Sovereignty of the ~~people~~ apps: introducing app-usage and trend filters based on actual popularity and usage rather than recommendations

1st Author Name
Affiliation
Address
e-mail address
Optional phone number

2nd Author Name
Affiliation
Address
e-mail address
Optional phone number

3rd Author Name
Affiliation
Address
e-mail address
Optional phone number

ABSTRACT

In this paper we describe the formatting requirements for SIGCHI Conference Proceedings, and this sample file offers recommendations on writing for the worldwide SIGCHI readership. Please review this document even if you have submitted to SIGCHI conferences before, some format details have changed relative to previous years.

Author Keywords

Guides; instructions; author's kit; conference publications; keywords should be separated by a semi-colon.
Optional section to be included in your final version, but strongly encouraged.

ACM Classification Keywords

H.5.m. Information Interfaces and Presentation (e.g. HCI): Miscellaneous

See: `http://www.acm.org/about/class/1998/` for more information and the full list of ACM classifiers and descriptors. Optional section to be included in your final version, but strongly encouraged. On the submission page only the classifiers letter-number combination will need to be entered.

INTRODUCTION

With continuing popularity of current smartphone operating systems, the number of available apps in the respective app stores continues to grow rapidly. The google play store currently features more than 1.4 million apps and Apple's IOS app store more than 1.3 million apps. Follow-ups are the windows playstore (>300.000), Amazon appstore (> 240.000) and BlackBerry world (> 130.000)[1].

It has become easy to find an app for virtually any possible category but challenging to identify good and reliable apps from this overwhelming assortment. Often, this tedious process might involve installing multiple apps and testing each of them individually. Although users are typically supported by a recommendation system in which apps can be rated on, such systems favour apps with a bigger crowd of users such as corporation-developed apps or older and therefore better known apps. For instance, an app which is popular and frequently used in a small community of users might have low chance to be discovered by a greater public due to less ratings. Absolute user number wins over relative high satisfaction of a smaller crowd of app-users. Paradoxically, offensively annoying users to submit recommendations is a superior strategy to just satisfying users with a superb app.

Furthermore, the system can be to some extent subject to fraud. And, most importantly, since they are based on historical data, recommendations might not reflect the actual usage patterns as well as current popularity and trends.

We propose the use of an app-usage filter which shows actual relative popularity of apps based on their daily usage, indicates trends regardless of absolute volume and identifies app-lifecycles. Lifecycles are, for instance, flops, dead applications, trendy apps quickly gaining users as well as apps of continouous high popularity.

We exploit the Carat data **[?]** collected from **???** phones over a period of **???** months (**???** till **???**) to demonstrate our approach. Based on this data, we cluster app-usage time series with similar patterns together in order to identify typical app life-cycles.

From these lifecycle-templates we are then able with alignment matching methods to identify for new apps at which stage in the lifecycle they are currently.

RELATED WORK

OVERVIEW

Overall, our system consists of several modules to process the app-time-series data. First, we extract from the CARAT usage statistics (count of installations of a specific app on a daily basis) the usage frequency for individual apps. This time series is potentially discontinuous and absolute numbers vary greatly among distinct apps.

Therefore, as a next step, these time sereis are then normalised to relative usage counts between 0 and 1, where 1

[1] http://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/

corresponds to the maximum usage count. In addition, missing values for intermediate days are interpolated so taht each time series has a valid value on each day.

These time series are then clustered for similar time-series patterns. The idea behind this is that an app undergoes an app-lifecycle and that it might be possible to represent this lifecycle with these usage patterns. We aim to group alll app with similar lifecycles together.

For this, time-series are first aligned at their peak before clustering them with k-means clustering approach. Features for the k-means are the variance of the time series, the area under the curve and the location of the peak relateive to the time-series length.

From each cluster, a consensus-lifecycle as a representative is generated by calculating for each respective day the mean from all time series in that cluster.

For a time-series of a new app, we are then able to identify the consensus-lifecycle to which it is most similar and where in that lifecycle it currently is by applying alignment methods as described in section **??**.

Description of the system (schematic view)

- normalize
- shift and interpolate
- cluster (k-means + features)
- Consensus
- Alignment (see ISWC note)

Normalisation of Timeseries to make them comparable

normalise, shift, interpolate

Clustering timeseries

k-means and features

Compute consensus timeseries

explain the approach

Alignment to find trend-state

explain alignment method – revise below text

In the algorithm, two strings are compared for the best approximative pattern among them, as depicted in figure 1. The figure illustrates the operation to find a sub-sequence similar to a pattern $p = p_1 \dots p_k$ in a longer sequence $s = s_1 \dots s_n$. The matrix is filled from left to right and from top to bottom. Each matrix entry M_{ij} contains the minimum cost to align $p_1 \dots p_i$ with $s_1 \dots s_j$. It is obtained iteratively from the cost of the cells $M_{i-1,j-1}, M_{i-1,j}, M_{i,j-1}$ by choosing the minimum cost for an extension of any of these three alignments (extension of both strings, extension of only the first string or extension of only the second string). The first row is initialised with 0 to allow the pattern to start at any position within s. The careful choice of the gap-penalty g is crucial for the approach to produce good matching results. The best

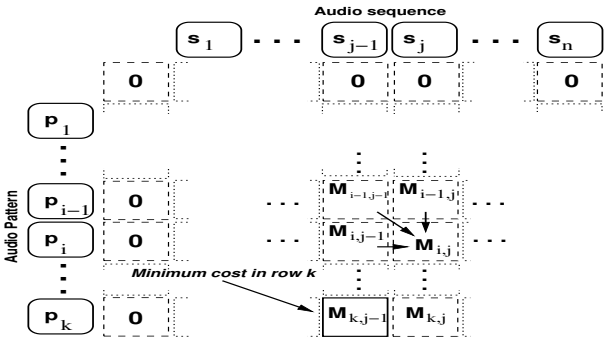

Figure 1. Schematic illustration of the Smith-Waterman pattern matching

matching string is obtained by backtracking the smallest entry in the last row.

Since the matching is approximative, we will always find a best matching position even though the absolute similarity of this very position might not be high. we exploit this property to be able to resign from any information transmitted among devices on the actual recorded audio. In particular, we utilize a predefined, characteristic pattern on both devices. since the pattern is known in advance, no information need to be transmitted. The downside to this implementation is, of course, that the pattern utilised might be very different from the actual audio recorded. However, since the Smith-Waterman algorithm always computes a best matching, we can speculate that this best matching is found at similar positions in the audio recordings, provided that the data has significant similarity. The best matching, although it might not be a good matching in absolute terms, is likely to be found at a similar position in the recorded audio of the remote devices.

The specific pattern used for matching is extracted randomly from consecutive samples of an arbitrary audio sequence. In our experiments, its length is 100 samples. Longer patterns increase running time of the algorithm without improving the accuracy of the matching. Due to the fact that our sampling rate is 44100Hz, a 100-sample pattern is equivalent to a 0.003-second audio chunk.

The pattern p is matched in the first part of each audio file s, which is 100000 samples long. The matching score of p and a local part l of s is the difference between amplitude values in p and l. The less the score, the more similar p and l are. We maintain a matrix to store the scores and another for tracing. Backtracking starts at the lowest scoring matrix cell. According to our experiments, the gap penalty that can yield an acceptable matching is 150. After finding the matching position, we can eliminate all samples preceding this position and generate the audio fingerprint from the remaining ones.

PAULAS METHOD

RESULTS

- top 102 apps
- 4 clusters
- show the four clusters (timeseries)

- and also show the consensus TS for each

CONCLUSION

ACKNOWLEDGMENTS

REFERENCES