Online Appendix A: Copycat Detection Technical Process

This online appendix describes the technical details of the copycat detection process.

Step 1: Detecting Functional Similarity Based on Textual Descriptions and Customer Reviews Using Natural Language Procession (NLP)

In the first step, we detect app features by mining the descriptions and app reviews. The ultimate output of this step is a square matrix where each cell represents the pairwise app distance in the app feature space. To produce the output, we first acquire app specific documents by merging the app descriptions with consumer reviews per app. Then the app documents are sorted by the app ID in ascending order. Meanwhile, we merge all the app documents into an overall raw text file. We follow a standard text pre-processing procedure to perform a series of text preprocessing on this raw file, including tokenization, removing stop words, Part-of-Speech tagging, and removing duplicate words. After Part-of-Speech tagging, words other than nouns and verbs are filtered out. The remaining nouns and verbs form a set of app features denoted as T. This is our app feature corpus. Then we use the TFIDF method (Salton and McGill 1983) to extract app features.

To present the mathematical calculation of the TFIDF approach, it is helpful to introduce a few notations. Let $d_j \in$ D be the document corresponding to the j^{th} app in the list of app documents D. Let N be the total number of documents in D. Let $t_i \in$ T be the i^{th} term in corpus T. Let M be the total number of terms in T. And let n_{t_i} denote the number of documents in which the term t_i occurs at least once. The TFIDF score of term t_i in document d_j is just a product of the term frequency and the inverse document frequency.

$$tfidf_{t_i,d_j} = tf_{t_i,d_j} idf_{t_i,d_j}$$

where the term frequency tf_{t_i,d_j} is the count of how many times the term t_i occurs in document d_j. And the inverse document frequency idf_{t_i,d_j}, which is typically measured in the logarithm scale, is the inverse of document frequency that is defined as the probability of t_i showing up in any document in D. Mathematically,

$$df_{t_i,d_j} := Prob(t_i \; occurs \; in \; any \; document) = \frac{n_{t_i}}{N}$$

$$idf_{t_i,d_j} := -\log\left(df_{t_i,d_j}\right) = \log(\frac{N}{n_{t_i}})$$

The purpose of TFIDF is to find the representative terms of a document by choosing the terms occurring many times within the document but only occurring in a small number of documents in general. In the operationalization of this idea, the algorithm overweighs the terms that are rare in the document population but are concentrated in a few documents. And the algorithm under weighs the terms that occur everywhere.

After the TFIDT calculation, the document d_j can be represented by a numerical column vector $\boldsymbol{T}_{d_j} = [tfidf_{t_1,d_j}, tfidf_{t_2,d_j}, \dots, tfidf_{t_M,d_j}]^T$. And the collection of documents D can be represented by a numerical matrix $\boldsymbol{M}_D = [\boldsymbol{T}_{d_1}, \boldsymbol{T}_{d_2}, \dots, \boldsymbol{T}_{d_N}]$. In practice, most of the elements in $\boldsymbol{T}_{d_j}$ and $\boldsymbol{M}_D$ are zeroes, indicating that the distribution of the app features is sparse. Furthermore, the TFIDF algorithm assumes that the terms are independent of each other; it ignores the synonymy, polysemy and underlying correlations between terms. To allow the terms to be dependent and to present the apps in a more concise way, we use Latent Semantic Analysis (LSA), in particular Singular Value Decomposition (SVD) to reduce the dimensions of the app feature space. Specifically,

$$\boldsymbol{M}_D = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^{\boldsymbol{T}}$$

where $\boldsymbol{U}$ is an M by M unitary matrix, $\boldsymbol{\Sigma}$ is an M by N diagonal matrix, and $\boldsymbol{V}^{\boldsymbol{T}}$ is an N by N unitary matrix. The diagonal entries σ_{ii} of $\boldsymbol{\Sigma}$ are singular values of $\boldsymbol{M}_D$. If σ_{ii} is negligible or close to zero, it means that adding the ith component does not contribute much to the variation of $\boldsymbol{M}_D$ and hence σ_{ii} can be replaced by zero without losing valuable information in D.

After reducing the dimensions of the app feature space, we measure app pairwise similarity using the cosine similarity. The cosine similarity of apps *i* and *j* is defined as:

$$sim_{d_i,d_j} = \frac{\boldsymbol{T}_{d_i} \cdot \boldsymbol{T}_{d_j}}{\left|\boldsymbol{T}_{d_i}\right| \left|\boldsymbol{T}_{d_j}\right|} = \frac{\sum_{k=1}^{M} tfidf_{t_k,d_i} tfidf_{t_k,d_j}}{\sqrt{\sum_{k=1}^{M} tfidf_{t_k,d_i}^2} \sqrt{\sum_{k=1}^{M} tfidf_{t_k,d_j}^2}}$$

In addition to cosine similarity, we also try other similarity scores such as Pearson Correlation Measure, Extended Jaccard Measure, and Dice Coefficient Measure. They yield consistent similarity measures. The cosine similarities of all app pairs form the final output of the square matrix **S** where $\boldsymbol{S}_{ij} = sim_{d_i,d_j}$.

Step 2: Network-Based Clustering Using the Markov Clustering Algorithm

The main goal of this step is to identify clusters of apps where the apps within a cluster are similar to each other while apps across clusters are dissimilar. To achieve this goal, we apply a network-based clustering method, because the square matrix **S** can be viewed as an undirected probabilistic graph. In this graph, a node corresponds to an app. An arc corresponds to the pairwise similarity of apps. Therefore, our goal is to cluster the graph.

In particular, we implement the Markov clustering algorithm (MCL) introduced by Dongen in 2000. The intuition of the algorithm is about iterative random walk. In each round of the iteration, the probability of visiting a connected node is proportional to the weight of the arc. In other words, the similarity matrix is expanded by itself. Then the similarity matrix is transformed by a column-wise inflation operation. The random walk will stabilize inside the dense regions of the network after many iterations. The stabilized regions become the clustering output and reflect the intrinsic structure of the network. The detailed clustering algorithm can be represented as follow.

Algorithm 1: Generating clusters of apps

Input: Similarity matrix **S**, power parameter e, inflation parameter r, convergence threshold δ.

Output: Clusters of apps represented by matrix **S**.

Add self-loops **S** = **S** + **I** ;

Normalize the matrix $\boldsymbol{S}_{ij} = \frac{\boldsymbol{S}_{ij}}{\sum_{k=1}^{N} \boldsymbol{S}_{kj}}$;

While $\mathbf{S} - \mathbf{S}^2 > \delta$ **do**

  $\mathbf{S} = \mathbf{S}^e$;

  For $i \in \{1, \ldots, N\}$

For $j \in \{1, \ldots, N\}$

$$S_{ij} = \frac{s_{ij}}{\sum_{k=1}^{D} s_{kj}^{r}};$$

Once we extract the clusters of similar apps, the next step is to distinguish the original apps from the copycats. We consider the app release date as the objective standard of originality. If an app is the first app released in a cluster, it is labeled as original. Otherwise, it is labeled as a copycat app. However, if the original developer has released several apps in the same cluster, e.g., "Angry Birds" and "Angry Birds Space", all these apps are labeled as original, but the ones except the first released app are also marked as sibling apps.

Step 3: Detecting Deceptive and Non-Deceptive Copycat Apps

In step 3, we classify whether a copycat app is deceptive or non-deceptive based on the app's appearance. A copycat app is labeled as deceptive if *either* its title is similar to the original app's title, *or* its icon is similar to the original app's icon. And a copycat app is labeled as non-deceptive if neither the title nor the icon is similar to the original app. To classify copycats apps, we conduct two separate analyses using string soft matching (Step 3a) and image matching analysis (Step 3b).

Step 3a: Detecting Appearance Similarity in App Titles Using String Soft Matching

We conduct string soft matching to measure title similarity using the edit distance method. The edit distance is defined as the minimum number of editing operations needed to convert from one string to the other (Elmagarmid et al. 2007). For each copycat app in a cluster, we compute the edit distance between the copycat app name and the original app name. Then the edit distance is normalized to a value between 0 and 1 using the Jaccard Index. Following the widely accepted rule of thumb threshold distance of 0.7 (Kim and Lee, 2012), we label a copycat as deceptive if its normalized edit distance is smaller than the threshold.

Step 3b: Detecting Appearance Similarity in App Icons Using Image Matching Analysis

To detect imitations of app icons, we need an efficient image matching algorithm. In the mobile app market, copycat providers may not copy the exact image of the original app. Instead, copycat developers are likely to rescale, rotate, recolor, or add noises to the original icon. Therefore we need a detection algorithm that is robust to image scaling, rotation, lighting, and illumination change, etc.

To address the challenge, we employ the Scale-Invariant Feature Transform (SIFT) algorithm proposed by Lowe (1999). The algorithm is one of the most popular and influential image matching algorithms in the field of computer vision (Mikolajczyk and Schmid 2005). The algorithm consists of the following six steps.

3b.1 Input Image Preprocessing

In the preprocessing stage, all pixels in the source image are converted to a floating point grayscale that falls in the range of [0, 1]. This reduces the full color images to the grayscale equivalent. The conversion follows the YIQ color model (Hearn & Baker 2004).

$$Gray = 0.299Red + 0.587Green + 0.144Blue.$$

Based on the pixel values, we transform an image to be a matrix. The horizontal axis is represented by the x axis with the positive direction to the right. And the vertical axis is represented by the y axis with the positive direction downward. Let $f(x, y)$ be the grayscale with respect to the pixel at location (x, y).

3b.2 Scale-Space Image Representation

In the initial stage of the algorithm, the source image is blurred by the following Gaussian kernel transformation. In consequence, the low level noises are averaged out quickly and the remaining image is robust to small noises. The kernel transformation technique is very common in many edge detection and object recognition algorithms.

$$G(x, y; \sigma) = \frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)/2\sigma^2},$$

$$L(x, y; \sigma) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} G(\tau_x, \tau_y; \sigma) f(x - \tau_x, y - \tau_y) d\tau_x d\tau_y,$$

where σ represents the scale parameter and L is the convolution operation of functions G and f. Now we create an image pyramid by repeatedly boosting the scale parameter σ by $k = \sqrt{2}$. By doing so, we acquire a series of images with different smoothing levels. Then we repeatedly resize the image pyramids to create a set of octaves.

3b.3 Keypoint Computation by the Difference of Gaussians (DoG)

In this step, the purpose is to find all potential keypoints of the source image. Potential keypoints are local extrema that have high contrast grayscales compared with the surrounding points. The potential keypoints

are found by calculating a spatial information indicator called Difference of Gaussians (DoG). The DoG of a pixel on a certain level of the image pyramid can be calculated as follows.

$$D(x, y; \sigma) = L(x, y; k\sigma) - L(x, y; \sigma)$$

In particular, potential key points are found by comparing a pixel with its 8 neighbors on the same level, 9 in the interval above, and 9 in the interval below. If and only if a pixel is greater or less than all of its 26 neighbors in terms of DoG then it is considered as a potential key point.

3b.4 Contrast-Based Edge Filter

Once all potential keypoint candidates have been identified, we remove the bad keypoint candidates that are unstable and only retain the stable ones. A stable keypoint represents a corner, meaning that two lines end at this point. It is a good feature to detect because it is scale invariant. There are two types of unstable keypoints: the ones with low contrast and the ones on the edges. Keypoints with low contrast are unstable because if the lighting conditions change, the keypoints produced by shadows will change the location very easily. Keypoints on the edges are not ideal either because the length of the edge can change when the image is rotated or deformed. These two types of keypoints have different properties and should be handled differently.

To reject the first type of unstable keypoints with low contrast, we compare the difference between a pixel and its neighbors using second-order Taylor expansion:

$$D(\boldsymbol{v}) = D + \frac{\partial D^T}{\partial \boldsymbol{v}}\boldsymbol{v} + \frac{1}{2}\boldsymbol{v}^T\frac{\partial^2 D}{\partial \boldsymbol{v}^2}\boldsymbol{v}\text{, where } \boldsymbol{v} = (x, y, \sigma)^T.$$

Next, we compute the location of the extreme value $\boldsymbol{z}$ by taking the derivative of this function $D(\boldsymbol{v})$ with respect to $\boldsymbol{v}$ and setting it to zero, giving

$$\boldsymbol{z} = -\left(\frac{\partial^2 D}{\partial \boldsymbol{v}^2}\right)^{-1}\frac{\partial D}{\partial \boldsymbol{v}}.$$

Then, we substitute $\boldsymbol{z}$ to the first order Taylor expansion, giving

$$D(\boldsymbol{z}) = D + \frac{1}{2}\frac{\partial D^T}{\partial \boldsymbol{v}}\boldsymbol{z}.$$

If $D(\boldsymbol{z})$ is below a certain threshold value, it means the pixel is in a low contrast region. Therefore the candidate keypoint is less desirable and should be excluded.

To reject the second type of unstable points on the edges, we evaluate the principle curvatures using the Hessian matrix:

$$\mathbf{H}=\begin{bmatrix} \frac{\partial^2 D}{\partial x^2} & \frac{\partial^2 D}{\partial x \partial y} \\ \frac{\partial^2 D}{\partial x \partial y} & \frac{\partial^2 D}{\partial y^2} \end{bmatrix}.$$

Then we check if $\frac{Tr(\boldsymbol{H})^2}{Det(\boldsymbol{H})} < \frac{(r+1)^2}{r}$ where r is a hyper-parameter that refers to a desired threshold value, $Tr(\boldsymbol{H}) = \frac{\partial^2 D}{\partial x^2} + \frac{\partial^2 D}{\partial y^2}$, and $Det(\boldsymbol{H}) = \frac{\partial^2 D}{\partial x^2}\frac{\partial^2 D}{\partial y^2} - (\frac{\partial^2 D}{\partial x \partial y})^2$. If the inequality fails, it means one gradient perpendicular to the edge is big while the other gradient along the edge is small. It indicates this pixel is an edge instead of a corner. We remove such pixels from the keypoint candidates.

3b.5 Keypoints Orientation

The idea of this step is to collect gradient directions and magnitudes around each keypoint. Then we figure out the most important orientation in that region. Finally, we assign this orientation to the keypoint. To efficiently calculate the gradient magnitude and orientation, we use finite differences as follows:

$$m(x,y;\sigma) = \sqrt{(L(x+1,y,\sigma) - L(x-1,y,\sigma))^2 + (L(x,y+1,\sigma) - L(x,y-1,\sigma))^2},$$

$$\theta(x,y;\sigma) = arctan\frac{L(x,y+1,\sigma) - L(x,y-1,\sigma)}{L(x+1,y,\sigma) - L(x-1,y,\sigma)}.$$

The gradient magnitudes and orientations are calculated for all pixels around the keypoint. Then a histogram is created. The peak of the histogram is used as the orientation of the keypoint. For each other orientation within 80% of the maximum orientation, a new keypoint with this orientation is created.

3b.6 SIFT descriptor computation

In the end of last subsection, each keypoint was represented by a vector of five dimensions $\mu = (\mathrm{x}, \mathrm{y}, \sigma, m, \theta)^T$. Go through the above process again for a second image. Then we can decide whether the two images are matched by comparing the keypoints. Let P, Q be the lists of keypoints of two images. A keypoint p of P matches the keypoint q of Q if

$$|D(p) - D(q)|^2 = \min_{r \in Q} |D(p) - D(r)|^2$$

If there is a big difference between the best match and the second-best match, say $\frac{|D(p)-D(q)|^2}{\min_{r\in Q-\{q\}}|D(p)-D(r)|^2}$ is bigger than a pre-defined threshold, then the match is acceptable; otherwise it is ambiguously matched and is rejected. After this comparison, the targeted two images are either labeled as matched or unmatched. And this is the end of the entire copycat detection process.

Online Appendix B: Screenshot of MTurk Questionnaire

Overview

- Hi there! Thanks for being interested in this study. We are a group of researchers in Carnegie Mellon University. Your contribution to this study will notably help us learn the craft of mobile applications (apps). Moreover, the study is fun!
- To qualify for taking the HIT, you must be an iPhone user and you must have used at least three mobile apps on your phone.
- In the following HIT, you are expected to answer 5 multiple choice questions. It takes approximately 5 minutes to finish a HIT. If your HIT is accepted, you will be paid by $0.05 within 8 hours. You can take multiple HITs if you want to.
- To ensure the quality of the work, some questions with known answers are included in the HIT. If your answer to the questions do not match the known answers, your HIT will be rejected. To ensure independence of the results, please do not discuss with your friends about the HIT content.

Task Description

- In each question, you are expected to visit the home pages of a pair of iPhone mobile apps. After visting the home pages, you need to decide whether the apps are similar in terms of **name, icon image, or gameplay**.
- If ANY of the above three aspects (name, image, and gameplay) is similar, please mark the two apps as similar. Then, please specify the (one or more) similar aspects.
- Meaning of **similar name:** the names of the apps contain the same lexical words. People can possibly take one app name as the other.
- Examples of similar names: "Ball Roll" -- "Ball Rolling", "Anns Resort" -- "Anns Resort Lite", "Cut the Bird" -- "Cut the Fruits", "Buddyman: Kick 2"--"Kick the Buddy: Second Kick"
- Meaning of **similar icon image:** the objects in the images are the same, while the colors, scales and orientations of the images can be different. People can possibly take one app icon as the other.
- Meaning of **similar gameplay:** the apps provide similar game rules, challenges, storylines, and environments. For example
- If the above tutorial is clear, you are ready to go!

Question 0. Please list the top three most frequently used apps on your iPhone:

1. Top most frequently used app

2. Second most frequently used app

3. Third most frequently used app

Question 1. Please read the home page of app 1:

Click here to visit app 1

Please read the home page of app 2:

Click here to visit app 2

Are the apps similar:

○ Yes
○ No

If yes, which aspects are similar: (If not, please don't check)

☐ Name
☐ Icon
☐ Gameplay

Question 2. Please read the home page of app 1:

Online Appendix C: Potential Discontinuous Effect and Alternative Copycat Definition

	Potential Discontinuous Effect				Alternative Copycat Definition			
	(1) Overall	**(2)Quality differences**	**(3)Deceptiveness**	**(4)Quality and deceptiveness**	**(1) Overall**	**(2)Quality differences**	**(3)Deceptiveness**	**(4)Quality and deceptiveness**
Non-Zero Copycat Download Dummy	-0.0204	-0.1105	-0.1455	-0.1322				
	(0.0949)	(0.0970)	(0.0975)	(0.0972)				
Log Copycat Apps	0.0043				0.0507			
	(0.0245)				(0.1549)			
Log High Quality Copycat		-0.0276*				-0.0409***		
		(0.0118)				(0.0078)		
Log Low Quality Copycat		0.0230***				0.1055***		
		(0.0052)				(0.0112)		
Log Deceptive Copycat			0.0292***				0.0769	
			(0.0022)				(0.0462)	
Log Non-deceptive Copycat			-0.0137				-0.0305***	
			(0.0091)				(0.0063)	
Log High Quality Deceptive Copycat				-0.0166***				-0.0231
				(0.0022)				(0.0209)
Log Low Quality Deceptive Copycat				0.0348***				0.0303***
				(0.0069)				(0.0076)
Log High Quality Non-deceptive Copycat				-0.0228***				-0.0439***
				(0.0039)				(0.0068)
Log Low Quality Non-deceptive Copycat				0.0131***				0.0755***
				(0.0047)				(0.0093)
Individual Fixed Effect	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Time Fixed Effect	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Adjusted/pseudo-R^2	0.1756	0.1769	0.1756	0.1791	0.1855	0.1594	0.1515	0.1548
Number of Apps	2,328	2,328	2,328	2,328	1,720	1,720	1,720	1,720
N	77,733	77,733	77,733	77,733	132,112	132,112	132,112	132,112

Std. Err. in parentheses * p < 0.1, ** p < 0.05, *** p < 0.01

Variables controlled for but not reported above include Log Original Price, Original Version, Log Developer Download, Dev Version, App Age, App Age^2, and Log Search