Foursquare recommendation system

Creation process:

- Started by exploring popular (but « popular » is hard to define) data, including time sensibility
- Lot of tests, everyday
- Personalization: collaborative filtering (personalization based on your checking history + "people like you"). 10M venues => compute similarity is 100trillion computation. => Hadoop cluster as part of Hive + Mahout package on top. + Need to decide constraints to say if a result is significant => Similarity matrix in less than an hour on 40 machines cluster.

"Cold start" = not enough data for a user to provide personal results. In that case: can sort places per pure check-in count or moving average. => Find interesting places but not specific hidden ones for the user.

How to feed the ranking?

Interesting data: number of users and number of unique users.

+ very important to tell the user WHY they should go to this place (which friends liked it, etc).

+ remember a place you haven't been for a while

Once retrieval, ranking and rendering is done, data queries can be done in 100ms.

Tricky queries:

- Querying initial corpus to rank: retrieve a set of venues within the radius you specify as candidate recommendations. => Requires geographically indexing. MongoDB can do that.
- Where have your friends been: pulling back friends history. Requires special cache on top of check-ins aggregating data about interaction between users and venues.

Foursquare's data and the Explore recommendation engine

Data model: Users (> 10 million), venues (> 15 million), check-ins (> 750 million, > 3 million/day).

"Explore" = social recommendation engine, leverage check-in data.

Real-time recommendations according to: time of the day + previous check-ins (user and friends history): User's history: places similar to where I've been (people that go here also go there…)

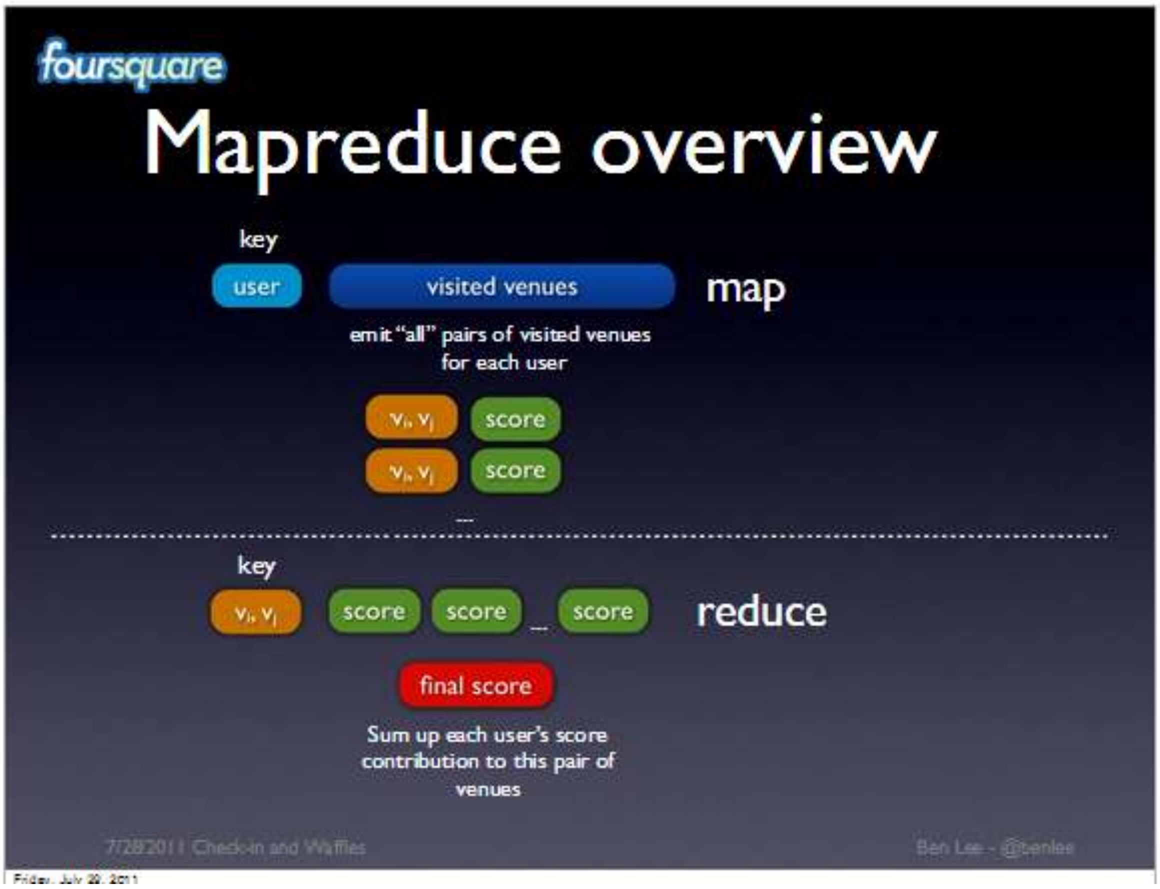
Users x Venues matrix is very sparse.

Venues similarity = similarity of columns.

Users similarity = similarity of lines

Similarity pipeline:

- Input: Mongo dumps on S3 or HDFS
- Java mapreduces (Hadoop/Elastic Mapreduce)
- Output loaded into Mongo.



Deal with nearby relevant venues, friend's check-in history and similarities, user's check-in history, similar venues, MOAR signals in less than 200ms.

Data stack: MongoDB (production), Amazon S3, Elastic Mapreduce, Hadoop, Hive, Flume, R/RStudio.

Machine Learning with Large Networks of People and Places

Why people use foursquare:

- Share with friends
- Discover new places
- Get tips
- Get deals
- Earn points and badges
- Keep track of visits

Place graph:

30M places interconnected with different signals (flow, co-visitation, categories, menus, tips and shouts).

People behaviour connect places: this user went there after going there. + can also use categories (after “coffee shops” people go to “office”)

Collaborative filtering: How to connect people to places they’ll like?

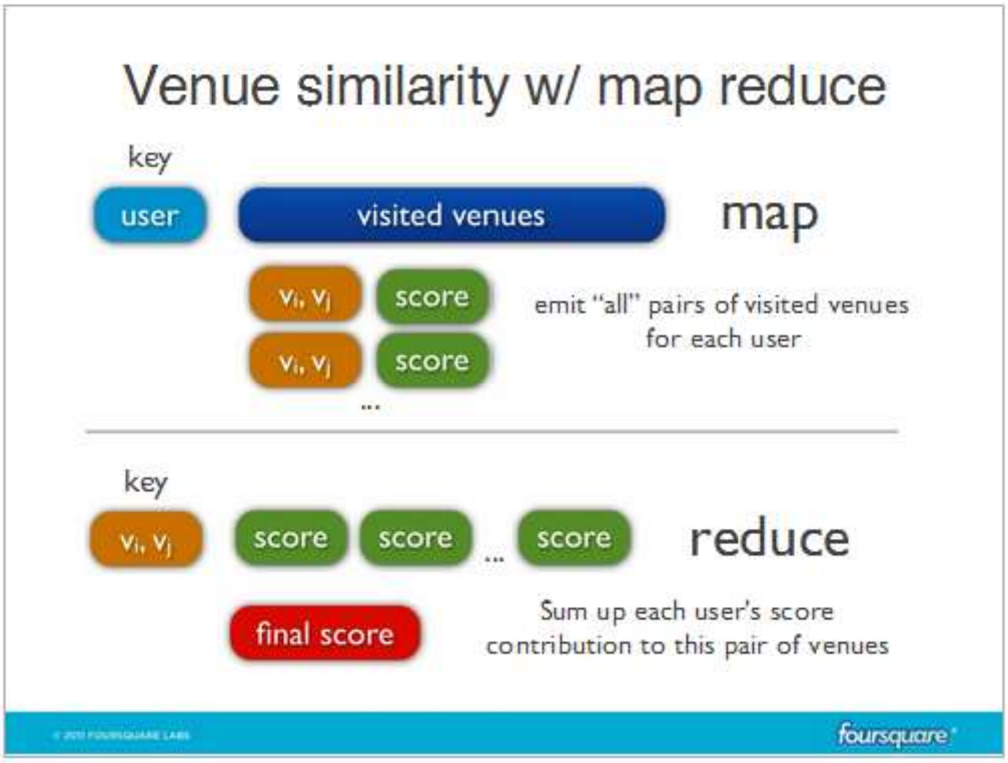
[Koren, Bell ‘08]

- Item-Item similarity: items similar to what user already liked
    - ☺ Can easily update with new data for a user + explainable
    - ☹ Not as performant as richer global models
- User-User similarity: items from similar users
    - ☺ Can leverage social signals as well
- Low-rank matrix factorization: find low-dimensional space of users and items and match the nearest items (in this place) to the user.

Finding similar items:

= Large sparse k-nearest neighbour problem, with places, people, or brands as objects and different distance metrics.

Object	Metrics
Places	Cosine similarity: $sim(x_i, x_j) = \frac{x_i x_j}{\|\|x_i\|\|\|\|x_j\|\|}$
Friends	Intersection: $sim(A, B) = \|A \cap B\|$
Brands	Jaccard similarity: $sim(A, B) = \frac{\|A \cap B\|}{\|A \cup B\|}$


Venue similarity w/ map reduce
key
user
visited venues
map
vᵢ, vⱼ
score
vᵢ, vⱼ
score
...
emit “all” pairs of visited venues for each user
key
vᵢ, vⱼ
score
score
...
score
reduce
final score
Sum up each user’s score contribution to this pair of venues
© 2011 FOURSQUARE LABS
foursquare

(same thing as above ☺)

Social graph:

15m person social network.

Interaction types: friends, follows, dones, comments, colocation

Graph embedding: example: spectral embedding (decompose adjacency matrix with a SVD and use eigenvectors with highest eigenvalues for coordinates)

We want to do embedding such as from the nodes we are able to recover the edges and thus the matrix A. (See Structure Preserving Embedding)

Measuring influence with PageRank

Iterative: start with random value and iterate (+ use MapReduces).

$$PR(i) = (1-d) + d \sum_{j \in \{A_{ij}=1\}} \frac{PR(j)}{\sum_k A_{ik}}$$

Equivalent to finding the principal eigenvector of the normalized adjacent matrix.