# SimClusters: Community-based Representations for Heterogeneous Recommendations at Twitter

## Overview

SimClusters is as a general-purpose representation layer based on overlapping communities into which users as well as heterogeneous content can be captured as sparse, interpretable vectors to support a multitude of recommendation tasks.

We build our user and tweet SimClusters embeddings based on the inferred communities, and the representations power our personalized tweet recommendation via our online serving service SimClusters ANN.

For more details, please read our paper that was published in KDD'2020 Applied Data Science Track: https://www.kdd.org/kdd2020/accepted-papers/view/simclusters-community-based-representations-for-heterogeneous-recommendatio

## Brief introduction to Simclusters Algorithm

### Follow relationships as a bipartite graph

Follow relationships on Twitter are perhaps most naturally thought of as directed graph, where each node is a user and each edge represents a Follow. Edges are directed in that User 1 can follow User 2, User 2 can follow User 1 or both User 1 and User 2 can follow each other.

This directed graph can be also viewed as a bipartite graph, where nodes are grouped into two sets, Producers and Consumers. In this bipartite graph, Producers are the users who are Followed and Consumers are the Followees. Below is a toy example of a follow graph for four users:

<img src="images/bipartite\_graph.png" width = "400px">

> Figure 1 - Left panel: A directed follow graph; Right panel: A bipartite graph representation of the directed graph

### Community Detection - Known For

The bipartite follow graph can be used to identify groups of Producers who have similar followers, or who are "Known For" a topic. Specifically, the bipartite follow graph can also be represented as an \*m x n\* matrix (\*A\*), where consumers are presented as \*u\* and producers are represented as \*v\*.

Producer-producer similarity is computed as the cosine similarity between users who follow each producer. The resulting cosine similarity values can be used to construct a producer-producer similarity graph, where the nodes are producers and edges are weighted by the corresponding cosine similarity value. Noise removal is performed, such that edges with weights below a specified threshold are deleted from the graph.

After noise removal has been completed, Metropolis-Hastings sampling-based community detection is then run on the Producer-Producer similarity graph to identify a community affiliation for each producer. This algorithm takes in a parameter \*k\* for the number of communities to be detected.

<img src="images/producer\_producer\_similarity.png">

> Figure 2 - Left panel: Matrix representation of the follow graph depicted in Figure 1; Middle panel: Producer-Producer similarity is estimated by calculating the cosine similarity between the users who follow each producer; Right panel: Cosine similarity scores are used to create the Producer-Producer similarity graph. A clustering algorithm is run on the graph to identify groups of Producers with similar followers.

Community affiliation scores are then used to construct an \*n x k\* "Known For" matrix (\*V\*). This matrix is maximally sparse, and each Producer is affiliated with at most one community. In production, the Known For dataset covers the top 20M producers and k ~= 145000. In other words, we discover around 145k communities based on Twitter's user follow graph.

<img src="images/knownfor.png">

> Figure 3 - The clustering algorithm returns community affiliation scores for each producer. These scores are represented in matrix V.

In the example above, Producer 1 is "Known For" community 2, Producer 2 is "Known For" community 1, and so forth.

### Consumer Embeddings - User InterestedIn

An Interested In matrix (\*U\*) can be computed by multiplying the matrix representation of the follow graph (\*A\*) by the Known For matrix (\*V\*):

<img src="images/interestedin.png">

In this toy example, consumer 1 is interested in community 1 only, whereas consumer 3 is interested in all three communities. There is also a noise removal step applied to the Interested In matrix.

We use the InterestedIn embeddings to capture consumer's long-term interest. The InterestedIn embeddings is one of our major source for consumer-based tweet recommendations.

### Producer Embeddings

When computing the Known For matrix, each producer can only be Known For a single community. Although this maximally sparse matrix is useful from a computational perspective, we know that our users tweet about many different topics and may be "Known" in many different communities. Producer embeddings ( \*Ṽ\* ) are used to capture this richer structure of the graph.

To calculate producer embeddings, the cosine similarity is calculated between each Producer’s follow graph and the Interested In vector for each community.

<img src="images/producer\_embeddings.png">

Producer embeddings are used for producer-based tweet recommendations. For example, we can recommend similar tweets based on an account you just followed.

### Entity Embeddings

SimClusters can also be used to generate embeddings for different kind of contents, such as

- Tweets (used for Tweet recommendations)

- Topics (used for TopicFollow)

#### Tweet embeddings

When a tweet is created, its tweet embedding is initialized as an empty vector.

Tweet embeddings are updated each time the tweet is favorited. Specifically, the InterestedIn vector of each user who Fav-ed the tweet is added to the tweet vector.

Since tweet embeddings are updated each time a tweet is favorited, they change over time.

Tweet embeddings are critical for our tweet recommendation tasks. We can calculate tweet similarity and recommend similar tweets to users based on their tweet engagement history.

We have a online Heron job that updates the tweet embeddings in realtime, check out [here](summingbird/README.md) for more.

#### Topic embeddings

Topic embeddings (\*\*R\*\*) are determined by taking the cosine similarity between consumers who are interested in a community and the number of aggregated favorites each consumer has taken on a tweet that has a topic annotation (with some time decay).

<img src="images/topic\_embeddings.png">

## Project Directory Overview

The whole SimClusters project can be understood as 2 main components

- SimClusters Offline Jobs (Scalding / GCP)

- SimClusters Real-time Streaming Jobs

### SimClusters Offline Jobs

\*\*SimClusters Scalding Jobs\*\*

| Jobs | Code | Description |

|---|---|---|

| KnownFor | [simclusters\_v2/scalding/update\_known\_for/UpdateKnownFor20M145K2020.scala](scalding/update\_known\_for/UpdateKnownFor20M145K2020.scala) | The job outputs the KnownFor dataset which stores the relationships between clusterId and producerUserId. </n> KnownFor dataset covers the top 20M followed producers. We use this KnownFor dataset (or so-called clusters) to build all other entity embeddings. |

| InterestedIn Embeddings| [simclusters\_v2/scalding/InterestedInFromKnownFor.scala](scalding/InterestedInFromKnownFor.scala) | This code implements the job for computing users' interestedIn embedding from the KnownFor dataset. </n> We use this dataset for consumer-based tweet recommendations.|

| Producer Embeddings | [simclusters\_v2/scalding/embedding/ProducerEmbeddingsFromInterestedIn.scala](scalding/embedding/ProducerEmbeddingsFromInterestedIn.scala) | The code implements the job for computer producer embeddings, which represents the content user produces. </n> We use this dataset for producer-based tweet recommendations.|

| Semantic Core Entity Embeddings | [simclusters\_v2/scalding/embedding/EntityToSimClustersEmbeddingsJob.scala](scalding/embedding/EntityToSimClustersEmbeddingsJob.scala) | The job computes the semantic core entity embeddings. It outputs datasets that stores the "SemanticCore entityId -> List(clusterId)" and "clusterId -> List(SemanticCore entityId))" relationships.|

| Topic Embeddings | [simclusters\_v2/scalding/embedding/tfg/FavTfgBasedTopicEmbeddings.scala](scalding/embedding/tfg/FavTfgBasedTopicEmbeddings.scala) | Jobs to generate Fav-based Topic-Follow-Graph (TFG) topic embeddings </n> A topic's fav-based TFG embedding is the sum of its followers' fav-based InterestedIn. We use this embedding for topic related recommendations.|

\*\*SimClusters GCP Jobs\*\*

We have a GCP pipeline where we build our SimClusters ANN index via BigQuery. This allows us to do fast iterations and build new embeddings more efficiently compared to Scalding.

All SimClusters related GCP jobs are under [src/scala/com/twitter/simclusters\_v2/scio/bq\_generation](scio/bq\_generation).

| Jobs | Code | Description |

|---|---|---|

| PushOpenBased SimClusters ANN Index | [EngagementEventBasedClusterToTweetIndexGenerationJob.scala](scio/bq\_generation/simclusters\_index\_generation/EngagementEventBasedClusterToTweetIndexGenerationJob.scala) | The job builds a clusterId -> TopTweet index based on user-open engagement history. </n> This SANN source is used for candidate generation for Notifications. |

| VideoViewBased SimClusters Index| [EngagementEventBasedClusterToTweetIndexGenerationJob.scala](scio/bq\_generation/simclusters\_index\_generation/EngagementEventBasedClusterToTweetIndexGenerationJob.scala) | The job builds a clusterId -> TopTweet index based on the user's video view history. </n> This SANN source is used for video recommendation on Home.|

### SimClusters Real-Time Streaming Tweets Jobs

| Jobs | Code | Description |

|---|---|---|

| Tweet Embedding Job | [simclusters\_v2/summingbird/storm/TweetJob.scala](summingbird/storm/TweetJob.scala) | Generate the Tweet embedding and index of tweets for the SimClusters |

| Persistent Tweet Embedding Job| [simclusters\_v2/summingbird/storm/PersistentTweetJob.scala](summingbird/storm/PersistentTweetJob.scala) | Persistent the tweet embeddings from MemCache into Manhattan.|