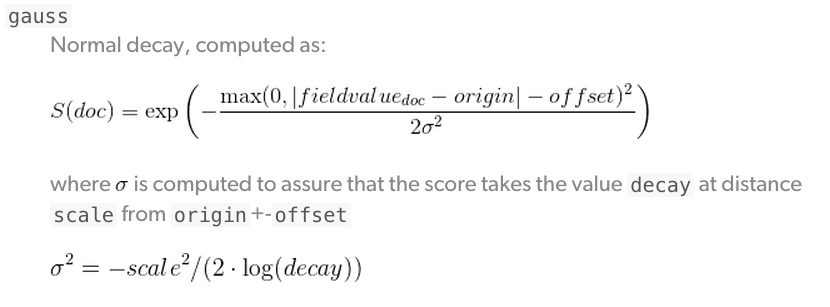
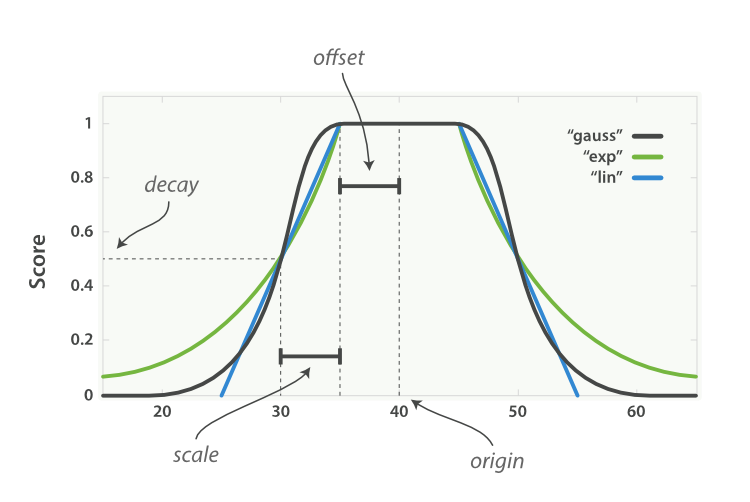
Ranking logics & algorithms

# Decay function

Ranking algorithms are based on decay function (currently Gauss), which reduces the score as fieldvalue moves farther from the origin (it never reaches 0):



* **origin -** the central point, or the best possible value for the field. Documents that fall at the origin will get a full of 1.0.
* **scale -** the rate of decay, how quickly the score should drop the further from the origin that a document lies (for example, every £10 or every 100 meters).
* **decay -** the score that a document at scale distance from the origin should receive.
* **offset -** setting a nonzero offset expands the central point to cover a range of values instead of just the single point specified by the origin. All values in the range -offset <= origin <= +offset will receive the full score of 1.0.



**Example:**

We show with example how Linear and Gauss decay function reduces the score, when videos are published on distinct locations.

We set origin to the user location, offset to 2 km, scale to 100 km and decay to 0.5.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| current user location | how far was video published from the user | Linear decay score | Gauss decay score | Exponential decay score | comment |
| 0 | 0, 1 or 2 km | 1 | 1 | 1 | no reduction of score as offset is set to 2 km |
| 0 | 52 km | 0.75 | 0.84 | 0.71 |  |
| 0 | 102 km | 0.5 | 0.5 | 0.5 | offset + scale = 102 km, function reduces the score to 0.5 (decay value) |
| 0 | 202 km | 0 | 0.06 | 0.25 | all videos that are farther than 204 km, receive 0 score with linear decay function. |
| 0 | 500 km | 0 | 3.4e-08 | 0.03 | Gauss and Exponential never reach 0 score |

# Video feed

For more information see unit test in social-video-community-server -> RankingServiceElasticTest.java.

## New videos around me

new\_score = w1 \* decay(created\_time) + w2 \* decay(location)

new\_score value is on interval = (0, 2]. Weights w1 and w2 define boost factor. Currently, weights are set to 1.

decay(created\_time) gives higher scores to new videos. Score is on interval (0 ,1]. We set origin parameter to current time, offset to 1 minute, scale to 3.5 days and decay to 0.5.  
Example:   
0 - 1 minutes old videos gets the score 1,   
3.5 days old video gets the score 0.5,   
7 days old video gets the score close to 0.1.

20 days old video gets the score close to 0.01.

decay(location) gives higher scores to videos that were published closer to user location. Score is on interval (0 ,1]. Origin parameter is set to current location, offset to 2km, scale to 100 km and decay to 0.5.  
Example:  
Video that was published 1.5 km away from current location gets the score 1,  
if it was published 100km away from current location gets the score 0.5,  
if it was published 200km away from current location gets the score close to 0.1.

## Hot videos around me

hot\_score = w1 \* decay(created\_time) + w2 \* decay(location) + w3 \* trending\_score

hot\_score value is on interval = (0, ~5). Weights w1, w2 and w3 define boost factor. Currently, weights are set to 1.

For decay(created\_time) and decay(location) look at explanation at New videos around me.

### Trending score

trending\_score = 0, if trending\_likes < 0

trending\_score = log10(trending\_likes + 2) \* decay(trending\_time)

decay(trending\_time) is calculated on the same way as decay(created\_time). decay(trending\_time) is on interval [0, 1].

log10(abs(trending\_likes) + 2) is on interval [0.33, ~5].

trending\_score gives higher score to popular videos.

Example:

We change the trending time to video with 10 likes:  
trending\_score(10, now) = 1.08

trending\_score(10, 1 day ago ) = 0.92

trending\_score(10, 4 days ago ) = 0.46

trending\_score(10, 7 days ago ) ~ 0.1

We change the number of likes to video with current trending time:

trending\_score(0, now) = 0.30

trending\_score(10, now) = 1.08

trending\_score(100, now) = 2.01

trending\_score(1000, now) = 3.01

## Top videos worldwide

Ordered by up\_votes - down\_votes for each video, descending.

# Music

For more information see unit test in social-video-community-server -> RankingServiceMusicElasticTest.java

## Trending music around me (MusicSort.top)

Algorithm calculates score for trending videos with music around user location as described in Video feed. It aggregates scores by musicID. If there is not enough results retrieved, algorithm uses backfill to add most used musics and sorts them by usages.

musicID\_score = w1 \* decay(location) + w2\* decay(created\_time) + w3\* trending\_score

## New music around me (MusicSort.latest)

Algorithm calculates score for new videos with music around user location as described in Video feed. It aggregates scores by musicID. If there is not enough results retrieved, algorithm uses backfill to add latest musics and sorts them by music\_created\_time.

musicID\_score = w1 \* decay(location) + w2\* decay(music\_created\_time)

## Uploaded music

List of uploaded music for current user, which is sorted by music creation timestamp, descending.

## Music categories

List of music in specific category sorted by top usage in videos nearby user location.

# HashTags

For more information see unit test in social-video-community-server -> RankingServiceHashTagsTest.java

## Trending HashTags around me

Algorithm calculates score for trending videos with HashTags around user location as described in Video feed and it aggregates scores by HashTags. It also retrieves a video with most trending\_likes for each HashTag, that clients are able to show a video with a HashTag.

HashTag\_score = w1 \* decay(location) + w2\* decay(created\_time) + w3\* trending\_score

# Text below was for previous trending algorithm

# Orders

* **New** - latest videos (ordered by created time, descending),
* **Top** - ordered by most upvotes for video, by most usages for music,
* **Hot** - trending (ordered by trending algorithm)

**NEARBY (around me):**

* **NEW**:
  + content is ordered by timestamp of creation descending
  + content is searched by nearby algorithm, this means that first we search 2.5km around user, than if we still do not get enough content we extend search range to ranges: 5km, 10km, 50km, 300km, 15 000km, 100 000km
    - when extending search range to get enough content it is possible that content won’t be ordered by timestamp of creation descending, because first order is how close is content to user current location
* **HOT**:
  + content is ordered by hot\_score, which is calculated every 3 hours.

hot\_score = decay(created\_time) + decay(location) + trending\_score

* + **decay(created\_time)** gives higher scores to new videos. We set origin parameter to current time, offset to 1 day, scale to 3 days and decay to 0.5.   
    Example:   
    0 day old video gets the score 1,  
    1 day old video gets the score 1 due to offset,   
    4 days old video gets the score 0.5 (decay value) due to offset + scale is 4 days,   
    8 days old video gets the score 0.
  + **decay(location)** gives higher scores to videos that were published closer to user location. Origin parameter is set to current location, offset to 2km, scale to 100 km and decay to 0.5.  
    Example:  
    Video that was published 1.5 km away from current location gets the score 1,  
    if it was published 100km away from current location gets the score 0.5,  
    if it was published 200km away from current location gets the score 0.
  + **trending\_score** gives higher score to popular videos. It is calculated with trending\_likes (up votes - down votes) and trending\_time (the time of the last vote).   
    trending\_score = log10(trending\_likes + 2) \* decay(trending\_time),  
    trending\_score = 0, if trending\_likes < 0,  
    decay(trending\_time) is calculated on the same way as decay(created\_time).

Example:  
 trending\_score(votes=10, yesterday) < trending\_score(votes=10, today)

* **NEW**:
  + content inside one or more communities (based on user membership in communities) is ordered by timestamp of creation descending
* **HOT**:
  + content inside one or more communities (based on user membership in communities) is ordered by trending score (trending score is calculated every 3 hours, might be changed in future)
  + **NOTE**: if content did not receive any upvote it won’t show up in this list or if it received upvotes in last 3 hours it might take up to 3 hours to be seen in this list

# MUSIC LIBRARY

**NEARBY:**

* **TOP**:
  + list of music sorted by top usage in videos nearby user location
  + content is searched by nearby algorithm, this means that first we search 2.5km around user, than if we still do not get enough content we extend search range to ranges: 5km, 10km, 50km, 300km, 15 000km, 100 000km
    - when extending search range to get enough content it is possible that content won’t be ordered by top usages descending, because first order is how close is content to user current location
* **NEW**:
  + list of music sorted by timestamp of music creation used in videos nearby user location
  + content is searched by nearby algorithm, this means that first we search 2.5km around user, than if we still do not get enough content we extend search range to ranges: 5km, 10km, 50km, 300km, 15 000km, 100 000km
    - when extending search range to get enough content it is possible that content won’t be ordered by music timestamp of creation descending, because first order is how close is content to user current location
* **UPLOADED**
  + list of current user logged uploaded music sorted by timestamp of music creation descending
* **CATEGORIES** (multiple categories)
  + list of music in specific category sorted by top usage in videos nearby user location
  + content is searched by nearby algorithm, this means that first we search 2.5km around user, than if we still do not get enough content we extend search range to ranges: 5km, 10km, 50km, 300km, 15 000km, 100 000km
    - when extending search range to get enough content it is possible that content won’t be ordered by top usages descending, because first order is how close is content to user current location

**GLOBAL:**

* **TOP**:
  + list of music sorted by top usage in videos globally
* **NEW**:
  + list of music sorted by timestamp of music creation used in videos globally
* **UPLOADED**
  + list of current user logged uploaded music sorted by timestamp of music creation descending
* **CATEGORIES** (multiple categories)
  + list of music in specific category sorted by top usage in videos globally

# Trending Score Calculations (Not user anymore)

Trending score is defined by the difference between up / down votes in a specified time window (configurable, default is 7 days). Each media record contains latest trending score update timestamp, used to filter out media records that are not up/down voted recently. The trending score also depends on media age ~ older posts are less relevant than the new one (with the same amount of up/down votes).

## Mathematical notation

