Predicting Breakout Apps in IOS

CS 341 Project Report
Quaizar and Saurabh
Mentor: Prof. Anand Rajaraman
With help from: Sailesh Ramakrishnan and Mike Chrzanowski

Problem Description

Over 230 apps are released on the iOS app store are released every day. This number gives an average release rate of 84,000 apps per year. However, less than 1 in 500 will go on to break into the top of popularity lists, and stay there for a considerable time. We will refer to such apps as *breakout* apps.

Our objective was to identify apps that will go on to become breakout apps *before* they have established themselves so. Specifically, we set out to predict apps that will enter the top M download ranks, and stay there for a minimum of N days. This is illustrated in the graphic below.

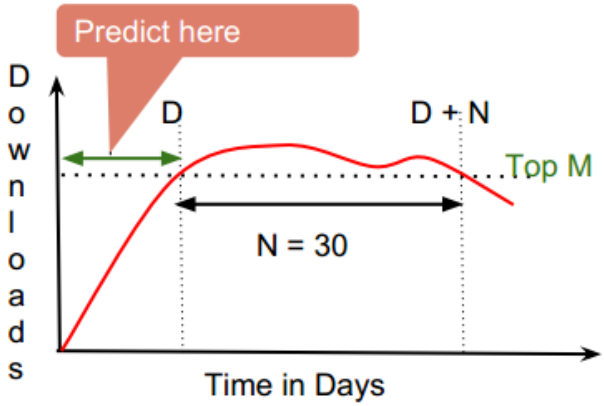

Figure 1: Breakout Identification objective

Data Sources

App Ranks

We were provided daily rank data for top 400 apps, including the appropriate rank category ('game', 'social networking') and subcategory ('game > puzzle') feed (IPAD/general, free/paid). Some of these details are shown below:

1. Apps: 399k apps, including release date, price, currency and publisher

id	name	price	zip_raw	currency	release_date	publisher_id
281656475	PAC-MAN	1.99	BLOB	GBP	2008-07-10...	281656478
281704574	AIM (Free): Fr...	0	BLOB	USD	2008-07-10...	281704577
281736535	Enigmo	1.99	BLOB	USD	2008-07-10...	281736538
281747159	Cro-Mag Rally	2.49	BLOB	GBP	2008-07-10...	281736538
281796108	Evernote	0	BLOB	GBP	2008-07-10...	281796111
281826146	SF Classic	0	BLOB	GBP	2008-07-10...	281826149

2. Ranks: Daily rank graph for top 400 apps

app_id	category_id	graph	feed_id	country
281656475	36	{"2014-02-09": 326, "2014-02-12": 391, "...	2	au
281656475	36	{"2014-01-28": 351, "2014-01-29": 244, "...	2	ca
281656475	36	{"2014-01-28": 337, "2014-01-29": 341, "...	2	gb
281656475	36	{"2014-03-15": 75, "2014-03-16": 235, "2...	2	ph

There was additional data with information on feeds and publishers.

Twitter

We were given access to a 10% firehose of Twitter for the months of March and April (and 1% for the 2 months before). This dataset was massive, with the following attributes:

- Twitter Firehose (12 TB): 4.5 Billion tweets
- Tweet fields: body, retweets, favorites, url, timestamp etc.
- Approx 1% of all tweets seem to be app-related

App Reviews (crawled for the project)

In the absence of app downloads data, it was challenging to define breakout apps (discussed in further detail in the next section). To circumvent this issue, we crawled user ratings at http://itunes.apple.com to build our training set.

- iOS App Store does not provide downloads data
- Intuition: Apps that breakout should have higher number of reviews and/or better ratings
- Daily App Ratings crawled to generate app review counts, user ratings

Impediment: Breakout Definition

We had set out to predict apps that will stay at the top of downloads list for a considerable time. However, identifying apps with this definition posed several challenges:

1. The app store does not provide the running count of downloads per app. Hence, using downloads as a criteria directly is not possible.
2. We tried to use 'app ranking' as a criteria, which brought up a new issue: an app could obtain a top rank in an obscure feed (e.g. 'IPAD > Education') and be considerably less popular than a lower rank in another feed (e.g. 'IPhone > Games > Puzzle').
3. We narrowed our data set to only top-level categories, e.g. games, finance, travel etc. But we found that games category has a very large proportion of breakouts compares to other top-level categories.
4. We realized that number of reviews/ratings would be proportional to the actual number of downloads. We tried narrowing down our data set to categories which have a high average number of ratings/reviews. Here we found that we were losing a large number of popular apps (seen from the tweet/review signals) while retaining apps which had no clear signal. What we realized is that popularity depends on individual apps and not so much on category.

From the above experiments, we realized is that # of ratings of an app is a good proxy for downloads and we should directly use that in our breakout definition rather than trying to narrow our dataset by categories or feeds. We thus augmented our breakout definition by adding a minimum number of reviews as a threshold.

Breakout definition:

App Rank <= 25, for 30 or more days, *with at least 2,000 reviews*

Pipeline and Other Impediments

(*Readers not interested in pipeline and feature design may skip to the 'Experiment & Results' section)*

We began by identifying breakout, fizzled and non-breakout apps and they designing features for these, based on data from Twitter and App Ratings. Below is a schematic of the pipeline.

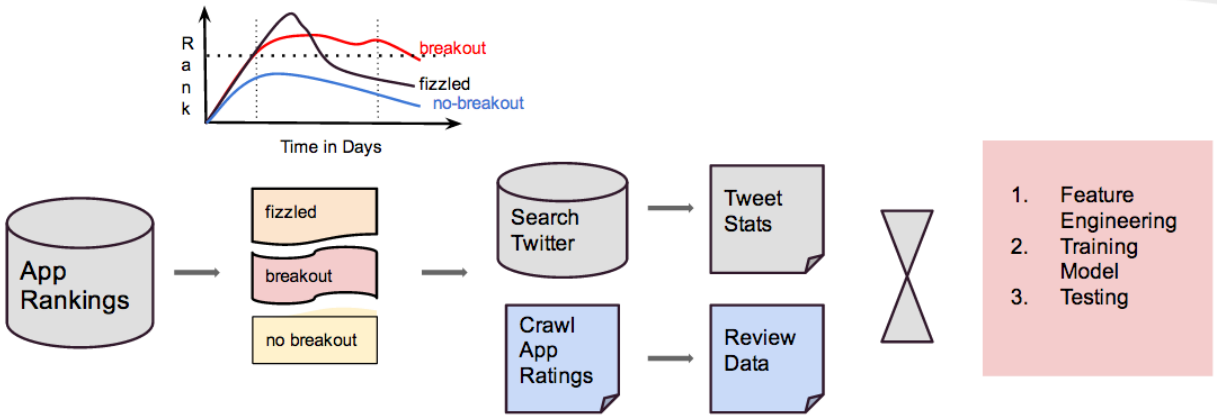

Figure 2: Data Pipeline

Since collecting data and manufacturing features was the most challenging and time consuming task for our project, we give a brief description of our pipeline and the challenges we faced and how we solved those.

First Stage : App Selection

The first stage in our pipeline select apps at 3 levels of breakout based on our breakout criteria: breakout, fizzled (apps that broke in to the top 25, but did not stay there for sufficient time) and no-breakout (apps which did not show any sign of breakout)

First, We identify likely breakout apps by analyzing the App ranks data using our breakout criteria of top 25 for 30 or more days. We further whittle it down by App ratings, i.e. Apps which have 2000 or more ratings. For this we wrote a crawler to crawl the ratings of all the Apps. Note that these were static ratings on a particular date (05/21/2014). We don't have time series data for this. *We can build time series data by crawling reviews, but wholesale crawling of reviews is a very slow process and is blocked by App store. As we describe in the 2nd stage, we do crawl reviews, but only for apps which are screened by this stage*.

In our case we have app-rankings from 01/15/2014 to now. But we only select apps that were released after March 15th, 2013 to match our Twitter Firehose Data. We only have Twitter data from March onwards until now. The first stage yields 25 breakout Apps. Then breakout & fizzled apps were selected using the the

category breakdown of breakout apps, i.e. number of apps across separate categories were selected in the same proportion as that of breakout apps. In all we selected 92 fizzled apps and 209 no-breakout apps.

Second Stage: Twitter Search & Review Crawl

Twitter Firehose Search:

The twitter firehose data is searched for tweets related to apps selected in the first stage. Since we have a very large number of tweets (4.5M Tweets in 12 TB of data) tweets we used the mapreduce framework. The search itself is akin to phrase query search with case folding. We use exact prefixes of app names in our search. Most tweets use exact app names. Large names are typically truncated to 3 to 5 words in the tweets. We also search for likely hashtags by coalescing the app names and adding suffixes like “games”, “ios” etc.

Impediment: Identifying app-related tweets

A large number of tweets may contain what looks like an app name but are not related to apps. This is due to the fact that a large number of apps use common words in the name.

For example a tweet containing "make it rain" could be talking about natural phenomena of rain, or a tweet containing "Monument Valley" could be about the national park.

Monument Valley
(breakout)
common words become app-related

Before Release Date		After Release Date (but before breakout)	
# Tweets	Co-occuring Words	# Tweets	Co-occuring Words
110	rt	76	rt
36	utah	22	ustwogames
29	over	18	arizona
25	earthpictures	15	ios
22	httptcobvmvoxwsfa	13	ustwo
21	orion	13	over
21	light	12	game
20	mars	11	sunrise
19	arizona	10	utah
16	outerspaceporn	10	usa
16	httptcobkx3vwipbd	10	techcrunch

Up Coffee
(non-breakout)

Before Release Date		After Release Date (but before breakout)	
# Tweets	Co-occuring Words	# Tweets	Co-occuring Words
1068	up	256	coffee
1017	coffee	233	up
442	app	83	rt
382	lent	54	heating
328	give	53	coffeedad
233	rt	40	jawbone
212	giving	36	upcoffee
185	caffeine	32	caffeine
163	im	31	give
144	jawbone	29	via
135	jawbones	21	wake

Figure 3: Co-occurring words for Monument Valley and Up Coffee

Feature Engineering

In order to avoid these false positives, we analyze co-occurring words and look for the following to identify app-related tweets.

- App related keywords like ios, games, itunes, scored, addicted etc
- Name of the app publisher
- Reviewers, e.g. polygon, techcrunch etc
- App id in the URL - this is a number which is often used by tweets to identify the app on the app store

Based on the above criteria, we designed 3 features:

1. *weak tweets*: all tweets with the app name

2. *strong tweets*: subset of tweets which *also* have other appstore-related words
3. *co-words scores*: a score indicating change in co-words with popularity

Appstore Reviews

The second part of this stage crawls the App store for reviews. We collected reviews for all the apps which were selected in stage 1.

Based on these reviews. We generated 3 additional features:

1. *Number of reviews:* Count of reviews for the app
2. *Rating volume:* Sum of all ratings for the app
3. *Running average rating:* Average rating over time

Each of these features were stored as a time series, with granularity of one day.

Experiments & Results

Train vs Test Data

Our breakout criteria resulted in the identification of 25 breakout apps and 92 fizzled apps from the apps dataset. All the remaining apps that were released in March or April, 2014 did not show a breakout. With this data, we constructed:

X	CoWordScore	StrongTwtFreq	WeakTwtFreq	RPD	RVPD	AvgRating	app_id	breakout
0	112	117	386	12	41	3.4	839767936	1
1	3	2	4	47	205	4.4	842274575	1
2	649	433	1307	505	2210	4.4	625257520	1
3	650	1170	22621	199	743	3.7	779561331	1
4	57	30	41	224	1010	4.5	824318267	1
5	0	112	6503	10	42	4.2	839599050	1
6	10477	3464	7175	44	173	3.9	840919914	1
7	54	36	40	356	1338	3.8	838500730	1
8	243	325	916	459	1254	2.7	783753727	1
9	5	2	3	8	29	3.6	828952581	0
10	0	0	0	8	39	4.9	829670934	0

Figure 4: A snapshot of the training data subset

Training set
Randomly sampled 16 breakout apps (out of 25 total)
32 non-breakout apps (16 fizzled apps + 16 non-breakout apps)

Test set
9 breakout apps
263 non-breakout (76 fizzled apps,187 non-breakout apps)

The size of the non-breakout apps in our test set was limited by the number of apps that we could crawl ratings data for. In our experiments, we observed that if we increase the number of non-breakout apps in our

test set (as is the case in actual app proportions), precision stays low but recall is still good. More details on this in the following sections.

Baseline and Learning Models:

We considered three obvious approaches to establish baseline yardsticks that could be used to evaluate our models against. These are shown in the table below.

	No-breakout predictor	Random Prediction (Coin-Toss)	Day 1 Rating >= 4.5
Accuracy*	0.97	0.50	0.54
Precision	0	0.035	0.041
Recall	0	0.556	0.556
F1-score	0	0.067	0.076

Table 1: Baselines

No-breakout Predictor for All Apps

This predictor classifies all apps as no-breakout. Accuracy of this model is shown in the table above. The reason for the high accuracy is that overwhelming majority of apps are no-breakouts and only very small fraction of Apps go on to breakout. So, by simply focusing on no-breakouts, i.e true negatives, this naive model obtains a very high accuracy. However, its precision and recall are zero. Such a model will never predict even a single breakout app.

This illustrates that accuracy by itself is not a useful metric for us.

Random Coin-toss Predictor

This classifies apps as breakouts or non breakouts by simply flipping a coin with equal probability for each case. This achieves 50% recall by the nature of it. But its precision is very poor due to the high rate of false positives.

Predictor using Launch Day Rating

This takes the rating on the release day of the App and if it is higher than 4.5 then it predicts the app as breakout. This has very similar precision/recall characteristics as the previous case because around half the apps tend to have high ratings on their release day.

These 3 baselines illustrate that finding breakout is like searching for a needle in a haystack. Probability of classifying an app wrongly as breakout (i.e. False Positives) will be high.

Hence, our objective is to achieve a ***good recall at reasonable precision.***

Evaluation Metrics:

Based on the discussion in previous section, we have selected the following metrics to evaluate our results:

- Precision
- Recall
- F1 measure

Of these, *Recall* is the most critical metric for us, as we want to identify as many of the breakouts as possible, with reasonable precision.

Results

To solve the prediction problem, we chose 3 learning algorithms:

1. Logistic Regression
2. SVM
3. Random Forests

We make predictions at 3 different points in time:

- 10 Days Before Breakout
- At Breakout Day
- 10 Days After Breakout

Feature Selection

Prediction Point	Co-words	Strong-Tweets	Weak-Tweets	#Reviews	Rating Volume	Average Rating
10 Days before Breakout	X	X		X		X
At Breakout			X	X	X	X
10 Days after Breakout			X	X	X	

Table 2: Feature Selection for Logistic Regression

For each learning model, we do a grid search to find the most optimal combination of features, i.e those which yield the best F1 measure. Tables 2 shows the optimal results for our logistic regression training model. From these tables and by examining the coefficients obtained from the training model we make following observations:

- Co-word score and strong tweets frequency are the dominant predictors at 10 days before breakout. But at breakout and 10 days after breakout weak tweets are being preferred by the training models. We don't know the exact reason for this but anecdotal evidence suggests that after breakout, weak tweets are strongly correlated with strong tweets as app related tweets start dominating the set of tweets with App Names.
- Number of reviews takes over as a dominant predictor at and after breakout. Intuitively, popularity is directly proportional to number of reviews and so this makes sense. On the other hand, average rating is negatively weighted by the training models. The reason for this is that a large number of Apps tend to have good average ratings on a very few number of reviews. This was also clear from the baseline predictor which uses launch day rating where we saw a very large number of apps have high ratings on their launch day.

Logistic Regression

Train Data:				
Prediction Point	**Precision**	**Recall**	**F1**	**Accuracy**
10 Days before Breakout	0.93	0.60	0.73	0.85
At Breakout	0.92	**0.92**	0.92	0.94
10 Days after Breakout	0.99	**0.99**	0.99	0.99

Table 3: Train Data Results for Logistic Regression

Test Data:				
Prediction Point	**Precision**	**Recall**	**F1**	**Accuracy**
10 Days before Breakout	0.18	0.44	0.25	0.91
At Breakout	0.27	0.75	0.40	0.93
10 Days after Breakout	0.60	0.83	0.70	0.98

Table 4: Test Data Results for Logistic Regression

Table 4 shows the performance of the Logistic Regression Model on the Test data. Before breakout we predict with modest recall but at a low precision. This precision is still much better than the baselines. Both precision and recall improve significantly as we change the point of prediction to be later.

SVM

Train Data:				
Prediction Point	**Precision**	**Recall**	**F1**	**Accuracy**
10 Days before Breakout	93.1	38.6	54.6	78.6
At Breakout	88.0	85.0	86.5	91.0
10 Days after Breakout	97.1	78.5	87.1	92.1

Table 5: Train Data Results for SVM

Test Data:				
Prediction Point	**Precision**	**Recall**	**F1**	**Accuracy**
10 Days before Breakout	0.21	0.26	0.23	0.94
At Breakout	0.35	0.74	0.48	0.94
10 Days after Breakout	0.56	0.75	0.64	0.97

Table 6: Train Data Results for SVM

Table 6 shows the performance of the SVM Model on the Test data. This model gives better precision at the cost of recall.

Random Forests

Train Data:				
Prediction Point	**Precision**	**Recall**	**F1**	**Accuracy**
10 Days before Breakout	0.99	0.83	0.88	0.94
At Breakout	1.00	1.00	1.00	1.00
10 Days after Breakout	1.00	1.00	1.00	1.00

Table 7: Train Data Results for Random Forests

Test Data:				
Prediction Point	**Precision**	**Recall**	**F1**	**Accuracy**
10 Days before Breakout	0.09	0.47	0.16	0.84
At Breakout	0.35	0.82	0.49	0.94
10 Days after Breakout	0.50	0.90	0.64	0.96

Table 8: Train Data Results for Random Forests

Table 8 shows the performance of the random forests Model on the Test data. This model trades recall for precision. This model gives us the best recall and precision for prediction at breakout.

We also tried following experiments:

- Increased the training data to have more fizzled and no-breakout apps. Our recall was slightly worse in this case.
- Restricted the data set to categories which have high average number of reviews. In this case we did not see a significant difference in the results compared to data set without the category restriction.

Comparison of Model Performance:

	Linear Models	**SVM**	**Random Forests**
Recall	75.3%	74%	**88.8%**
Accuracy	90.4%	93.8%	91.5%
Pros/Cons	+ Interpretable + Easy to implement - Low Recall	- Low interpretability - Slower than linear - Low Recall	- Low interpretability - slower than linear methods + **Highest Recall**

Table 9: Comparison of Results

Random Forests yield the highest recall, thereby maximizing the number of breakouts identified early. Actual training sets are not huge (the process of getting the features, e.g. tweets and reviews is). Further, performance speed for prediction is not critical. Hence we recommend using random forests for this problem.

Future Work

1. In our analysis we realized that most of our data with the exception of tweets do not have leading indication of breakout. The natural question is how do people learn about a new app and what triggers its rise in popularity. We investigated this and found that there are several triggers. Apps are sometimes adopted from other gaming platforms or Android. Also a large number of apps are clones and by improving on user experience and they end up becoming more popular than their originals. Game reviewers like polygon and edge play a strong role as influencers. By further examining these sources one could come up with a model which would predict breakouts earlier. We could also experiment crawling other signals like reddit apps, or Google Trends (currently no API available)
2. More granular rank and downloads data from commercial sources like http://appannie.com might be useful for further analysis of rank curves and application of methods like image analysis or clustering based on rank trajectories.
3. Large duration data could be used to augment our model and test it further

Conclusion

We built a model to predict breakouts with 89% recall using data from Twitter, App Ranks, and (crawled) app review data. We also designed several reviews-related features and identified co-occurring words as signal. We also observed the following:

- Tweets are useful as leading indicator while number of reviews/ratings indicate persistence of popularity.
- Majority of breakout apps are games, and games breakout quickly after being released
- Several games apps which go on to become very popular are clones of others (e.g. 2048 is a clone of ‘threes’, piano tiles is a clone of don’t step on white tile) and they go on to become far more popular than the originals

Acknowledgements

We are grateful to Prof. Anand Rajaraman for his valuable mentorship, guidance and suggestions. Several of the ideas here were motivated by him. We would also like to thank Mr. Sailesh Ramakrishnan for getting us the data and useful feedback on our progress. Finally, we thank Mike Chrzanowski for helping us get set up with AWS, access data, and being a constant support for us throughout the course.