decisions. In the testing phase, 16 target apps and corresponding predictions were presented to participants (4 apps for each approach, randomly presented to the participants).

As shown in Figure 3, around 66.67% of S&P experts, 62.80% of crowd workers, and 77.84% of IT workers are satisfied with the decisions taken using the naive prediction model, and this represents the lowest satisfaction level of the four prediction models; whereas around 73.15%, 79.63%, and 81.13% are fine

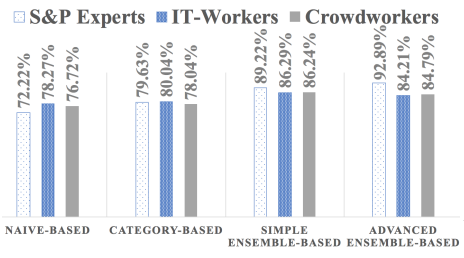

Fig. 2. Four prediction models' accuracy

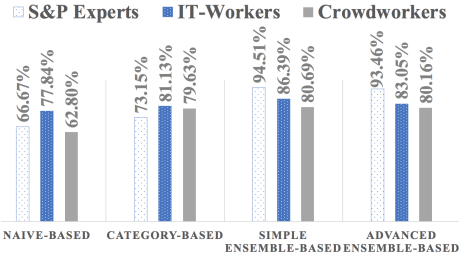

Fig. 3. Participants satisfaction level

with the decisions taken using the category-based prediction model. The highest satisfaction levels are obtained with the ensemble-based prediction models. Specifically, the satisfaction level of the decisions taken by advanced ensemble-based model ranges from 80.69% (crowd workers) to 94.51% (S&P experts); whereas, the satisfaction level of the simple ensemble-based model ranges from 80.16% (crowd workers) to 93.46% (S&P experts). This experiment further confirms that ensemble approaches outperform the others.

The difference between the two ensemble-based approaches is that if the simple ensemble-based approach considers essential relationships among the app's features of the same type, the advanced ensemble-based approach considers a combination of dimensions. These relationships may affect the app installation decision of the participants. Indeed, some participants' feedback points out that combining the app's fields (which is exploited by the ensemble-based prediction models) is essential for predicting the labels of target apps. For instance, **Expert 27**: *"I will install the app if it does not collect my location ... the collected data is used for healthcare purposes"*; **Crowd 181** : *"I don't want to use apps that collect too much personal information and share it with third parties..."*; or **IT-worker 31**: *"... In the same app's category, I only install the app in the same app's category if I know the collected data and its purposes ..."*.

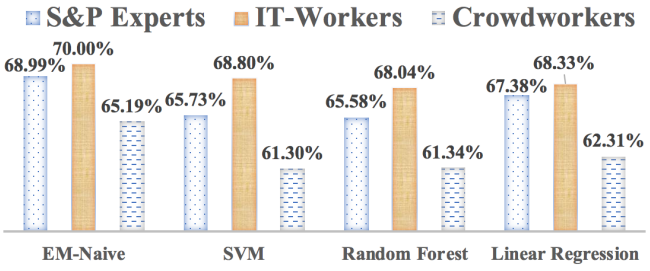

Fig. 4. Accuracy of the EM-based naive prediction models vs SVM, Random Forest, and Linear Regression

Supervised vs. semi-supervised learning As mentioned in Section 3, we select semi-supervised EM as our learning approach, because we aim at reducing participant burden on the training dataset. Nevertheless, we also tested the performance of hard clustering supervised machine learning techniques. For this purpose, we compare the accuracy of our EM-based naive prediction model against three supervised hard-clustering algorithms, namely Linear Regression, Random Forest, and SVM. We use the naive prediction model to compare since it is the weakest among the four proposed models. The naive approach exploits all app's features to define the prediction model. Figure 4 shows the results, with a training dataset consisting of 6 apps and a testing dataset of 16 apps. We randomly selected 6 training apps instead of 10 since we would like to analyze the performance given a small training dataset. The results show that the performance of our naive prediction model is the highest for the three participant groups (see Figure 4). Moreover, the accuracy of the three supervised hard-clustering prediction models is less than that of the naive prediction model because these models need more extensive training datasets for the prediction process. This motivates us to use semi-supervised rather than supervised learning.

Table 4. Comparison of the four prediction models for the test dataset

		Naive prediction model			Category-based prediction model			Simple ensemble-based prediction model			Advanced ensemble-based prediction model		
		Y (%)	N (%)	M (%)	Y (%)	N (%)	M (%)	Y (%)	N (%)	M (%)	Y (%)	N (%)	M (%)
S&P experts (N=51)	Precision	63.38	82.61	71.05	78.79	81.54	78.82	92.31	86.49	88.46	94.53	86.36	94.87
	Recall	90.00	60.00	76.06	69.33	85.48	84.81	83.72	96.97	88.46	95.28	95.00	84.09
	F1	74.38	69.51	73.46	73.76	83.47	81.71	87.81	91.43	88.46	94.90	90.48	89.16
IT workers (N=90)	Precision	83.33	67.54	84.19	75.48	85.50	80.00	91.18	89.10	76.92	81.11	87.50	91.22
	Recall	75.43	81.91	78.17	81.25	75.68	82.76	81.58	89.10	90.10	94.81	68.86	74.29
	F1	79.19	74.94	81.07	78.26	80.29	81.36	86.11	89.10	83.33	87.45	77.06	81.89
Crowd workers (N=300)	Precision	79.12	74.59	76.43	74.61	80.70	79.04	87.35	84.02	87.23	85.06	81.73	86.00
	Recall	82.42	71.65	76.43	78.28	76.35	79.34	86.54	88.26	83.25	94.05	66.93	73.71
	F1	80.74	73.09	76.43	76.40	78.46	79.19	86.94	86.13	85.19	89.33	73.59	79.38

F1 Measure Finally, we measure the F_1 score for each label (Y, N, M) in the test datasets (see Table 4). F_1 score considers both the precision and the recall aspects (see Table 6 in Appendix A). Our analysis proves that both ensemble-based approaches work better than naive and category-based for the test dataset. It means that combining multiple aspects/classifiers helps predict participants' app installation decisions better for both participants with and without a security/privacy background.

From our experiments, we obtain an accuracy for the three participant groups ranging between 86.24% and 92.89%. We believe that this is acceptable since we strike a balance between the user burden and the prediction quality. However, we plan to improve our tool by testing alternative prediction models, especially for less skilled participants.

When a user decides to install Pri-App Install on his/her smartphone, it takes approx 11 minutes to train the tool (with 10 sample apps) in such a way

that it learns the privacy preferences of the user w.r.t the app's installation. This training phase, although time-consuming, is done only once when the tool is installed.

6 Related work

Literature offers several proposals for app recommendation. For instance, some papers propose tools to determine if target apps are benign based on over-privileged apps' detection. As an example, Taylor et al. [17,18] propose a contextual permission analysis tool, called SecuRank, which allow users to detect if any of their installed apps could be replaced with functionally similar apps that collect less sensitive data. By running SecuRank on the app market, they found that up to 50% of privacy-breaching apps could be replaced with more privacy-preserving apps.

Exploiting the same approach, Tingmin et al. [23] introduces the PERSCRIPTION platform that aims to replace the app's description with a new personalized one based on users' preferences on security concerns and linguistic preferences. More precisely, to model user's security concerns, given a target app, PERSCRIPTION asks user to label as grant/deny all app's default permissions. Additionally, to generate the personalised description w.r.t. linguistic preferences, PERSCRIPTION classifies users into distinct personalities based on their behaviors. Specifically, they considered five personality traits models [4], (e.g., Extraversion, Agreeableness, Conscientiousness, Neuroticism, and Openness) to bridge between the users' mobile behaviours (i.e., app adoption) and linguistic preferences. To each personality a set of preferred words (linguistic preference) is associated. The final app's description is then generated by using proper words, based on users personality and highlighting risky permissions.

Zhiqiang et al. [24] proposes the FideDroid framework that predicts permissions needed by an app by analyzing its description. After that, FideDroid compares the inferred permissions with those marked as suspicious and unnecessary by determining whether the predicted permissions from the app's description are used in the app's code. Moreover, Huseyin et al. [1] proposes a model, called Attention, to discover inconsistencies between the app's descriptions and the requested permissions, by leveraging on recurrent neural networks (RNNs) [5].

However, all the above-mentioned approaches: (a) only estimate the risk of installing an app without considering the possible benefits; and (b) only consider the app's requested permissions that do not accurately describe the actual behavior of an app w.r.t. the consumption of personal data. Indeed, many studies (e.g., [10]) have shown that apps could actually exploit more permissions than what they provide in the app market.

Other approaches take this issue into account by leveraging on app's code analysis. For instance, Zhang et al. [26] develops a cloud-based system, called Privet, that logs app's behavior w.r.t resources' usage (e.g., CPU, memory) and personal data collection (e.g., location, media). For dangerous API usages, Zhuo et al. [13] exploits static analysis to generate a graph representation of the data

flow among API classes and possibly personal data collection. These graphs are trained on a set of malicious and benign apps. The two graphs are then compared to determine the set of dangerous APIs. Son et al. [15,16] leverage an app's static analysis to model an app's behavior w.r.t data collection on the basis of APIs, classes, functions, and constant usage. They measure an app's risk level by quantifying how much the app's behaviour diverges from the behavior of most apps in the same category.

Although those approaches show some similarities with our proposal, in that we also leverage on static analysis to have a more precise assessment of the app's collected personal data (that we borrow from [15]), all these approaches only estimate the privacy risk connected to a target app. In contrast, the goal of our proposal is not to propose yet another privacy risk measure for mobile apps. Rather, we design a tool that learns from end users their preferences w.r.t app's installation. This tool can complement the risk analysis performed by the above mentioned approaches. Learning takes into account not only the privacy risk of an app (in terms of personal data it consumes and share with third parties), but also the benefits an app can provide (in terms of delivered service), as well as the subjective aptitude that each user might have towards privacy.

7 Conclusion

This paper proposed PriApp-Install, a tool to guide users in selecting the apps to install on their smartphones based on their subjective assessment of utility and privacy risks. PriApp-Install leverages semi-supervised learning and a rich set of features to determine user privacy preferences on app's installation. We have experimentally tested our learning approaches by using feedback from different set of participants (i.e., security and privacy experts, IT workers, and crowd workers). The experimental results for ensemble-based approaches are promising. In the future, we plan to perform a more extensive experimental evaluation. We also plan to test more sophisticated learning strategies, such as those combining semi-supervised classification with deep learning. This combination may increase performance by, at the same time, not requiring a considerable amount of resources, time, and effort.Finally, we plan to extend the current approach to be adopted in the Personal Data Storage (PDS) [14] architecture, where users can store and control access to their personal data.

Acknowledgments

This work has received funding from RAIS (Real-time analytics for the Internet of Sports), Marie Skłodowska-Curie Innovative Training Networks (ITN), under grant agreement No 813162 and from CONCORDIA, (Cybersecurity Competence Network) supported by H2020 Research and Innovation program under grant agreement No 830927. The content of this paper reflects only the authors' view and the Agency and the Commission are not responsible for any use that may be made of the information it contains.

References

1. Alecakir, H., Can, B., Sen, S.: Attention: there is an inconsistency between android permissions and application metadata! International Journal of Information Security pp. 1–19 (2021)
2. Borman, S.: The expectation maximization algorithm-a short tutorial. Submitted for publication **41** (2004)
3. Feng, Y., et al.: Ac-net: Assessing the consistency of description and permission in android apps. IEEE Access **7**, 57829–57842 (2019)
4. John, O.P., Srivastava, S., et al.: The Big-Five trait taxonomy: History, measurement, and theoretical perspectives, vol. 2. University of California Berkeley (1999)
5. Jones, K.S., et al.: Readings in information retrieval. Morgan Kaufmann (1997)
6. Lin, J., et al.: Expectation and purpose: understanding users' mental models of mobile app privacy through crowdsourcing. In: Proceedings of the 2012 ACM conference on ubiquitous computing. pp. 501–510 (2012)
7. Martin, K., Shilton, K.: Putting mobile application privacy in context: An empirical study of user privacy expectations for mobile devices. The Information Society **32**(3), 200–216 (2016)
8. Nguyen, T.T., et al.: Measuring user perception for detecting unexpected access to sensitive resource in mobile apps. In: Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security. pp. 578–592 (2021)
9. Nigam, K., Ghani, R.: Analyzing the effectiveness and applicability of co-training. In: Proceedings of the ninth international conference on Information and knowledge management. pp. 86–93 (2000)
10. Olukoya, O., et al.: Security-oriented view of app behaviour using textual descriptions and user-granted permission requests. Computers & Security **89** (2020)
11. Polikar, R.: Ensemble based systems in decision making. IEEE Circuits and systems magazine **6**(3), 21–45 (2006)
12. Polikar, R.: Ensemble learning. In: Ensemble machine learning. Springer (2012)
13. Singh, A.K., et al.: Experimental analysis of android malware detection based on combinations of permissions and api-calls. Journal of Computer Virology and Hacking Techniques pp. 209–218 (2019)
14. Singh, B.C., Carminati, B., Ferrari, E.: Privacy-aware personal data storage (p-pds): Learning how to protect user privacy from external applications. IEEE Transactions on Dependable and Secure Computing (2019)
15. Son, H.X., Carminati, B., Ferrari, E.: A risk assessment mechanism for android apps. In: 2021 IEEE International Conference on Smart Internet of Things (SmartIoT). pp. 237–244. IEEE (2021)
16. Son, H.X., Carminati, B., Ferrari, E.: A risk estimation mechanism for android apps based on hybrid analysis. Data Science and Engineering pp. 1–11 (2022)
17. Taylor, V.F., Martinovic, I.: Securank: Starving permission-hungry apps using contextual permission analysis. In: Workshop on Security and Privacy in Smartphones and Mobile Devices. pp. 43–52 (2016)
18. Taylor, V.F., et al.: There are many apps for that: Quantifying the availability of privacy-preserving apps. In: ACM Conference on Security and Privacy in Wireless and Mobile Networks. pp. 247–252 (2017)
19. Triguero, I., García, S., Herrera, F.: Self-labeled techniques for semi-supervised learning: taxonomy, software and empirical study. Knowledge and Information systems **42**(2), 245–284 (2015)

20. Van Engelen, J.E., Hoos, H.H.: A survey on semi-supervised learning. Machine Learning **109**(2), 373–440 (2020)
21. Van Kleek, M., et al.: Better the devil you know: Exposing the data sharing practices of smartphone apps. In: Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems. pp. 5208–5220 (2017)
22. Wilson, S., et al.: The creation and analysis of a website privacy policy corpus. In: Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics. pp. 1330–1340 (2016)
23. Wu, T., et al.: Catering to your concerns: automatic generation of personalised security-centric descriptions for android apps. ACM Transactions on Cyber-Physical Systems **3**(4), 1–21 (2019)
24. Wu, Z., et al.: Enhancing fidelity of description in android apps with category-based common permissions. IEEE Access **9**, 105493–105505 (2021)
25. Xiao, J., et al.: An android application risk evaluation framework based on minimum permission set identification. Journal of Systems and Software **163** (2020)
26. Zhang, L.L., et al.: Characterizing privacy risks of mobile apps with sensitivity analysis. IEEE Transactions on Mobile Computing **17**(2), 279–292 (2017)

Appendix A: Metrics

We use conventional measures to measure the effectiveness of the proposed learning approaches. In particular, since we have classes with three labels (Y, N, and M), we exploit a 3×3 confusion matrix, see Table 5, where columns represent predicted labels, rows possible actual value and cells denote error value (E) or true positive value (TP). From the confusion matrix, we define the evaluation metrics given in Table 6.

Table 5. Confusion matrix

	Predicted value: Y	**Predicted value: N**	**Predicted value: M**
Actual value: Y	TP_Y	$E_{Y,N}$	$E_{Y,M}$
Actual value: N	$E_{N,Y}$	TP_N	$E_{N,M}$
Actual value: M	$E_{M,Y}$	$E_{M,N}$	TP_M

Table 6. Metrics definition

$Accuracy$	$(TP_Y + TP_N + TP_M)/\#samples$
Pre_X	$TP_X/(TP_X + E_{N,X} + E_{M,X})$
Re_X	$TP_X/(TP_X + E_{X,N} + E_{X,M})$
$F1_X$	$2^*\ (Pre_X * Re_X)/(Pre_X + Re_X)$

where $X \in \{Y, N, M\}$