SimApp: A Framework for Detecting Similar Mobile Applications by Online Kernel Learning

Ning Chen, Steven C. H. Hoi†, Shaohua Li, Xiaokui Xiao
Nanyang Technological University, Singapore, †Singapore Management University, Singapore
{nchen1,lishaohua,xkxiao}@ntu.edu.sg,†chhoi@smu.edu.sg

ABSTRACT

With the popularity of smart phones and mobile devices, the number of mobile applications (a.k.a. "apps") has been growing rapidly. Detecting semantically similar apps from a large pool of apps is a basic and important problem, as it is beneficial for various applications, such as app recommendation, app search, etc. However, there is no systematic and comprehensive work so far that focuses on addressing this problem. In order to fill this gap, in this paper, we explore multi-modal heterogeneous data in app markets (e.g., description text, images, user reviews, etc.), and present "SimApp" – a novel framework for detecting similar apps using machine learning. Specifically, it consists of two stages: (i) a variety of kernel functions are constructed to measure app similarity for each modality of data; and (ii) an online kernel learning algorithm is proposed to learn the optimal combination of similarity functions of multiple modalities. We conduct an extensive set of experiments on a real-world dataset crawled from Google Play to evaluate SimApp, from which the encouraging results demonstrate that SimApp is effective and promising.

Categories and Subject Descriptors

I.2.6 [**Artificial Intelligence**]: Learning; H.4 [**Information Systems Applications**]: Miscellaneous

General Terms

Algorithm and Experimentation

Keywords

Mobile applications; similarity function; multi-modal data; multiple kernels; online kernel learning

1. INTRODUCTION

The proliferation of smart phones in recent years has led to a huge and rapid growth of mobile applications ("apps"). According to a recent report [1], as of June 2014, there were over 1.5 million apps available on Google Play (one of the largest app markets), and the number of apps grew by around 60% between July 2013 and June 2014. With such a large amount of apps, if a specific app is given as a query, it is very difficult to find all other apps that are *similar* to the query app. In this paper, two apps are considered to be *similar* to each other if they implement related semantic (high-level) requirements.

The problem of knowing the semantic similarity between apps is very important because it has many benefits for different stakeholders in the mobile app ecosystem. For example, it can help app platform providers improve the performance of their app recommendation systems and enhance the user experience of app search engines. For app developers, detecting similar apps can be useful for various purposes, such as identifying direct competing apps, assessing reusability (if open source) and many more. The potential application value of this problem motivates our work.

Detecting similar apps is a nontrivial and difficult problem, as our goal is to find apps that share the same high-level concept. In this paper, we try to solve this problem by exploiting multi-modal heterogeneous data in app markets, such as Google Play, Apple App Store, etc. Generally, app markets contain rich contents associated with apps, e.g., description text, screenshot images, user reviews, etc. Such information usually describes conceptual characteristics of apps, and thus is helpful in addressing our problem.

One of the key challenges is how to explore and combine different modalities of data in app markets to measure the similarity between apps in a principled way. Previous studies [24, 2] provided simple solutions in their app recommendation systems, which were based on app description, title and user reviews. However, these methods are far from comprehensive and systematic, since other kinds of rich metadata have not been well exploited. To fill this gap, in this paper, we present a novel framework named "SimApp" for modeling app similarity by leveraging multi-modal heterogeneous data in app markets via an online kernel learning approach. Figure 1 depicts the overview of the SimApp framework. First of all, we measure the pairwise app similarity by defining a variety of kernel (similarity) functions on different modalities. Second, we assume the target app similarity function is a linear combination of the multiple kernels, and then employ online learning techniques [17] to learn the optimal combination weights from streams of training data. Finally, the learned app similarity function can facilitate a number of applications. In particular, we conduct an extensive set of experiments to evaluate the efficacy of SimApp based on a

WSDM'15, February 2–6, 2015, Shanghai, China.

http://dx.doi.org/10.1145/2684822.2685305.

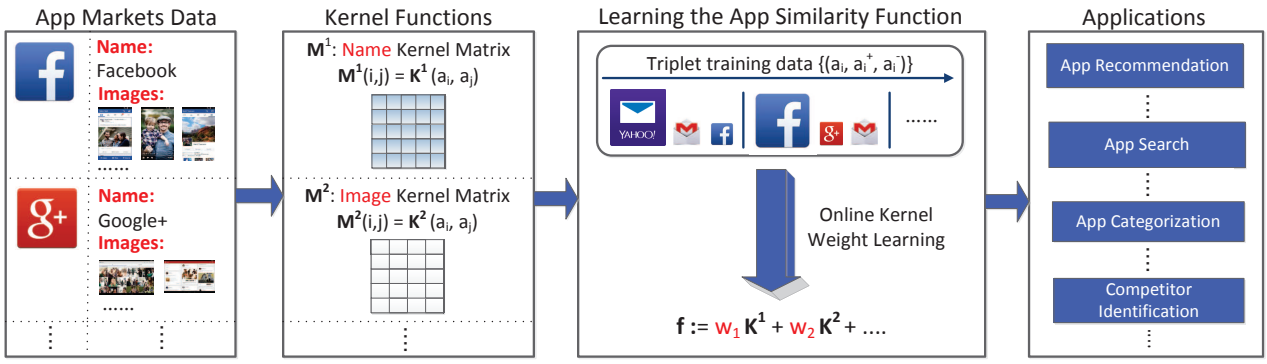

Figure 1: Overview of our proposed SimApp framework. The first block shows the data associated with apps. Only two modalities, i.e., *Name* and *Images*, are presented for illustration. In the second block, we define two kernel functions K^1 and K^2 to measure the pairwise app similarity in modalities of *Name* and *Images*, respectively. In the third block, we learn the optimal combination weights (w_1, w_2, ...) by using our proposed Online Kernel Weight Learning (OKWL) algorithm. The learned app similarity function f can be applied to a variety of applications presented in the last block.

similar app recommendation task. Preliminary empirical results have validated the effectiveness of SimApp in modeling app similarity. Supplementary materials (including datasets, code, etc.) are publicly available at[1].

In summary, this paper makes the following contributions:

- We study the problem of modeling high-level mobile app similarity. To the best of our knowledge, this is the first systematic and comprehensive work that focuses on this problem;
- We present SimApp as a novel framework to tackle this problem, which fuses multi-modal data in app markets by learning the optimal combination weights from streams of training data;
- We conduct an extensive set of experiments to evaluate the performance of SimApp on a real-world dataset.

The rest of the paper is organized as follows: Section 2 discusses related work; Section 3 gives the problem formulation; Section 4 presents SimApp; Section 5 shows the empirical results; finally Section 6 concludes this paper.

2. RELATED WORK

Our work relates to various studies in multiple important research areas, ranging from data mining, machine learning, software engineering to security. In this section, we group related studies into four categories, and survey the literature of each category in detail.

2.1 High-level Software Similarity

Our work **closely** relates to studies on detecting semantically similar software applications. To our best knowledge, there are two lines of related work in this direction.

In [26], Zhu et al. explored the mobile app classification problem, however we aim to address a different problem, i.e., deriving a function to measure the high-level similarity between apps. Thus far, little work has been done on this problem. Two recent studies, which focused on app recommendation, presented simple methods to measure the similarity between apps by using meta-information in app markets [24, 2]. Specifically, Yin et al. [24] treated the description of an app as a document and applied LDA [3] to learn its latent topic distribution. In this way, each app is represented as a fixed length vector. Then, the similarity between two apps is computed as the cosine similarity of their vectors. Similarly, Bhandari et al. [2] linked the title, description and user reviews of an app as one document, and then built the vector using the *tf-idf* weighting scheme. They also used cosine similarity to calculate the pairwise similarity. In contrast, we exploit more types of metadata (e.g., images, numerics, etc.) in app markets, and propose a principled way to combine such multi-modal heterogeneous data to measure the high-level similarity between apps.

Another line of related work is to categorize or detect semantically similar traditional software applications by using low-level data, e.g., source code, byte code, etc [19, 15]. For example, McMillan et al. [15] proposed an approach called CLAN to detect similar Java applications using API calls. Compared with this line of work, our work differs in that we focus on using high-level (not implementation-level) data in app markets to measure the similarity between apps, and propose a very different solution.

2.2 Low-level Software Similarity

Our work is also related to studies on modeling low-level software similarity, e.g., code clone detection [16]. Fragments of code (low-level implementation) are identified as code clones if they are exactly the same as or similar to each other [16]. Many techniques have been developed to detect code clones for traditional software applications [10, 12]. For example, Liu et al. [12] proposed a tool named G-PLAG, which applies subgraph isomorphism comparison to Program Dependence Graphs (PDGs) to detect plagiarized (similar) code. Recently, there are some studies focusing on detecting mobile app clones via analyzing byte codes or opcodes [25, 5]. For example, Zhou et al. [25] presented an app similarity measurement system named *DroidMOSS* based on fuzzy hashing technique to detect app clones.

In general, our work differs from works in this area in two aspects. First and foremost, the goal of our problem is different. Our problem aims to find apps that implement similar semantic requirements. Whether two apps are similar or not is determined by the high-level functionalities they perform rather than their low-level implementations. Second, to

[1]https://sites.google.com/site/appsimilarity/

achieve our goal, we explore a very different data source, i.e., app markets, and present a new framework using very different techniques.

2.3 Data Mining with App Markets Data

Our work is in general related to the emerging studies on mining app markets data to facilitate various applications [11, 7, 27, 6, 20]. For example, Lin et al. [11] proposed a framework to incorporate version histories into app recommendation systems. Zhu et al. [27] presented a ranking fraud detection system for apps, which is based on mining apps' historical ranking and rating records.

Although the nature of data studied in the above works is similar to ours, the techniques used and research goals are totally different. Our work aims to use the knowledge discovered from app markets data to detect similar apps.

2.4 Online Learning

In this paper, we explore online learning techniques [9, 17] to learn the best combination weights of various modalities from streams of training data in the form of triplets.

Compared with batch learning, online learning has several advantages, e.g., fast and simple, thus making it applicable to applications in a variety of domains [14, 4, 22, 23, 21]. For example, Chechik el al. [4] proposed an online learning algorithm named OASIS to incrementally learn a better distance metric from image data for retrieval. Compared with the above studies, our work differs in two points. First, we formulate and solve a challenging problem in a completely different domain with its distinct features. Second, the goal of our proposed technique is different, i.e., learning a kernel function from multi-modal heterogeneous data.

Our work is also related to recent work on Online Multiple Kernel Learning (OMKL) [8]. In general, our work differs from studies in this area in two aspects. First, we focus on a more specific problem, i.e., detecting similar apps, and present the SimApp framework to solve it. Second, the OMKL technique aims to learn the optimal weights of multiple kernels for classification tasks, while the goal of our proposed online learning algorithm in SimApp is to learn the app similarity function from streams of triplets.

3. PROBLEM FORMULATION

In this section, we formally formulate the problem of mobile app similarity modeling using app markets data.

DEFINITION 1 (MOBILE APP SIMILARITY MODELING). Given a collection of mobile apps $\mathcal{A}$, the objective of mobile app similarity modeling problem is to learn a function $f : \mathcal{A} \times \mathcal{A} \rightarrow \mathbb{R}_+$, such that $f(a_i, a_j)$ measures the semantic similarity between app a_i and app a_j.

In our problem, two apps are considered to be similar to each other if they implement related semantic requirements. An intuitive example is shown in Figure 2 to better understand our problem. Figure 2 presents the names and logos of four popular apps. Obviously, "Facebook" and "Google+" are both social networking apps, so they are considered to be similar to each other in our problem. In the same way, "Dropbox" is similar to "Google Drive" as both apps are used to store and share files. "Facebook" is not similar to "Dropbox" as they perform very different functionalities. The goal of our problem is to learn a function f, which assigns higher scores to pairs of more similar apps, e.g., f ("Facebook", "Google+") $>$ f ("Facebook", "Dropbox").

Figure 2: Four popular mobile apps.

In Definition 1, a key element is a mobile app $a_i \in \mathcal{A}$, which is defined as follows.

DEFINITION 2 (MOBILE APP). Each mobile app $a_i \in \mathcal{A}$ is modeled as a k-dimensional tuple $a_i = [m_{i1}, m_{i2}, ..., m_{ik}]$, where each $m_{ij} (1 \leq j \leq k)$ is a modality (attribute) of mobile app a_i.

In this work, we explore the multi-modal heterogeneous data in app markets to model apps. Specifically, we use the modalities shown in Table 1 to represent an app, since they are supported by most mainstream app markets. The "Modality" column of Table 1 shows the names of the modalities, and the "Description" column briefly describes these modalities. Figure 3 shows an example of the multi-modal information associated with the "Facebook" app on Google Play. From Table 1 and Figure 3, we can see that, these multi-modal data in general describe high-level characteristics of apps, and thus is suitable for solving the problem presented in Definition 1.

Table 1: The modality set of mobile apps.

ID	Modality	Description
1	Name	The title of the app.
2	Category	The category label of the app.
3	Developer	The developer of the app.
4	Description	The description text of the app.
5	Update	The latest changes to the app.
6	Permissions	The permissions required by the app.
7	Images	The screenshots of the app.
8	Content Rating	The content level of the app.
9	Size	The storage space needed for the app.
10	Reviews	The comments posted by users.

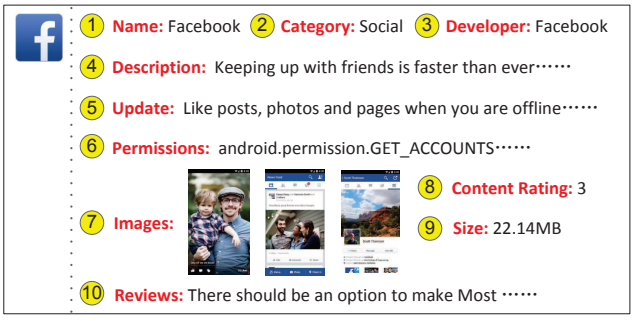

Figure 3: Example of the multi-modal information associated with the "Facebook" app on Google Play. The circled numbers correspond to the IDs shown in Table 1.

4. APP SIMILARITY MODELING

In this section, we propose the SimApp framework for mobile app similarity modeling. As shown in Figure 1, SimApp consists of two stages. First, we define a series of 10 kernel (similarity) functions in different modalities (as shown in Table 1) for measuring the similarity between apps (see Section 4.1). Second, we assume the target app similarity function f is a linear combination of the multiple kernels, and present a novel Online Kernel Weight Learning (OKWL) algorithm to learn the optimal combination weights of these 10 kernels from streams of training triplets (see Section 4.2).

4.1 Kernels for Measuring App Similarity

In machine learning, kernel is essentially a mapping function that transforms a given low-dimensional space into some higher-dimensional (possibly infinite) space. A kernel function can be thought of as a pairwise similarity function. In this subsection, we build a variety of kernel functions, denoted as $K^k (1 \leq k \leq 10)$, to measure app similarity for each modality shown in Table 1. Without loss of generality, all kernel values are within the range of $[0, 1]$.

4.1.1 App Similarity using Name

Each mobile app has a *name* (title), which is given by its developer. Since app name is the first thing potential users see when they browse the app markets, it is often descriptive, explicit, and stating what the app can do for users. When two app names share many common words, it is reasonable to infer that these two apps perform similar functionalities. As app name is essentially a short string of characters, we employ the well-known string kernel [13] to measure the app similarity in this modality. Let s_i and s_j denote the names of app a_i and app a_j, respectively. Then, we have,

$$K^1(a_i, a_j) = \sum_{u \in \Sigma^*} \phi_u(s_i)\phi_u(s_j)$$

where Σ^* denotes the set of all subsequences, u denotes a subsequence, and ϕ is a feature mapping function. For space limitation, please refer to [13] for more details about string kernel. In particular, we set the max length of subsequences equal to 3, and the decay factor equal to 0.5.

4.1.2 App Similarity using Category

To make it more easier for users to browse and discover apps, app markets usually define a list of *categories* to organize apps. In general, apps belong to the same category are more related to each other than apps in other categories. Therefore, we can use such category information to measure the similarity between apps. Let c_i and c_j denote the category labels of app a_i and app a_j, respectively, thus,

$$K^2(a_i, a_j) = \begin{cases} \alpha & \text{if} \quad c_i == c_j \\ 0 & \text{if} \quad c_i \neq c_j \end{cases}$$

where α is a pre-defined parameter. For simplicity, we set α equal to 1.0 in this work.

4.1.3 App Similarity using Developer

Every app is created by one *developer*, who is either an individual or a company. A developer usually maintains a list of apps that belong to different categories in the app market. We collect all the category labels in the app market and build a category dictionary with size d_c. Then, the developer of an app can be converted into a feature vector $\mathbf{d} \in \mathbb{R}^{d_c}$ using the *tf-idf* weighting scheme. Here, *tf* is the number of the developer's apps that belong to a category, and *idf* is the total number of apps in that category. Let $\mathbf{d}_i$ and $\mathbf{d}_j$ represent the developers of app a_i and app a_j, respectively. We measure the app similarity in this modality by using the RBF (radial basis function) kernel, which is usually a reasonable first choice,

$$K^3(a_i, a_j) = exp\left(-\frac{\|\mathbf{d}_i - \mathbf{d}_j\|^2}{2\sigma^2}\right)$$

where σ is the bandwidth parameter. If not specified, we set σ as the average Euclidean distance in this work.

4.1.4 App Similarity using Description

The *description* text of an app usually describes its main functionalities provided for users. If two app descriptions are related to each other, probably there are some function overlaps between these two apps. We collect all the descriptions and treat them as documents. Then, we apply LDA [3] to learn the latent topic distribution for each of them. In this way, the description of an app is represented as a fixed length vector $\mathbf{t} \in \mathbb{R}^{d_t}$, where d_t is the number of discovered topics (we set $d_t = 1000$). The i-th dimension of this vector is the distribution over the i-th topic. Let $\mathbf{t}_i$ and $\mathbf{t}_j$ denote the descriptions of app a_i and app a_j, respectively. We use the normalized liner kernel, which is widely used for text classification, for measuring the app similarity in this modality, formally,

$$K^4(a_i, a_j) = \frac{\mathbf{t}_i^T \mathbf{t}_j}{\|\mathbf{t}_i\| \, \|\mathbf{t}_j\|}$$

Remark. K^4 is exactly the same as the method used in [24] to measure the app similarity. Our purpose in doing so is to make more fair comparisons in our experiments (see Section 5.4). Without loss of generality, the scheme used in K^4 is also employed for other text data in this work.

4.1.5 App Similarity using Update

The *update* text is provided by developers to keep users informed about changes made to the latest version of apps. Following the same scheme used for *description* text, each update of an app is converted into a fixed length vector $\mathbf{u} \in \mathbb{R}^{d_u}$, where d_u is the number of latent topics (we set $d_u = 1000$). Let $\mathbf{u}_i$ and $\mathbf{u}_j$ denote the updates of app a_i and app a_j, respectively. Then, we measure the app similarity in this modality by using the normalized liner kernel,

$$K^5(a_i, a_j) = \frac{\mathbf{u}_i^T \mathbf{u}_j}{\|\mathbf{u}_i\| \, \|\mathbf{u}_j\|}$$

4.1.6 App Similarity using Permissions

Each app has a list of *permissions* that it needs to access specific functionalities or information on users' smart phones. To represent the permissions of an app, we use the well-known bag-of-word (BoW) model. First, we collect all the permissions and compile a permission dictionary with size d_p. Then, the permissions of an app are transformed into a feature vector in $\mathbb{R}^{d_p}$ using the *tf-idf* weighting scheme. In such a way, an app a_i can be represented by a feature vector $\mathbf{p}_i \in \mathbb{R}^{d_p}$. We also use the RBF kernel to measure the app similarity in the permission space, formally,

$$K^6(a_i, a_j) = exp\left(-\frac{\|\mathbf{p}_i - \mathbf{p}_j\|^2}{2\sigma^2}\right)$$

4.1.7 App Similarity using Images

In app markets, app developers can upload some screenshot images to show their apps' features and functionalities. We use the bag-of-(visual) word model for visual feature representation. Specifically, we first compute the SIFT descriptors for each screenshot image. Then, we apply the K-means algorithm over all the SIFT descriptors, and obtain d_i (we set $d_i = 2000$) clusters as the visual words. Finally, each given image is represented as a histogram of visual words, i.e., a fixed length feature vector $\mathbf{i} \in \mathbb{R}^{d_i}$. Since an app a usually has more than one screenshot image, we use the centroid of all images belong to app a to represent it in the visual space, formally,

$$\overline{\mathbf{a}} = \sum_m \frac{\mathbf{i}_m}{|a|}$$

where $\mathbf{i}_m$ denotes one image of app a, and $|a|$ is the total number of images that belong to app a. We measure the visual similarity between two apps a_i and a_j using the RBF kernel, formally,

$$K^7(a_i, a_j) = exp\left(-\frac{\|\overline{\mathbf{a}}_i - \overline{\mathbf{a}}_j\|^2}{2\sigma^2}\right)$$

4.1.8 App Similarity using Content Rating

The *content rating* of an app provides users concise and impartial information about the content and age appropriateness of this app. Let cr_i and cr_j denote the content ratings of app a_i and app a_j, respectively. The similarity between a_i and a_j in this modality is given by,

$$K^8(a_i, a_j) = \begin{cases} \beta^{|cr_i - cr_j|} & \text{if} \quad |cr_i - cr_j| < cr_{max} - cr_{min} \\ 0 & \text{if} \quad |cr_i - cr_j| == cr_{max} - cr_{min} \end{cases}$$

where cr_{max} and cr_{min} are the maximum rating and minimum rating, respectively, in the target app market content rating system. β is a decay paratemer which we set equal to 1/3 in this work.

4.1.9 App Similarity using Size

The *size* information indicates the storage space required by the apps. Two apps have large difference in size tend to be not similar to each other. Let s_i and s_j denote the size of app a_i and app a_j, respectively. We measure the similarity between a_i and a_j in this modality as follows,

$$K^9(a_i, a_j) = exp\left(-\frac{|s_i - s_j|}{\gamma}\right)$$

where we set γ as the median size (unit: MB) of all apps in our data collection.

4.1.10 App Similarity using Reviews

User review is a crucial component of app markets. Each app usually has a list of reviews posted by different users. These user reviews contain rich information on various aspects of apps (such as functionality, quality, performance, etc.), thus could be useful for measuring app similarity. To represent the reviews of an app, we concatenate all the reviews of one app together as a document. Then, following the same scheme employed for *description* and *update* text, for each app a_i, we denote $\mathbf{r}_i \in \mathbb{R}^{d_r}$ as its review topics distribution, where d_r is the number of discovered topics (we set $d_r = 1000$). Finally, we employ the normalized liner kernel to quantify the app similarity in this modality,

$$K^{10}(a_i, a_j) = \frac{\mathbf{r}_i^T \mathbf{r}_j}{\|\mathbf{r}_i\| \, \|\mathbf{r}_j\|}$$

4.2 Learning Optimal Weights for Kernels

In the previous subsection, we introduce various kernel functions ($K^1 \sim K^{10}$) to measure the similarity between apps in different modalities. The next challenge is how to find the best way to combine these kernels. In this work, we assume the target app similarity function f is a linear combination of the multiple kernels, i.e.,

$$K(a_i, a_j; \mathbf{w}) = \sum_{k=1}^{n} w_k K^k(a_i, a_j) \tag{1}$$

where a_i and a_j are the i-th and j-th app, respectively. $K(a_i, a_j; \mathbf{w})$ represents the target app similarity function f. K^k is the kernel defined under the k-th view (modality) of apps, n is the total number of kernels, and $\mathbf{w} \in \mathbb{R}^n$ is the weight vector with each element w_k represents the weight of the k-th kernel.

One simple way is to let humans assign the weights of different kernels. However, such strategy depends too much on domain knowledge and often cannot find the best combination. Therefore, in this work, we explore online learning techniques to learn the optimum combination weights $\mathbf{w}$ from streams of triplets. The reasons we choose the online learning scheme are that (i) it can avoid the expensive retraining cost when new training data arrives; and (ii) it is highly efficient and scalable for large scale applications.

4.2.1 Relative Similarity Learning

In the training phase, we assume a collection of training instances is given sequentially in the form of triplet [4], i.e.,

$$\mathcal{T} = \{(a_i, a_i^+, a_i^-), i = 1, ..., m\}$$

where each triplet (a_i, a_i^+, a_i^-) indicates that app a_i is more semantically similar to app a_i^+ than a_i^-. Here m is the total number of triplets in the training data. We aim to learn the target app similarity function $K(a_i, a_j; \mathbf{w})$ in Equation (1), which assigns higher similarity scores to pairs of more relevant apps, i.e.,

$$K(a_i, a_i^+) > K(a_i, a_i^-), (a_i, a_i^+, a_i^-); \forall i$$

In such a triplet setting, we only need to give the relative order of similarity rather than an exact measure of similarity, thus is more feasible in practice. Such triplet instances can be generated in practice as follows. Let $\mathcal{A}$ denote a set of training apps. In the 1st step, for each app $a_i \in \mathcal{A}$, we need to create a list of apps that are relevant to a_i, denoted as $L(a_i)$. This can be achieved through various ways. For example, we can use existing app search engines to find apps that share the same query. In the 2nd step, in order to build a training triplet (a_i, a_i^+, a_i^-), we can first uniformly sample an app a_i from $\mathcal{A}$; then uniformly sample an app a_i^+ from $L(a_i)$; and finally uniformly sample an app a_i^- from $\mathcal{A} - L(a_i)$.

4.2.2 Online Kernel Weight Learning Algorithm

We propose an online kernel learning algorithm which is an application of the stochastic sub-gradient method [18]. The algorithm learns from side information in the form of

triplet as described above. Our goal is to learn a similarity function $K(a_i, a_j; \mathbf{w})$, for all triplets in $\mathcal{T}$ satisfying,

$$K(a_i, a_i^+) > K(a_i, a_i^-) + \epsilon$$

where ϵ is a margin factor (should be positive) which is set equal to 1.0 in our experiments. For each triplet (a_i, a_i^+, a_i^-), we define the following hinge loss,

$$\begin{aligned} l(a_i, a_i^+, a_i^-) &= max\{0, \epsilon - K(a_i, a_i^+) + K(a_i, a_i^-)\} \\ &= max\{0, \epsilon - \mathbf{w} \cdot \mathbf{s}_i^+ + \mathbf{w} \cdot \mathbf{s}_i^-\} \end{aligned}$$

where $\mathbf{s}_i^+ = [K^1(a_i, a_i^+), ..., K^k(a_i, a_i^+), ..., K^n(a_i, a_i^+)]^T$ and $\mathbf{s}_i^- = [K^1(a_i, a_i^-), ..., K^k(a_i, a_i^-), ..., K^n(a_i, a_i^-)]^T$.

Our goal is to find the minimizer of the object function,

$$\min_{\mathbf{w}} \frac{\lambda}{2} \|\mathbf{w}\|^2 + \frac{1}{m} \sum_{i=1}^{m} l(a_i, a_i^+, a_i^-) \quad (2)$$

where λ is a regulization parameter.

Algorithm 1: Online Kernel Weight Learning

Input: $\mathcal{T}$, λ, T, η_0
1 Initialize weights: $w_{1,k} = 1/n, \forall k = 1, ..., n$
2 **for** $t = 1, 2, ..., T$ **do**
3 Set $\eta_t = \eta_0/(1 + \lambda\eta_0 t)$.
4 Receive one triplet $(a_{i_t}, a_{i_t}^+, a_{i_t}^-)$ from $\mathcal{T}$.
5 Compute $\mathbf{s}_{i_t}^+$ and $\mathbf{s}_{i_t}^-$.
6 **if** $\boldsymbol{w}_t \cdot \boldsymbol{s}_{i_t}^+ - \boldsymbol{w}_t \cdot \boldsymbol{s}_{i_t}^- \geqslant \epsilon$ **then**
7 update $\mathbf{w}_{t+1} \leftarrow (1 - \eta_t\lambda)\mathbf{w}_t$
8 **end**
9 **else if** $\boldsymbol{w}_t \cdot \boldsymbol{s}_{i_t}^+ - \boldsymbol{w}_t \cdot \boldsymbol{s}_{i_t}^- < \epsilon$ **then**
10 update $\mathbf{w}_{t+1} \leftarrow (1 - \eta_t\lambda)\mathbf{w}_t + \eta_t(\mathbf{s}_{i_t}^+ - \mathbf{s}_{i_t}^-)$
11 **end**
12 **end**
Output: $\mathbf{w}_{T+1}$

Next, we describe the core procedure of our proposed Online Kernel Weight Learning (OKWL) algorithm for solving the optimization problem given in Equation (2) in detail.

Algorithm 1 shows the pseudo-code of the OKWL algorithm. The inputs include (i) a set of training triplets $\mathcal{T}$; (ii) a regulization parameter λ; (iii) a learning rate constant η_0; and (iv) the number of iterations T. Among them, the proper η_0 can be determined experimentally by using a sample of training triplets. Initially, we set $\mathbf{w}_1 = [w_{1,1}, ..., w_{1,k}, ..., w_{1,n}]^T$, where $w_{1,k} = 1/n, \forall k = 1, ..., n$, n is the number of kernels, so each kernel is assigned the same weight (Line 1). For each training iteration t, we first set the learning rate $\eta_t = \eta_0/(1 + \lambda\eta_0 t)$ (Line 3). Then, for a triplet $(a_{i_t}, a_{i_t}^+, a_{i_t}^-)$ received from $\mathcal{T}$ (Line 4), we compute $\mathbf{s}_{i_t}^+$ and $\mathbf{s}_{i_t}^-$, respectively (Line 5). The objective function based on this triplet is:

$$\mathcal{L}(\mathbf{w}; i_t) = \frac{\lambda}{2} \|\mathbf{w}\|^2 + l(\mathbf{w}; (a_{i_t}, a_{i_t}^+, a_{i_t}^-)) \quad (3)$$

Then, the sub-gradient of $\mathcal{L}(\mathbf{w}; i_t)$ with respect to $\mathbf{w}$ is given by:

$$\nabla_t = \frac{\partial \mathcal{L}(\mathbf{w}; i_t)}{\partial \mathbf{w}} = \begin{cases} \lambda \mathbf{w}_t & \text{if} \quad \mathbf{w}_t \cdot (\mathbf{s}_{i_t}^+ - \mathbf{s}_{i_t}^-) \geqslant \epsilon \\ \lambda \mathbf{w}_t + \mathbf{s}_{i_t}^- - \mathbf{s}_{i_t}^+ & \text{if} \quad \mathbf{w}_t \cdot (\mathbf{s}_{i_t}^+ - \mathbf{s}_{i_t}^-) < \epsilon \end{cases}$$

We then update $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \eta_t \nabla_t$ (Line 6-11), formally:

$$\mathbf{w}_{t+1} = \begin{cases} (1 - \eta_t\lambda)\mathbf{w}_t & \text{if} \quad \mathbf{w}_t \cdot (\mathbf{s}_{i_t}^+ - \mathbf{s}_{i_t}^-) \geqslant \epsilon \\ (1 - \eta_t\lambda)\mathbf{w}_t + \eta_t(\mathbf{s}_{i_t}^+ - \mathbf{s}_{i_t}^-) & \text{if} \quad \mathbf{w}_t \cdot (\mathbf{s}_{i_t}^+ - \mathbf{s}_{i_t}^-) < \epsilon \end{cases}$$

Finally, after the predefined T iterations, we output the learned weight vector $\mathbf{w}_{T+1}$ as the optimal combination of different kernels.

5. EMPIRICAL EVALUATION

In this section, we report the results of an extensive set of experiments based on a similar app recommendation task.

5.1 Dataset

Our empirical evaluation is based on a real-world dataset crawled from Google Play. For each app, we collected all the data (associated with it) available on Google Play, including app name, description, screenshots, user reviews, etc. Note that, each kernel function defined in Section 4.1 corresponds to a specific kind of data we have collected. This yielded a dataset that consists of 21,624 apps from 42 different categories. We think such a scale is enough to work with to draw solid conclusions. The "Total" column of Table 2 shows some statistics of the whole dataset. To facilitate our empirical studies, we further split the whole dataset into a training set and a test set (see "Training" and "Test" columns in Table 2). Specifically, from each of the 42 categories, we randomly choose about 80% of the apps in the category as the training data and the remaining as the test data.

Table 2: Some statistics of the Google Play dataset. The notation "#" represents the number of some object.

Set	Training	Test	Total
# Apps	16,955	4,669	21,624
# Permissions	153,639	42,869	196,508
# Reviews	16,872,290	5,037,246	21,909,536
# Images	136,792	37,601	174,393

5.2 Evaluation Measures

We introduce the evaluation metrics used in our empirical studies. For each query app, all other apps are ranked according to their similarities to the query app. Thus, we adopt two rank-based metrics, i.e., Precision@K and mean Average Precision (mAP), which are presented below,

- **Precision@K:** For each query app, we compute the proportion of similar apps in the top-K results. When averaged across all query apps, this yields the Precision@K measure.

- **mean Average Precision:** mAP for a set of app queries is the mean of the average precision scores for each app query, i.e.,

$$mAP = \frac{\sum_{q=1}^{Q} AveP(q)}{Q}$$

where Q is the number of app queries, and $AveP(q)$ is the average precision for an app query q. Average precision for each app query is the average value of all the precision values from the rank positions that have a similar app.

Table 3: The base kernels and the optimal weights learned by OKWL ($\eta_0 = 0.01, \lambda = 10^{-4}, T = 100K$). The largest weight is in blue color, and the lowest weight is in red color.

Kernel	K^1	K^2	K^3	K^4	K^5	K^6	K^7	K^8	K^9	K^{10}
Modality	Name	Category	Developer	Description	Update	Permissions	Images	Content Rating	Size	Reviews
Weight	0.1679	0.1676	0.0827	0.1288	0.0455	0.0027	0.0960	0.0394	0.0353	0.2338

5.3 Experimental Setup

5.3.1 Build an App-App Relevance Matrix

To quantitatively evaluate SimApp, we need to create an app-app relevance matrix $\mathbf{R}_{AA}$ as our ground truth labels. Google Play has a *"Similar"* functionality, which recommends users a list of similar apps for each app. We collected a set of m such lists $L = \{l_1, l_2, ..., l_m\}$ from the web portal of Google Play, where $l_k (1 \leq k \leq m)$ is the k-th list. Given two apps a_i and a_j, let $freq(a_i, a_j)$ denote the number of lists they both appear in. We consider a_i and a_j are similar to each other if and only if $freq(a_i, a_j)$ exceeds a threshold θ (which is used to reduce noise), formally,

$$\mathbf{R}_{AA}(a_i, a_j) = \begin{cases} 1 & if \quad freq(a_i, a_j) \geq \theta \\ 0 & if \quad freq(a_i, a_j) < \theta \end{cases}$$

For the training set and the test set shown in Table 2, we build two matrices $\mathbf{R}_{AA}^{train}$ and $\mathbf{R}_{AA}^{test}$, respectively, where we use the threshold $\theta = 2$.

5.3.2 Training Triplets Sampling

We use $\mathbf{R}_{AA}^{train}$ to sample training triplets $\mathcal{T}' = \{(a_i, a_i^+, a_i^-), i = 1, ..., m\}$ as follows. Let $\mathcal{A}$ be all the apps in the training set. First of all, we randomly sample an app a_i from $\mathcal{A}$. Then we uniformly sample an app a_i^+ from the set of apps which are similar to a_i ($\mathbf{R}_{AA}^{train}(a_i, a_i^+) = 1$). Finally, we uniformly sample an app a_i^- from the set of apps which are not similar to a_i ($\mathbf{R}_{AA}^{train}(a_i, a_i^-) = 0$). In such a way, we generate a set of 100K training triplets $\mathcal{T}'$ which is used to train OKWL in our experiments.

5.3.3 Choose the Learning Rate Constant

The learning rate constant η_0 is a key parameter of the OKWL algorithm, which is determined experimentally as follows. Given the training triplets $\mathcal{T}'$, we fix $\lambda = 10^{-4}$, $T = 30K$, and then run OKWL with different values of $\eta_0 \in [0.0001, 0.1]$. Figure 4 traces the training error over the training triplets as it progresses during learning. From the results shown in Figure 4, we can see that, larger values of η_0 (i.e., 0.1 and 0.01) are more attractive, because OKWL (i) achieves better asymptotic performance; and (ii) converges much faster. Between 0.1 and 0.01, to make OKWL more robust, we choose $\eta_0 = 0.01$ as its training error curve shown in Figure 4 is more smooth.

5.3.4 Compared Methods

We compared the following methods in our experiments:

- Single: A single kernel defined in Section 4.1. We examine $K^1 \sim K^{10}$ one by one.
- Uniform: $K^1 \sim K^{10}$ are uniformly combined with each kernel assigned the same weight.
- SimApp: We use the proposed OKWL algorithm for combining $K^1 \sim K^{10}$. Specifically, we run OKWL with $\eta_0 = 0.01, \lambda = 10^{-4}$ and $T = 100K$, the learned weights of kernels $K^1 \sim K^{10}$ are presented in Table 3.

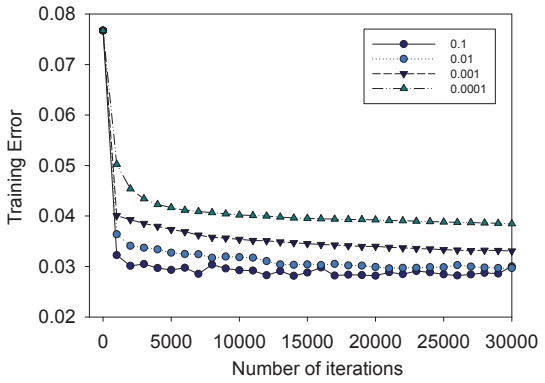

Figure 4: Training error of OKWL as a function of the number of iterations. $\eta_0 \in \{0.1, 0.01, 0.001, 0.0001\}$.

5.4 Quantitative Results

We evaluate the efficacy of SimApp by measuring the similar app recommendation accuracy. Specifically, for each query app in the test set shown in Table 2, we rank all other test apps according to their similarities to the query app, and extract top ones as recommended apps.

In the **first** experiment, we evaluate and compare all the methods listed in Section 5.3.4 in terms of Precision@K ($1 \leq K \leq 5$). We use $\mathbf{R}_{AA}^{test}$ (described in Section 5.3.1) as the ground truth. Since the largest K value in this experiment is 5, we only use test apps that have at least 5 similar apps as query apps, thus resulting in a total of 1910 query apps. Figure 5 shows the top-K app recommendation results, from which we can draw some observations.

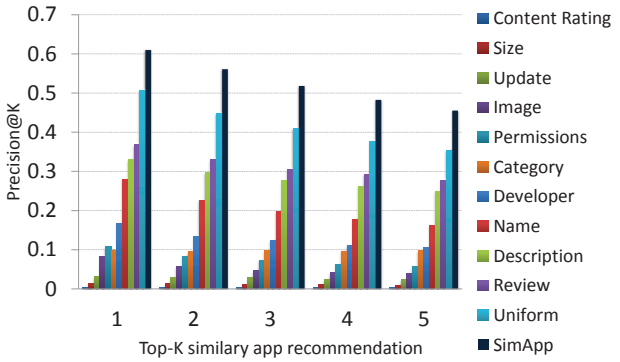

Figure 5: The average precision of top-K similar app recommendation (better viewed in color).

First of all, with K varying from 1 to 5, SimApp always achieves the best performance among all the evaluated methods. By looking into the results, we find that (i) compared with *Uniform* combination, SimApp improves the precision scores by more than 20%; and (ii) the top-1 and top-5 precision scores achieved by SimApp are 0.610 and 0.455, respectively. Such fair results indicate that SimApp is able to model the app similarity.

Second, the *Uniform* combination performs better than all the single kernels ($K^1 \sim K^{10}$), and reports the second best results. This fact indicates that the idea of combining different modalities is helpful in measuring app similarity. However, such kind of naive combination cannot yield the best results, thus learning the weights of different modalities in an effective way is needed.

Third, the *Reviews* kernel (K^{10}) reports the best performance among all the single kernels. Its relatively good results indicate that user review is the most informative modality when measuring app similarity. Two possible reasons are: (i) users often discuss functionalities of apps in the reviews; (ii) users tend to compare apps with their similar competitors in their reviews.

Fourth, the *Description* kernel (K^4) attains the second best results among all the single kernels. Note that, K^4 is used in [24] as a part of their app recommendation system. From Figure 5, we can see that, SimApp performs much better than K^4. For example, SimApp achieves 0.610 while K^4 only achieves 0.331 in terms of Precision@1. This fact indicates that SimApp can improve the overall performance of the app recommendation system proposed in [24].

Finally, the *Content Rating* kernel (K^8) and the *Size* kernel (K^9) report the worst performance among the 10 single kernels. For K from 1 to 5, their precision scores are most zero consistently. Such results indicate that *Content Rating* and *Size* are the least useful modalities.

In the **second** experiment, we evaluate and compare all the methods listed in Section 5.3.4 in terms of mAP. We also use $\mathbf{R}_{AA}^{test}$ as the ground truth and all the apps (that have at least one similar app) in the test set shown in Table 2 as query apps. The mAP results are presented in Figure 6, from which we can draw two observations.

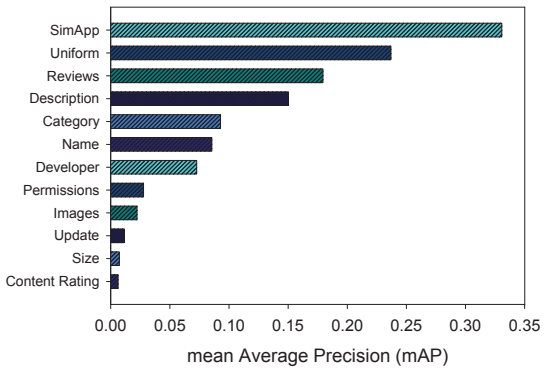

Figure 6: Evaluation of the mAP performance.

First of all, SimApp again performs the best among all the evaluated methods, which further validates its efficacy in app similarity modeling.

Second, the mAP results attained by all the methods are not high, e.g., 0.331 achieved by SimApp. This could be the result of several reasons. Most importantly, our labels that measure the pairwise app similarity (i.e., $\mathbf{R}_{AA}^{test}$) may be partial. This means that some pairs of apps that are similar to each other are not labeled as such; while some pairs of apps that are not similar to each other are labeled as such. The partial labels would lead to an underestimate of the performance of SimApp (in terms of both mAP and Precision@K). To obtain a more accurate estimate of the real power of SimApp, we conduct a human evaluation experiment which is presented in the next subsection.

5.5 Human Evaluation Experiment

5.5.1 Setup

We choose a subset of test apps as query apps used in this experiment as follows. For each test app in the test set, we retrieve its top-10 similar apps as determined by Google Play's (web portal) *"Similar"* functionality ("Google Play" in short). If all these 10 similar apps exist in our whole dataset shown in Table 2, we select this test app as a candidate query app. In such a way, we get a set of candidate query apps covering 32 different categories. Then, for each of the 32 categories, we randomly select one query app from the candidates, thus obtaining a total number of 32 query apps.

For each query app, apart from the top-10 similar apps as ranked by Google Play (verified on July 9th, 2014), we also retrieved its top-10 similar apps (from the whole dataset) as determined by SimApp (using the weights shown in Table 3). All 32 query apps were presented to two human annotators (the first and the third authors), asking them to label which of the 20 retrieved apps are semantically similar to the query app. The two annotators made the first round of labeling independently, then had a discussion on those inconsistent cases, and finally reached an agreement. We collected the final labeling results and calculated Precision@K values.

5.5.2 Results

Table 4 shows the average precisions across all 32 query apps. From the results shown in Table 4, we can draw the following observations. First, SimApp consistently achieves better results than Google Play throughout the full range of K evaluated. Second, SimApp attains 0.875 and 0.819 in terms of Precision@1 and Precision@5, respectively, which are much higher than the values calculated using $\mathbf{R}_{AA}^{test}$ as shown in Figure 5. Such good results indicate that SimApp is effective in modeling app similarity, and potentially valuable for stakeholders in the mobile app ecosystem (even for those who have built some app similarity functions).

Table 4: Precision@1, @5, @10 of Google Play and SimApp.

	Precision@1	Precision@5	Precision@10
Google Play	0.688	0.725	0.663
SimApp	**0.875**	**0.819**	**0.769**

Threats to Validity. Despite the encouraging results, this study has three threats to validity. First, Google Play's *"Similar"* function is a "black box" to us, it may consider more factors (e.g., popularity) than just similarity measure, thus may influence the comparison results. Second, our dataset is relatively small, which makes it difficult for SimApp to retrieve similar apps for some query apps, thus lowering SimApp's performance. Third, the limited number of query apps and subjectivity (due to expensive cost of labeling) may affect the accuracy of the results. We plan to examine these threats in great efforts in our future work.

Next, in Figure 7, we show two successful (failure) cases of SimApp (Google Play). The first column of Figure 7 shows two query apps, i.e., "AntiVirus Security - FREE" (antivirus app) and "Subway Surfers" (parkour game). For each query app, we list two lines of top-10 results, where the upper line is ranked by Google Play, and the lower line is ranked by

Figure 7: Two examples of top-10 recommended similar apps (better viewed in color).

SimApp. The name of each app is shown beneath the app logo. One app is similar (not similar) to the query app if its name is in black (red) color.

The **first** example presents the query app "AntiVirus Security - FREE". The top-10 results ranked by Google Play are bad, since only "Lookout Security & Antivirus" (5th) and "Free Antivirus & Security" (10th) are similar to the query app, and the rest 8 apps bear no semantic similarity as they are either communication tools or web browsers. In contrast, SimApp achieves much better results. All 10 apps ranked high provide antivirus functionalities, and thus are similar to the query app. In the following, we illustrate the behaviour of SimApp and explain why it performs well in this example via the results shown in Table 5.

Table 5 presents the similarity scores between the query app "AntiVirus Security - FREE" (denoted as q) and a list of apps computed by SimApp. The "Apps" row of Table 5 lists 10 apps, where apps $a_1 \sim a_5$ and $a_6 \sim a_{10}$ are the top-5 apps ranked by SimApp and Google Play, respectively (shown in Figure 7). Every cell (a_i, K^j) $(i, j \in [1, 10])$ in Table 5 represents the kernel value $K^j(a_i, q)$, which measures the similarity between a_i and q in the j-th modality. The "Total" row shows the overall similarity scores computed by using the kernel weights shown in Table 3. The "Rank" row presents the rank positions given by SimApp.

Some observations can be drawn from the results shown in Table 5. First, app a_1 "AntiVirus PRO Android Security" is reasonably ranked 1st by SimApp, since most kernel values $K^j(a_1, q)(j \in [1, 10])$ are high. Second, app a_6 "WhatsApp Messenger" (not similar to q), which is ranked 1st by Google Play, is reasonably ranked much lower by SimApp (401st). The reason is that, although $K^2(a_6, q) = 1.00$, other kernel (especially kernel with large weight) values are low, e.g., $K^{10}(a_6, q) = 0.07$. Similarly, for apps a_7, a_8, a_9 (all are not similar to q), which are ranked high by Google Play, SimApp also correctly assigns them low ranking positions.

The **second** example shown in Figure 7 presents the query app "Subway Surfers", which is a popular parkour game. The top-10 results ranked by Google Play are also not very good. Half of top-10 results are not semantically similar to the query app as they are not parkour games. For example, the ranked 1st game "Fruit Ninja Free" is a popular juicy action game. SimApp again attains much better results than Google Play. All top-10 results share the same concept with the query app, i.e., "running".

6. CONCLUSION AND FUTURE WORK

This paper presents SimApp, a novel framework for finding similar mobile apps. We found encouraging results from a set of experiments, which not only validate the efficacy but also show the potential application prospect of our technique. In the future, we plan to (i) explore more modalities of apps; and (ii) apply our technique to solve other challenging tasks, e.g., finer-granularity app categorization.

7. ACKNOWLEDGMENTS

This work was supported by a grant (ARC19/14) from MOE, Singapore and a gift from Microsoft Research Asia. Steven HOI's work is funded through a research grant 14-C220-SMU-016 from MOE's AcRF Tier 1 funding support through Singapore Management University. We thank the anonymous reviewers for their greatly helpful comments.

8. REFERENCES

[1] App Annie Special Report: A Look at the Growth of Google Play. `http://blog.appannie.com/google-io-special-report-launch-2014/`.

[2] U. Bhandari, K. Sugiyama, A. Datta, and R. Jindal. Serendipitous recommendation for mobile apps using item-item similarity graph. In *AIRS*, pages 440–451, 2013.

[3] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. *JMLR*, 3:993–1022, Mar. 2003.

[4] G. Chechik, V. Sharma, U. Shalit, and S. Bengio. Large scale online learning of image similarity through ranking. *JMLR*, 11:1109–1135, Mar. 2010.

[5] K. Chen, P. Liu, and Y. Zhang. Achieving accuracy and scalability simultaneously in detecting application clones on android markets. In *ICSE*, pages 175–186, 2014.

Table 5: Similarity scores (computed by SimApp) between the query app "AntiVirus Security - FREE" and a list of apps. $K^1 \sim K^{10}$ are defined in Section 4.1. Kernel values larger than 0.5 are in bold blue. "AntiVirus PRO", "Tablet AntiVirus" and "Lookout" are short for "AntiVirus PRO Android Security", "Tablet AntiVirus Security FREE" and "Lookout Security & Antivirus", respectively.

	Top-5 apps ranked by SimApp					**Top-5 apps ranked by Google Play**				
Apps	AntiVirus PRO (a_1)	Tablet AntiVirus (a_2)	Anti-Virus (a_3)	Antivirus for Android (a_4)	360 Security (a_5)	WhatsApp Messenger (a_6)	LINE (a_7)	Skype (a_8)	Viber (a_9)	Lookout (a_{10})
K^1 (Name)	0.43	**0.68**	0.18	0.20	0.50	0.03	0.02	0.04	0.06	0.13
K^2 (Category)	**1.00**	0.00	**1.00**	**1.00**	0.00	**1.00**	**1.00**	**1.00**	**1.00**	0.00
K^3 (Developer)	**1.00**	**1.00**	**0.56**	0.37	**0.81**	0.44	**0.62**	0.40	0.15	0.40
K^4 (Desc.)	**0.95**	**0.93**	**0.83**	**0.72**	**0.80**	0.03	0.17	0.10	0.05	**0.75**
K^5 (Update)	**0.80**	0.29	0.43	0.37	**0.52**	0.32	0.46	**0.55**	0.37	**0.51**
K^6 (Perm.)	**1.00**	**0.99**	**0.53**	0.29	**0.65**	0.43	0.45	**0.51**	**0.56**	**0.59**
K^7 (Images)	**0.62**	**0.72**	0.48	0.41	0.37	0.26	0.39	0.31	0.33	**0.58**
K^8 (Content)	**1.00**	**1.00**	**1.00**	0.33	**1.00**	0.33	0.33	0.33	0.33	**1.00**
K^9 (Size)	**0.99**	**0.83**	0.41	0.38	**0.96**	0.44	0.16	0.27	0.18	**0.63**
K^{10}(Reviews)	**0.84**	**0.97**	**0.68**	**0.90**	**0.83**	0.07	0.04	0.02	0.02	0.49
Total	0.81	0.70	0.63	0.62	0.58	0.30	0.33	0.31	0.27	0.41
Rank	1	2	3	4	5	401	137	311	615	57

[6] N. Chen, J. Lin, S. C. H. Hoi, X. Xiao, and B. Zhang. Ar-miner: Mining informative reviews for developers from mobile app marketplace. In *ICSE*, pages 767–778, 2014.

[7] Y. Chen, H. Xu, Y. Zhou, and S. Zhu. Is this app safe for children?: A comparison study of maturity ratings on android and ios applications. In *WWW*, pages 201–212, 2013.

[8] S. C. Hoi, R. Jin, P. Zhao, and T. Yang. Online multiple kernel classification. *Mach. Learn.*, 90(2):289–316, Feb. 2013.

[9] S. C. Hoi, J. Wang, and P. Zhao. Libol: A library for online learning algorithms. *JMLR*, 15:495–499, 2014.

[10] Z. Li, S. Lu, S. Myagmar, and Y. Zhou. Cp-miner: Finding copy-paste and related bugs in large-scale software code. *IEEE Trans. Softw. Eng.*, 32(3):176–192, 2006.

[11] J. Lin, K. Sugiyama, M.-Y. Kan, and T.-S. Chua. New and improved: Modeling versions to improve app recommendation. In *SIGIR*, pages 647–656, 2014.

[12] C. Liu, C. Chen, J. Han, and P. S. Yu. Gplag: Detection of software plagiarism by program dependence graph analysis. In *KDD*, pages 872–881, 2006.

[13] H. Lodhi, C. Saunders, J. Shawe-Taylor, N. Cristianini, and C. Watkins. Text classification using string kernels. *JMLR*, 2:419–444, Mar. 2002.

[14] J. Ma, L. K. Saul, S. Savage, and G. M. Voelker. Identifying suspicious urls: An application of large-scale online learning. In *ICML*, pages 681–688, 2009.

[15] C. McMillan, M. Grechanik, and D. Poshyvanyk. Detecting similar software applications. In *ICSE*, pages 364–374, 2012.

[16] C. K. Roy and J. R. Cordy. A survey on software clone detection research. *SCHOOL OF COMPUTING TR 2007-541, QUEEN'S UNIVERSITY*, 115, 2007.

[17] S. Shalev-Shwartz. Online learning and online convex optimization. *Found. Trends Mach. Learn.*, 4(2):107–194, Feb. 2012.

[18] J. C. Spall. *Introduction to Stochastic Search and Optimization*. John Wiley & Sons, Inc., New York, NY, USA, 1 edition, 2003.

[19] S. Ugurel, R. Krovetz, and C. L. Giles. What's the code?: Automatic classification of source code archives. In *KDD*, pages 632–638, 2002.

[20] N. Viennot, E. Garcia, and J. Nieh. A measurement study of google play. In *SIGMETRICS*, pages 221–233, 2014.

[21] P. Wu, S. C. H. Hoi, H. Xia, P. Zhao, D. Wang, and C. Miao. Online multimodal deep similarity learning with application to image retrieval. In *MM*, pages 153–162, 2013.

[22] H. Xia, S. C. H. Hoi, R. Jin, and P. Zhao. Online multiple kernel similarity learning for visual search. *IEEE Trans. Pattern Anal. Mach. Intell.*, 36(3):536–549, 2014.

[23] H. Xia, P. Wu, and S. C. H. Hoi. Online multi-modal distance learning for scalable multimedia retrieval. In *WSDM*, pages 455–464, 2013.

[24] P. Yin, P. Luo, W.-C. Lee, and M. Wang. App recommendation: A contest between satisfaction and temptation. In *WSDM*, pages 395–404, 2013.

[25] W. Zhou, Y. Zhou, X. Jiang, and P. Ning. Detecting repackaged smartphone applications in third-party android marketplaces. In *CODASPY*, pages 317–326, 2012.

[26] H. Zhu, H. Cao, E. Chen, H. Xiong, and J. Tian. Mobile app classification with enriched contextual information. *TMC*, 2013.

[27] H. Zhu, H. Xiong, Y. Ge, and E. Chen. Ranking fraud detection for mobile apps: A holistic view. In *CIKM*, pages 619–628, 2013.