Master Thesis

Crowdsourced Product Descriptions and Price Estimations

Steve Aschwanden
Dammstrasse 4
CH-2540 Grenchen
steve.aschwanden@students.unibe.ch
05-480-686

Supervisor
Dr. Gianluca Demartini
C312, Bd de Pérolles 90
CH-1700 Fribourg
demartini@exascale.info

Grenchen, March 3, 2014

Declaration

I declare that this written submission represents my ideas in my own words and where others' ideas or words have been included, I have adequately cited and referenced the original sources. I also declare that I have adhered to all the principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. I understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

Steve Aschwanden, 05-480-686

Grenchen; March 3, 2014:

_____

(Signature)

Acknowledgements

I like to acknowledge ...

Abstract

I like to acknowledge ...

Contents

List of Figures

List of Tables

Listings

Chapter 1

Introduction

eBay Inc.[1] is one of the world's largest online marketplaces and reported 128 million active users worldwide during the last quarter of the year 2013. Online auction platforms make consumer-to-consumer transactions possible. The seller can present articles by uploading pictures and describing them. The creation of an auction is time consuming and needs a lot of investigations. Searching for descriptions on the internet or finding a selling price for the same or similar article, for example. In 2005, Jeff Howe and Mark Robinson created a term called 'Crowdsourcing' which is a combination of the words crowd and outsourcing. The idea behind the term is to outsource different tasks, which are difficult to solve by machines, to the crowd. To reduce the costs of collecting information for an article to sell on an auction platform, tasks will be created and outsourced to the crowd. Amazon Mechanical Turk[2], short MTurk, is a crowdsourcing marketplace which enables requesters to publish human intelligence tasks (HITs). The workers can solve these tasks and earn money for good work.

1.1 Statement of the problem

The first step of creating an online auction is mostly to take pictures of the corresponding item. This help the buyers to get information about the state and quality of the article. After that the item needs a short and clear description, some properties (category, state) and a starting offer. If the seller wants to create a lot of different auctions, the whole procedure is time consuming and boring. A price estimation of an article can be difficult, because the background knowledge is missing and other auctions to compare aren't available at any time. Machines aren't able to solve all these steps by them self, because the spectrum of the articles is huge and image processing methods aren't capable to classify them all correctly. To get all the needed parts of an online auction, a human powered approach is necessary. Crowdsourcing platforms provide the possibility to solve tasks, which are difficult to handle for a computer.

1.2 Existing research

tbd

1.3 Goals and objectives

The thesis has the following goals and their corresponding objectives:

- **Collect auction item properties by the crowd**
  - Analyse the composition of an auction item on eBay and select the parts which can be crowdsourced

[1] http://www.ebay.com
[2] http://www.mturk.com

  - Form a ground truth including different auctions created by real online auction platform users by using the eBay API
  - Study literature which covers similar crowdsourcing problems
  - Design and publish tasks on Amazon Mechanical Turk to gather data from the crowd
  - Evaluate the quality of the generated content
- **Try to improve the initial solution by implementing a hybrid approach**
  - Search for image processing or machine learning methods which can simplify and/or support a human intelligence task
  - Implement the methods and adapt the design of the tasks
  - Publish the new tasks on the same crowdsourcing platform
  - Evaluate the results and compare them to the first solution

If the main goals of the thesis are fulfilled, some *optional* goals can be covered by the thesis:

- **Implement a web application which combines the created subtasks to a complete workflow**
  - Find a web application framework which provide an API in the same programming language as the Amazon Mechanical Turk API
  - Create a workflow which put all the subtasks together to an overall solution
  - The user can manage the items (upload pictures to create new items, edit and remove items) and directly create an online auction

1.4 Organization

The thesis is splited into several chapters:

- eBay online marketplace
- Crowdsourcing
- Evaluation
- Conclusion

Chapter 2

eBay online marketplace

Contents

2.1 History

eBay was founded 1995 in San Jose (CA) as AuctionWeb by Pierre Omidyar. One year later, eBay bought a third-party licence from Electronic Travel Auction to sell plane tickets and other travelling stuff. During the year 1996, over 200'000 auctions were available on the website. At the beginning of 1997 the number of auctions exploded (about 2 million articles). In the same year the company get their well-known name eBay and received 6.7 million dollar from the venture capital firm Benchmark Capital. The company went public on the stock exchange on September 21, 1998 and the share price increased from 18 to 53.5 dollar on the first day of trading. Four years later the growth continues and eBay bought the online money transfer service PayPal. eBay expanded worldwide in early 2008, had hundred millions of registered users and 15'000 employees. Today, the firm is one of the world's largest online marketplaces. During the fourth quarter of the year 2013 about 128 million active users were reported. A cell phone was sold every 4 seconds, a pair of shoes every 2 seconds and a Ford Mustang every 55 minutes.

2.2 Auction item composition

Every eBay user has the possibility to create auctions for different kind of items. To present the article, the seller has to provide accurate information about it. The standard eBay auction consists of the following fields:

- **Title** The title of the item is limited to 80 characters. The sellers should use descriptive keywords to clearly and accurately convey what they are selling
- **Description** The description is the opportunity to provide the buyers with more information about the item
- **Category** An item can have multiple predefined categories. eBay provides a list of categories which the seller can select
- **Condition** The condition of the item is dependent on the selected category. eBay provides different condition schemas. For clothing items the seller can select between 'New with tags', 'New without tags', 'New with defects' or 'Pre-owned'. For other categories like books, other condition values are present: 'Brand new', 'Like new', 'Very good', 'Good', 'Acceptable'
- **Pictures** To visualise the item the auction creator can upload up to twelve pictures. The first image is important, because it appears next to the item's title in the search result. The pictures will be stored for 90 days on the eBay servers.
- **Shipping costs** The seller has to tell the future buyers how much shipping will cost. There are three possibilities:
  - Free shipping
  - Flat shipping, same cost to all buyers
  - Shipping rate tables, eBay calculates the cost for every individual buyer dependent on the location
- **Duration** An auction can have a duration of 1, 3, 5, 7 or 10 days. If the item has a fixed price, the auction is finished if a buyer is willing to pay this price.
- **Pricing** The seller can select a starting price and then the bidding will start at this price. A 'But it now' option is also available. The buyer can skip the bidding process.
- **Payment** The seller has to select the desired paying method like 'PayPal' or 'Payment upon pickup'

2.3 APIs

eBay provides multiple APIs for developing third party applications. This allows developers to search for auctions or create listings over the XML format. Three main interfaces are available:

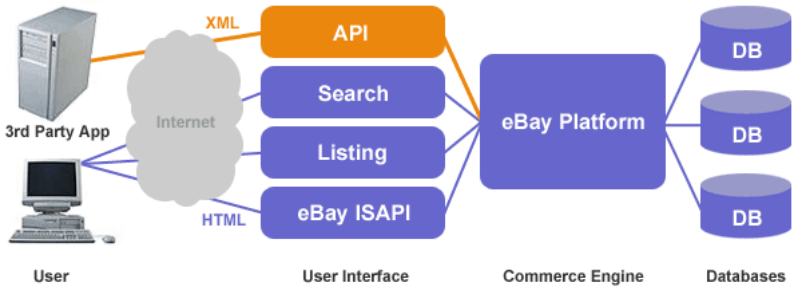

Figure 2.1: eBay API overview

2.3.1 Trading API

Developers use the Trading API to build applications such as selling and post-sales management applications, manage user information, and initiate the item purchase flow on eBay. The API is available in .NET, Java, PHP and Python.

2.3.2 Shopping API

The Shopping API provides a search engine for user information, popular items and reviews. The API is available in PHP and Python. Example calls for this API are:

- *findProducts()*: Search for products by keywords or ProductId
- *GetSingleItem()*: Buyer specific view of an item
- *GetUserProfile()*: Get the user profile and feedback information

2.3.3 Finding API

The Finding API provides access to the next generation search capabilities on the eBay platform. The developer can search and browse for items based on keyword queries, categories or an image. The API is available in .NET, Java and Python. Example calls for this API are:

- *findCompletedItems()*: Find the items which are listened as completed or no longer available on eBay
- *findItemsByCategory()*: Find items in a specific category
- *findItemsByImage()*: Find items which have a high similarity to a given image

2.3.4 Example

The following listing in Python illustrate the functionality of the Finding API. The developer has to register to the eBay developers program first. After that, a application ID can be created. This is necessary to get access to the eBay databases. A functioning Python environment and the additional eBay Python SDK are requirements to successfully execute the example

```
1  from ebaysdk.finding import Connection as Finding
2  from ebaysdk.exception import ConnectionError
3  import json
4
5  try:
6      api = Finding(appid='Universi-3c25-4b4e-b3e6-8c2568808b12')
7      api.execute('findCompletedItems', {
8          'keywords': 'ford mustang',
9          'itemFilter': [
10             {'name': 'ListingType',
11              'value': 'Auction'},
12             {'name': 'Currency',
13              'value': 'USD'},
14             {'name': 'SoldItemsOnly',
15              'value': 'true'},
16         ],
17         'sortOrder': 'StartTimeNewest',
18         })
19     response = json.loads(api.response_json())
20
21     print response['searchResult']['item'][0]
22
23 except ConnectionError as e:
24     raise e
```

Listing 2.1: eBay Finding API example

The initialisation of the application is done on line 6. A correct application ID is required. Then the API call *findCompletedItems()* is executed with some keywords and filter options. Only the newest auctions with at least one bidder and a payment in US dollar will be returned. The function *response_json()* (on line 19) returns the first 100 items by default. At the end, the first result will be printed to the console. Here is a shorter simplified version with the most important fields of the output:

Name	**Value**
itemId	281273507096
title	2014 Hot Wheels Super Treasure Hunt 71 Mustang Mach 1
categoryName	Diecast-Modern Manufacture
shippingType	Calculated
currentPrice	18.5 USD
bidCount	1
paymentMethod	PayPal
conditionDisplayName	New
startTime	2014-02-25T04:32:17.000Z
endTime	2014-02-25T05:27:14.000Z

Table 2.1: eBay Finding API example output

Chapter 3

Crowdsourcing

Contents

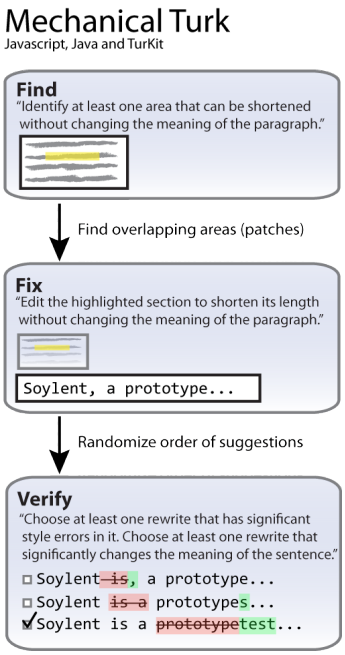

Figure 3.1: Soylent Fix-Find-Verify pattern

3.1 Introduction

3.2 Patterns

3.2.1 Find-Fix-Verify

The Find-Fix-Verify pattern was introduced by the Soylent paper [1]. The pattern divide the overall task into three stages. During the Find stage, the workers will identify patches of work done by the crowd or create new patches. For example, the workers has to select a sentence which seems to be incorrect and will need further investigations during the Fix phase. Some workers will revise the identified patches and try to provide some alternatives. The last step of the pattern will present the generated alternatives during the Fix stage to a few new workers in a randomize order. The answer with the most votes (plurality voting) will be used to replace the identified patch during the first phase. The creators of the new suggestions will be suspended so that they can't vote for their own input. To illustrate the meaning of the Find-Fix-Verify pattern, the implementation of Soylent will be discussed (Figure 3.1). The approach begins by splitting a text into paragraphs. During the Find stage, the workers has to identify candidate areas for shortening in each paragraph. If a certain number of workers has selected the same area then this patch goes to the next stage. Every worker in the Fix stage has to present a shorter version of the identified patch if possible. He has also the possibility to say that the text can't be reduced. During the last step, the crowd has to select rewrites which has significant spelling, style or grammar problems or change the meaning of the sentence significantly. At the end they remove these patches by majority voting.

3.2.2 Iterative

Most of the published assignments on MTurk are independent, parallel tasks. But also iterative, sequential tasks can be useful. The authors of the TurKit paper [5] implemented a tool which make iterative tasks possible. They developed an example application for creating an image description (Figure 3.2). During the first iteration, the worker will contribute the initial description of the provided image. The next iteration will show the initial description and a request to improve it.

Iteration 1: Lightening strike in a blue sky near a tree and a building.
Iteration 2: The image depicts a strike of fork lightening, striking a blue sky over a silhoutted building and trees. (4/5 votes)
Iteration 3: The image depicts a strike of fork lightning, against a blue sky with a few white clouds over a silhouetted building and trees. (5/5 votes)
Iteration 4: ~~The image depicts a strike of fork lightning, against a blue sky wonderful capture of the nature.~~ (1/5 votes)
Iteration 5: This image shows a large white strike of lightning coming down from a blue sky with the tops of the trees and rooftop peaking from the bottom. (3/5 votes)
Iteration 6: This image shows a large white strike of lightning coming down from a blue sky with the silhouettes of tops of the trees and rooftop peeking from the bottom. The sky is a dark blue and the lightening is a contrasting bright white. The lightening has many arms of electricity coming off of it. (4/5 votes)

Figure 3.2: Iterative image description created by TurKit

A few workers will evaluate the extension of the description by voting. If the extended description doesn't receive enough votes then the iteration will be ignored. The final description is generated after a fixed number of iterations. To make the iterative solution possible, the crash-an-rerun programming model was introduced by the authors of the paper. This model allows a script to be re-executed after a crash without generating costly side-effects. That means, if there is a crash during the second iteration of an iterative problem the first iteration will be skipped after re-running the script. TurKit is able to persist the state of the program and will never repeat the successfully completed task. This is helpful for prototyping algorithms.

3.3 Design

MTurk best practices, Iteration, Very important, Instructions are the key

3.4 Hybrid

3.5 Quality control

3.5.1 Honey pots

3.6 Workflow

A workflow is a set of tasks which are interconnected and easier to solve by crowd. The output of a single subtask will be used for one or multiple subsequent subtask. The output of the last element of the flow is the result of the entire complex task. It exists a lot of literature which covers the problematic of finding and interconnect subtask:

The process of decomposing complex tasks into simpler ones is not always easy and need a lot of clarifications. The developers of the Turkomatic [4] tool had an obvious idea and source the workflow decomposition out to the crowd. The workers should decide how the final workflow should look like and what are the belonging tasks. The system consists of two major parts. The meta-workflow is used to design and execute workflows by applying the price-divide-solve (PDS)

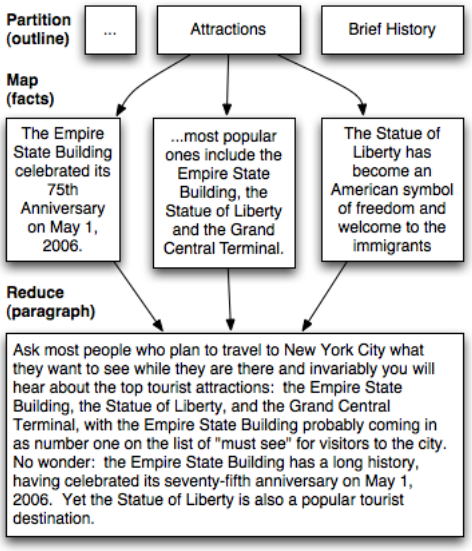

Figure 3.3: CrowdForge example workflow

procedure. The workers has to recursively divide the complex task into smaller ones until they are simple enough. After this step the workers will solve the generated tasks and other workers are asked to check the solutions. At the end, the results are combined into a cohesive answer. The second part of the Turkomatic system allows a visualisation of the created workflows and an edit function to manually adapt the crowdsourced results.

Another idea pursues the developers of CrowdForge [3]. They designed a framework to create a workflow by using several partition, map and reduce steps. The partition step split a larger task into smaller subtasks, the map step let one or more workers process a specified task. The results of the workers are merged into a single output during the reduce step. For example, the workers should write an encyclopaedia article about a given topic (Figure 3.3). The authors of the paper solved this problem by the presented partition/map/reduce steps. First, the partition step asks the workers to create an outline of the article by defining section headings (e.g. "History", "Geography"). During the map phase, multiple workers are asked to provide a single fact about the section (e.g. "The Empire State Building celebrated its 75th Anniversary on May 1, 2006" if it's an encyclopaedia article about "New York" and the section heading is "Attractions"). The workers has to piece the collected facts together to a completed paragraph during the reduction step.
The CrowdForge prototype is written in Python using the Django web framework and Boto, an interface to the Amazon Web Services which is available in Python. The user can define complex flows by creating HIT templates (which can be either a partition, map or reduce task) and dependencies between the templates to define a flow. Flows are implemented as Python classes. The prototype is also responsible for the sequential coordination between the HITs (including data transfer). Multiple independent flows can be executed simultaneously. One of the limitations is that CrowdForge does not support iteration or recursion. The further development of the project was suspended in 2011.

The same crew developed CrowdWeaver [2] which is an advancement of the CrowdForge project. They use CrowdFlower, an other crowdsourcing platform, instead of Amazon Mechanical Turk. On CrowdFlower, the requester can create tasks on multiple markets (including MTurk). Flows can be created visually and doesn't assume any programming skills. Another feature is the tracking and notification of crowd factors, for example latency or price.

Jabberwocky

3.7 Incentives

3.7.1 Game

The ESP game [6] makes the labelling of any kind of images on the web possible. There are no guidelines to provide images and no computer vision method exists which can handle the diversity of all images. Search engines are dependent on accurate image descriptions to represent relevant results. Therefore, another approach was introduced by the article. A online, web-based game was developed to attract workers. Two players are randomly assigned to label the same image simultaneously. There is no possibility to communicate with the game partner. Now, every player has to guess the description of the image independently without using the 'Taboo words'. These words are evaluated by a prior round and will be ignored for the actual turn. If there is a match between both players, the score will be increased and another image description was detected. The discovered word will only be taken as a valid description and 'Taboo word' if a predefined number of players had the same agreement. The duration of the whole game is 150 seconds and both parties can guess as many images as possible within this time. During a period of four months, the game was played by 13'630 people and 1'271'451 labels for 293'760 images were generated. These numbers show the power of the idea. The players (crowd) didn't know what's going on behind the scenes and they also didn't realise the purpose of their inputs.

Money, Financial Incentives and the "Performance of Crowds"

Chapter 4

Implementation

Contents

4.1 Tasks workflow

The macro task of generating an auction on eBay is splited into four simpler micro task:

- Generate a title for the auction item based on the provided images of the item
- Generate the description of the item
- Find the category of auction item
- Define a starting price for the auction

[Create image of workflow]

4.2 Task design

4.2.1 Price estimation

Another idea to estimate the starting price is inspired by a German TV game show. The candidate has to predict the cost of an article. After the first guess, the game master answers with 'higher' or 'lower' until the right guess occur or the time is running out. If the player finds out the correct price then she/he will win the object. The idea of the show is modified to implement a game with a purpose, similar to the ESP game project [6]. The general procedure of the game is the following:

1. The system waits until two independent players are connected and ready to play
2. A few pictures, title and description of the article are displayed and the players had to read them first
3. Then the game starts and a first guess of the price will be shown by the system
4. Both users have to decide if the real price is higher or lower than the displayed one
5. Dependent on the previous response, the system will present a higher or lower price until the countdown is expired or there are no guesses left
6. The players will receive a score dependent on the difference of the price estimation. A smaller difference leads to a higher score, a higher one to a lower score

The first guess of the system will be the mean value μ of a large number of sold items on eBay. The value can be determined by the eBay API. The guessing structure will be implemented as a directed binary tree. The root node represents the mean value and every following child node will have a lower (left child) v_l or higher (right child) value v_r, determined by the value of the parent node v_p and the depth d of the tree. The following formula calculates the values of the nodes:

$$v_l(v_p, d) = v_p - \frac{\mu}{2^d} \tag{4.1}$$

$$v_r(v_p, d) = v_p + \frac{\mu}{2^d} \tag{4.2}$$

The leafs are integer values which can't be divided by two and represents the final guess of a player. If the time is up and the guesser doesn't reach a leaf node, the value of the actual node is taken. The score of the price prediction is determined by a scoring function s, where x_1 and x_2 are the price estimations of player 1 and 2.

$$s(x_1, x_2) = 1 - |\varphi(x_1) - \varphi(x_2)| \tag{4.3}$$

The function φ is responsible to normalise the estimations (interval from 0 to 1).

$$\varphi(x) = \frac{x}{2\mu} \tag{4.4}$$

The function is also used to weight the different estimations for the same product. If n rounds are played for a given object, the final price t is calculated:

$$t = \frac{1}{\sum_{k=1}^{n} s(x_{k1}, x_{k2})} \left(\sum_{i=1}^{n} s(x_{i1}, x_{i2}) \frac{x_{i1} + x_{i2}}{2} \right) \tag{4.5}$$

The reliability r of the price estimation is the mean score of all played games for the same object:

$$r = \frac{1}{n} \left(\sum_{i=1}^{n} s(x_{i1}, x_{i2}) \right) \tag{4.6}$$

Bibliography

[1] Michael S. Bernstein/ Greg Little/ Robert C. Miller/ Björn Hartmann/ Mark S. Ackerman/ David R. Karger/ David Crowell/ and Katrina Panovich. *Soylent: A Word Processor with a Crowd Inside*. In *Proceedings of the 23Nd Annual ACM Symposium on User Interface Software and Technology*, UIST '10, pages 313–322. ACM, New York, NY, USA (2010). ISBN 978-1-4503-0271-5. doi:10.1145/1866029.1866078. URL `http://doi.acm.org/10.1145/1866029.1866078`.

[2] Aniket Kittur/ Susheel Khamkar/ Paul André/ and Robert Kraut. *CrowdWeaver: Visually Managing Complex Crowd Work*. In *Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work*, CSCW '12, pages 1033–1036. ACM, New York, NY, USA (2012). ISBN 978-1-4503-1086-4. doi:10.1145/2145204.2145357. URL `http://doi.acm.org/10.1145/2145204.2145357`.

[3] Aniket Kittur/ Boris Smus/ Susheel Khamkar/ and Robert E. Kraut. *CrowdForge: Crowdsourcing Complex Work*. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*, UIST '11, pages 43–52. ACM, New York, NY, USA (2011). ISBN 978-1-4503-0716-1. doi:10.1145/2047196.2047202. URL `http://doi.acm.org/10.1145/2047196.2047202`.

[4] Anand Kulkarni/ Matthew Can/ and Björn Hartmann. *Collaboratively Crowdsourcing Workflows with Turkomatic*. In *Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work*, CSCW '12, pages 1003–1012. ACM, New York, NY, USA (2012). ISBN 978-1-4503-1086-4. doi:10.1145/2145204.2145354. URL `http://doi.acm.org/10.1145/2145204.2145354`.

[5] Greg Little/ Lydia B. Chilton/ Max Goldman/ and Robert C. Miller. *TurKit: Human Computation Algorithms on Mechanical Turk*. In *Proceedings of the 23Nd Annual ACM Symposium on User Interface Software and Technology*, UIST '10, pages 57–66. ACM, New York, NY, USA (2010). ISBN 978-1-4503-0271-5. doi:10.1145/1866029.1866040. URL `http://doi.acm.org/10.1145/1866029.1866040`.

[6] Luis von Ahn and Laura Dabbish. *Labeling Images with a Computer Game*. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '04, pages 319–326. ACM, New York, NY, USA (2004). ISBN 1-58113-702-8. doi:10.1145/985692.985733. URL `http://doi.acm.org/10.1145/985692.985733`.

Appendix A

Some Appendix

A.1 README

```
Fuzzily classify twitter messages using storm and store to cassandra
===


Setup Cassandra (on ubuntu):
---
1. Make sure oracle JDK is installed (1.6+): https://help.ubuntu.com/community/Java#Oracle_Java_7
2. Add the DataStax repository key to your aptitude trusted keys.
> $ curl -L http://debian.datastax.com/debian/repo_key | sudo apt-key add -
3. Install Cassandra:
> sudo apt-get update && sudo apt-get install cassandra
4. Create keyspace and tables:
> cqlsh
> run commands from src/main/resources/createDatabase.txt

Build Runnable jar
---
1. Open a terminal window, navigate to pom.xml directory (project root)
2. Execute the following command:
> mvn clean compile assembly:single
3. In target/, a runnable jar tsfc.jar is created

Run Program
---
> java -jar tsfc.jar <<comma separated list of topics to watch (without whitespace)>>
```