# TrueSkill Documentation

***Release 0.4.3***

**Heungsub Lee**

September 24, 2015

Contents

the video game rating system

CHAPTER 1

# What's TrueSkill?

TrueSkill is a rating system among game players. It was developed by Microsoft Research and has been used on Xbox LIVE for ranking and matchmaking service. This system quantifies players' **TRUE** skill points by the Bayesian inference algorithm. It also works well with any type of match rule including N:N team game or free-for-all.

This project is a Python package which implements the TrueSkill rating system:

```
from trueskill import Rating, quality_1vs1, rate_1vs1
alice, bob = Rating(25), Rating(30)  # assign Alice and Bob's ratings
if quality_1vs1(alice, bob) < 0.50:
    print('This match seems to be not so fair')
alice, bob = rate_1vs1(alice, bob)  # update the ratings after the match
```

CHAPTER 2

# Installing

The package is available in PyPI. To install it in your system, use **easy_install**:

```
$ easy_install trueskill
```

Or check out developement version:

```
$ git clone git://github.com/sublee/trueskill.git
```

CHAPTER 3

# Learning

## 3.1 Rating, the model for skill

In TrueSkill, rating is a Gaussian distribution which starts from $\mathcal{N}(25, \frac{25}{3}^2)$. $\mu$ is an average skill of player, and $\sigma$ is a confidence of the guessed rating. A real skill of player is between $\mu \pm 2\sigma$ with 95% confidence.

```
>>> from trueskill import Rating
>>> Rating()  # use the default mu and sigma
trueskill.Rating(mu=25.000, sigma=8.333)
```

If some player's rating is higher $\beta$ than another player's, the player may have about 75.6% of chance to beat the other player. The default value of $\beta$ is $\frac{25}{6}$.

Ratings will approach real skills through few times of the TrueSkill's Bayesian inference algorithm. How many matches TrueSkill needs to estimate real skills? It depends on the game rule. See the below table:

| Rule | Matches |
|---|---|
| 16P free-for-all | 3 |
| 8P free-for-all | 3 |
| 4P free-for-all | 5 |
| 2P free-for-all | 12 |
| 2:2:2:2 | 10 |
| 4:4:4:4 | 20 |
| 4:4 | 46 |
| 8:8 | 91 |

## 3.2 Head-to-head (1 vs. 1) match rule

Most competition games follows 1:1 match rule. If your game does, just use `_1vs1` shortcuts containing *`rate_1vs1()`* and *`quality_1vs1()`*. These are very easy to use.

First of all, we need 2 *`Rating`* objects:

```
>>> r1 = Rating()  # 1P's skill
>>> r2 = Rating()  # 2P's skill
```

Then we can guess match quality which is equivalent with draw probability of this match using *`quality_1vs1()`*:

```
>>> print('{:.1%} chance to draw'.format(quality_1vs1(r1, r2)))
44.7% chance to draw
```

After the game, TrueSkill recalculates their ratings by the game result. For example, if 1P beat 2P:

```
>>> new_r1, new_r2 = rate_1vs1(r1, r2)
>>> print(new_r1)
trueskill.Rating(mu=29.396, sigma=7.171)
>>> print(new_r2)
trueskill.Rating(mu=20.604, sigma=7.171)
```

Mu value follows player's win/draw/lose records. Higher value means higher game skill. And sigma value follows the number of games. Lower value means many game plays and higher rating confidence.

So 1P, a winner's skill grew up from 25 to 29.396 but 2P, a loser's skill shrank to 20.604. And both sigma values became narrow about same magnitude.

Of course, you can also handle a tie game with `drawn=True`:

```
>>> new_r1, new_r2 = rate_1vs1(r1, r2, drawn=True)
>>> print(new_r1)
trueskill.Rating(mu=25.000, sigma=6.458)
>>> print(new_r2)
trueskill.Rating(mu=25.000, sigma=6.458)
```

## 3.3 Other match rules

There are many other match rules such as N:N team match, N:N:N multiple team match, N:M unbalanced match, free-for-all (Player vs. All), and so on. Mostly other rating systems cannot work with them but TrueSkill does. TrueSkill accepts any types of matches.

We should arrange ratings into a group by their team:

```
>>> r1 = Rating()  # 1P's skill
>>> r2 = Rating()  # 2P's skill
>>> r3 = Rating()  # 3P's skill
>>> t1 = [r1]  # Team A contains just 1P
>>> t2 = [r2, r3]  # Team B contains 2P and 3P
```

Then we can calculate the match quality and rate them:

```
>>> print('{:.1%} chance to draw'.format(quality([t1, t2])))
13.5% chance to draw
>>> (new_r1,), (new_r2, new_r3) = rate([t1, t2], ranks=[0, 1])
>>> print(new_r1)
trueskill.Rating(mu=33.731, sigma=7.317)
>>> print(new_r2)
trueskill.Rating(mu=16.269, sigma=7.317)
>>> print(new_r3)
trueskill.Rating(mu=16.269, sigma=7.317)
```

If you want to describe other game results, set the `ranks` argument like the below examples:

- A drawn game – `ranks=[0, 0]`
- Team B won not team A – `ranks=[1, 0]` (Lower rank is better)

Additionally, here are varied patterns of rating groups. All variables which start with `r` are *Rating* objects:

- N:N team match – `[(r1, r2, r3), (r4, r5, r6)]`
- N:N:N multiple team match – `[(r1, r2), (r3, r4), (r5, r6)]`

- N:M unbalanced match – `[(r1,), (r2, r3, r4)]`
- Free-for-all – `[(r1,), (r2,), (r3,), (r4,)]`

## 3.4 Partial play

Let's assume that there are 2 teams which each has 2 players. The game was for a hour but the one of players on the first team entered the game at 30 minutes later.

If some player wasn't present for the entire duration of the game, use the concept of "partial play" by `weights` parameter. The above situation can be described by the following weights:

- 1P on team A – 1.0 = Full time
- 2P on team A – 0.5 = $\frac{30}{60}$ minutes
- 3P on team B – 1.0
- 4P on team B – 1.0

As a code with a 2-dimensional list:

```
# set each weights to 1, 0.5, 1, 1
rate([(r1, r2), (r3, r4)], weights=[(1, 0.5), (1, 1)])
quality([(r1, r2), (r3, r4)], weights=[(1, 0.5), (1, 1)])
```

Or with a dictionary:

```
# set a weight of 2nd player in 1st team to 0.5, otherwise leave as 1
rate([(r1, r2), (r3, r4)], weights={(0, 1): 0.5})
quality([(r1, r2), (r3, r4)], weights={(0, 1): 0.5})
```

## 3.5 Backends

The TrueSkill algorithm uses $\Phi$, the cumulative distribution function; $\phi$, the probability density function; and $\Phi^{-1}$, the inverse cumulative distribution function. But standard mathematics library doesn't provide the functions. Therefore this package implements them.

Meanwhile, there are third-party libraries which implement the functions. You may want to use another implementation because that's more expert. Then set `backend` option of *`TrueSkill`* to the backend you chose:

```
>>> TrueSkill().cdf  # internal implementation
<function cdf at ...>
>>> TrueSkill(backend='mpmath').cdf  # mpmath.ncdf
<bound method MPContext.f_wrapped of <mpmath.ctx_mp.MPContext object at ...>>
```

Here's the list of the available backends:

- `None` – the internal implementation. (Default)
- "mpmath" – requires mpmath installed.
- "scipy" – requires scipy installed.

---

**Note:** When winners have too lower rating than losers, *`TrueSkill.rate()`* will raise `FloatingPointError`. In this case, you need higher floating-point precision. The mpmath library offers flexible floating-point precision. You can solve the problem with mpmath as a backend and higher precision setting.

---

CHAPTER 4

# API

## 4.1 TrueSkill objects

**class** trueskill.**Rating**(*mu=None*, *sigma=None*)
Represents a player's skill as Gaussian distrubution.

The default mu and sigma value follows the global environment's settings. If you don't want to use the global, use *TrueSkill.create_rating()* to create the rating object.

**Parameters**

- **mu** – the mean.
- **sigma** – the standard deviation.

**mu**
A property which returns the mean.

**sigma**
A property which returns the the square root of the variance.

**class** trueskill.**TrueSkill**(*mu=25.0*, *sigma=8.333333333333334*, *beta=4.166666666666667*, *tau=0.08333333333333334*, *draw_probability=0.1*, *backend=None*)
Implements a TrueSkill environment. An environment could have customized constants. Every games have not same design and may need to customize TrueSkill constants.

For example, 60% of matches in your game have finished as draw then you should set `draw_probability` to 0.60:

```
env = TrueSkill(draw_probability=0.60)
```

For more details of the constants, see The Math Behind TrueSkill by Jeff Moser.

**Parameters**

- **mu** – the initial mean of ratings.
- **sigma** – the initial standard deviation of ratings. The recommended value is a third of `mu`.
- **beta** – the distance which guarantees about 75.6% chance of winning. The recommended value is a half of `sigma`.
- **tau** – the dynamic factor which restrains a fixation of rating. The recommended value is `sigma` per cent.
- **draw_probability** – the draw probability between two teams. It can be a `float` or function which returns a `float` by the given two rating (team performance) arguments and

the beta value. If it is a `float`, the game has fixed draw probability. Otherwise, the draw probability will be decided dynamically per each match.

- **backend** – the name of a backend which implements cdf, pdf, ppf. See *trueskill.backends* for more details. Defaults to `None`.

**create_rating**(*mu=None*, *sigma=None*)

Initializes new *Rating* object, but it fixes default mu and sigma to the environment's.

```
>>> env = TrueSkill(mu=0, sigma=1)
>>> env.Rating()
trueskill.Rating(mu=0.000, sigma=1.000)
```

**expose**(*rating*)

Returns the value of the rating exposure. It starts from 0 and converges to the mean. Use this as a sort key in a leaderboard:

```
leaderboard = sorted(ratings, key=env.expose, reverse=True)
```

New in version 0.4.

**make_as_global**()

Registers the environment as the global environment.

```
>>> env = TrueSkill(mu=50)
>>> Rating()
trueskill.Rating(mu=25.000, sigma=8.333)
>>> env.make_as_global()
trueskill.TrueSkill(mu=50.000, ...)
>>> Rating()
trueskill.Rating(mu=50.000, sigma=8.333)
```

But if you need just one environment, *setup()* is better to use.

**quality**(*rating_groups*, *weights=None*)

Calculates the match quality of the given rating groups. A result is the draw probability in the association:

```
env = TrueSkill()
if env.quality([team1, team2, team3]) < 0.50:
    print('This match seems to be not so fair')
```

**Parameters**

- **rating_groups** – a list of tuples or dictionaries containing *Rating* objects.
- **weights** – weights of each players for "partial play".

New in version 0.2.

**rate**(*rating_groups*, *ranks=None*, *weights=None*, *min_delta=0.0001*)

Recalculates ratings by the ranking table:

```
env = TrueSkill()  # uses default settings
# create ratings
r1 = env.create_rating(42.222)
r2 = env.create_rating(89.999)
# calculate new ratings
rating_groups = [(r1,), (r2,)]
rated_rating_groups = env.rate(rating_groups, ranks=[0, 1])
# save new ratings
(r1,), (r2,) = rated_rating_groups
```

`rating_groups` is a list of rating tuples or dictionaries that represents each team of the match. You will get a result as same structure as this argument. Rating dictionaries for this may be useful to choose specific player's new rating:

```
# load players from the database
p1 = load_player_from_database('Arpad Emrick Elo')
p2 = load_player_from_database('Mark Glickman')
p3 = load_player_from_database('Heungsub Lee')
# calculate new ratings
rating_groups = [{p1: p1.rating, p2: p2.rating}, {p3: p3.rating}]
rated_rating_groups = env.rate(rating_groups, ranks=[0, 1])
# save new ratings
for player in [p1, p2, p3]:
    player.rating = rated_rating_groups[player.team][player]
```

**Parameters**

- **rating_groups** – a list of tuples or dictionaries containing *Rating* objects.
- **ranks** – a ranking table. By default, it is same as the order of the `rating_groups`.
- **weights** – weights of each players for "partial play".
- **min_delta** – each loop checks a delta of changes and the loop will stop if the delta is less then this argument.

**Returns** recalculated ratings same structure as `rating_groups`.

**Raises** `FloatingPointError` occurs when winners have too lower rating than losers. higher floating-point precision couls solve this error. set the backend to "mpmath".

New in version 0.2.

## 4.2 Default values

`trueskill.`**MU = 25.0**
Default initial mean of ratings.

`trueskill.`**SIGMA = 8.333333333333334**
Default initial standard deviation of ratings.

`trueskill.`**BETA = 4.166666666666667**
Default distance that guarantees about 75.6% chance of winning.

`trueskill.`**TAU = 0.08333333333333334**
Default dynamic factor.

`trueskill.`**DRAW_PROBABILITY = 0.1**
Default draw probability of the game.

## 4.3 Head-to-head shortcuts

`trueskill.`**rate_1vs1** (*rating1*, *rating2*, *drawn=False*, *min_delta=0.0001*, *env=None*)
A shortcut to rate just 2 players in a head-to-head match:

```
    alice, bob = Rating(25), Rating(30)
    alice, bob = rate_1vs1(alice, bob)
    alice, bob = rate_1vs1(alice, bob, drawn=True)
```

**Parameters**

- **rating1** – the winner's rating if they didn't draw.
- **rating2** – the loser's rating if they didn't draw.
- **drawn** – if the players drew, set this to `True`. Defaults to `False`.
- **min_delta** – will be passed to *rate()*.
- **env** – the *TrueSkill* object. Defaults to the global environment.

**Returns** a tuple containing recalculated 2 ratings.

New in version 0.2.

`trueskill.`**quality_1vs1**(*rating1*, *rating2*, *env=None*)

A shortcut to calculate the match quality between just 2 players in a head-to-head match:

```
    if quality_1vs1(alice, bob) < 0.50:
        print('This match seems to be not so fair')
```

**Parameters**

- **rating1** – the rating.
- **rating2** – the another rating.
- **env** – the *TrueSkill* object. Defaults to the global environment.

New in version 0.2.

## 4.4 Functions for the global environment

`trueskill.`**global_env**()

Gets the *TrueSkill* object which is the global environment.

`trueskill.`**setup**(*mu=25.0*, *sigma=8.333333333333334*, *beta=4.166666666666667*, *tau=0.08333333333333334*, *draw_probability=0.1*, *backend=None*, *env=None*)

Setups the global environment.

**Parameters** **env** – the specific *TrueSkill* object to be the global environment. It is optional.

```
    >>> Rating()
    trueskill.Rating(mu=25.000, sigma=8.333)
    >>> setup(mu=50)
    trueskill.TrueSkill(mu=50.000, ...)
    >>> Rating()
    trueskill.Rating(mu=50.000, sigma=8.333)
```

`trueskill.`**rate**(*rating_groups*, *ranks=None*, *weights=None*, *min_delta=0.0001*)

A proxy function for *TrueSkill.rate()* of the global environment.

New in version 0.2.

trueskill.**quality**(*rating_groups*, *weights=None*)
A proxy function for *TrueSkill.quality()* of the global environment.

New in version 0.2.

trueskill.**expose**(*rating*)
A proxy function for *TrueSkill.expose()* of the global environment.

New in version 0.4.

## 4.5 Draw probability helpers

trueskill.**calc_draw_probability**(*draw_margin*, *size*, *env=None*)
Calculates a draw-probability from the given `draw_margin`.

**Parameters**

- **draw_margin** – the draw-margin.
- **size** – the number of players in two comparing teams.
- **env** – the *TrueSkill* object. Defaults to the global environment.

trueskill.**calc_draw_margin**(*draw_probability*, *size*, *env=None*)
Calculates a draw-margin from the given `draw_probability`.

**Parameters**

- **draw_probability** – the draw-probability.
- **size** – the number of players in two comparing teams.
- **env** – the *TrueSkill* object. Defaults to the global environment.

## 4.6 Mathematical statistics backends

trueskill.backends.**choose_backend**(*backend*)
Returns a tuple containing cdf, pdf, ppf from the chosen backend.

```
>>> cdf, pdf, ppf = choose_backend(None)
>>> cdf(-10)
7.619853263532764e-24
>>> cdf, pdf, ppf = choose_backend('mpmath')
>>> cdf(-10)
mpf('7.6198530241605255e-24')
```

New in version 0.3.

trueskill.backends.**available_backends**()
Detects list of available backends. All of defined backends are `None` – internal implementation, “mpmath”, “scipy”.

You can check if the backend is available in the current environment with this function:

```
if 'mpmath' in available_backends():
    # mpmath can be used in the current environment
    setup(backend='mpmath')
```

New in version 0.3.

CHAPTER 5

# Changelog

## 5.1 Version 0.4.1

Released on Jun 6th 2013.

- Deprecates `dynamic_draw_probability()`.

## 5.2 Version 0.4

Released on Mar 25th 2013.

- Supports dynamic draw probability.
- Replaces `Rating.exposure()` with `TrueSkiil.expose()`. Because the TrueSkill settings have to adjust a fomula to calculate an exposure.
- Deprecates head-to-head shortcut methods in `TrueSkill`. The top-level shortcut functions are still alive.

## 5.3 Version 0.3.1

Released on Mar 6th 2013.

Raises `FloatingPointError` instead of `ValueError` (math domain error) for a problem similar to issue #5 but with more extreme input.

## 5.4 Version 0.3

Released on Mar 5th 2013.

`TrueSkill` got a new option `backend` to choose cdf, pdf, ppf implementation.

When winners have too lower rating than losers, `TrueSkill.rate()` will raise `FloatingPointError` if the backend is `None` or "scipy". But from this version, you can avoid the problem with "mpmath" backend. This was reported at issue #5.

## 5.5 Version 0.2.1

Released on Dec 6th 2012.

Fixes a printing bug on `TrueSkill.quality()`.

## 5.6 Version 0.2

Released on Nov 30th 2012.

- Implements "Partial play".
- Works well in many Python versions, 2.5, 2.6, 2.7, 3.1, 3.2, 3.3 and many interpreters, CPython, Jython, PyPy.
- Supports that using dictionaries as a `rating_group` to choose specific player's rating simply.
- Adds shorcut functions for 2 players individual match, the most usage: `rate_1vs1()` and `quality_1vs1()`,
- `TrueSkill.transform_ratings()` is now called `TrueSkill.rate()`.
- `TrueSkill.match_quality()` is now called `TrueSkill.quality()`.

## 5.7 Version 0.1.4

Released on Oct 5th 2012.

Fixes `ZeroDivisionError` issue. For more detail, see issue#3. Thanks to Yunwon Jeong and Nikos Kokolakis.

## 5.8 Version 0.1.3

Released on Mar 10th 2012.

Improves the match quality performance.

## 5.9 Version 0.1.1

Released on Jan 12th 2012.

Fixes an error in "A" matrix of the match quality algorithm.

## 5.10 Version 0.1

First public preview release.

CHAPTER 6

# Further more

There's the list for users. To subscribe the list, just send a mail to trueskill@librelist.com.

If you want to more details of the TrueSkill algorithm, see also:

- TrueSkill: A Bayesian Skill Rating System by Herbrich, Ralf and Graepel, Thore
- TrueSkill Calcurator by Microsoft Research
- Computing Your Skill by Jeff Moser
- The Math Behind TrueSkill by Jeff Moser

CHAPTER 7

# Licensing and Author

This TrueSkill package is opened under the BSD license but the TrueSkill™ brand is not. Microsoft permits only Xbox Live games or non-commercial projects to use TrueSkill™. If your project is commercial, you should find another rating system. See LICENSE for the details.

I'm Heungsub Lee, a game developer. Any regarding questions or patches are welcomed.

## t