Bayesian ranking for tennis players in PyMC

PyData Amsterdam 2023

Francesco Bruzzesi

Topics of the day

- What’s wrong with the current tennis ranking
- Introduction to Bradley-Terry model
- Implementation in PyMC
- Ranking
- Extensions and other applications

What’s wrong with the ranking?

What’s wrong with the ranking?

The current system works, but it has a few flaws:

What’s wrong with the ranking?

The current system works, but it has a few flaws:

- All opponents are equal

Round	ATP Points
Winner	2 000 points
Finalist	1 200 points
Semi-finalists	720 points
Quarter-finalists	360 points

What's wrong with the ranking?

The current system works, but it has a few flaws:

- All opponents are equal
- It's a number game

name	wins	played	rank	win_rate
Stefanos Tsitsipas	51	71	4	0.718310
Matteo Berrettini	25	34	8	0.735294

What’s wrong with the ranking?

The current system works, but it has a few flaws:

- All opponents are equal
- It’s a number game
- Surfaces are interchangeable

*"I don't want to play here on this (**clay**) surface"* [Daniil Medvedev]

"(...) grass is for golf players" [Casper Ruud]

What’s wrong with the ranking?

The current system works, but it has a few flaws:

- All opponents are equal
- It’s a number game
- Surfaces are interchangeable
- Last week points count as last year

What’s wrong with the ranking?

The current system works, but it has a few flaws:

- All opponents are equal
- It’s a number game
- Surfaces are interchangeable
- Last week points count as last year

TL;DR The current ranking system is a *sum* of player performance over the last 52 weeks.

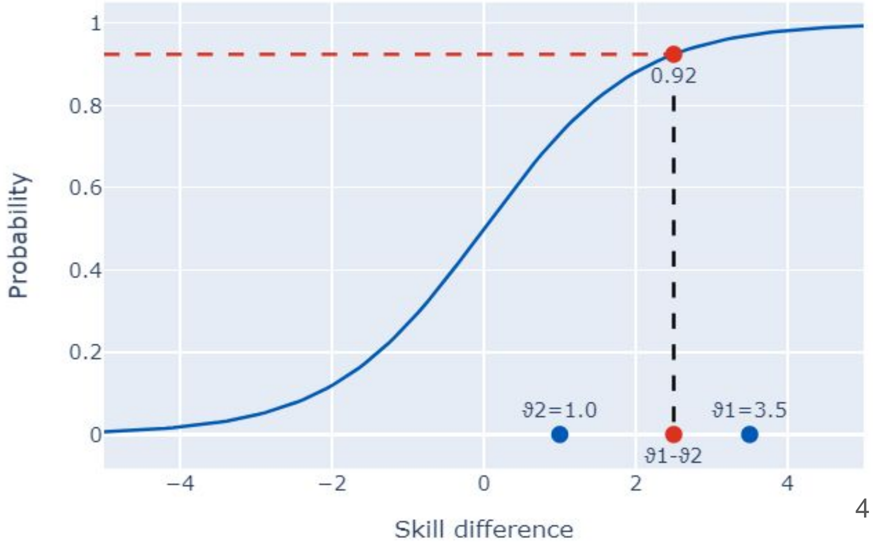
How does Bradley-Terry work?

The Bradley-Terry model is a probabilistic method for *paired comparisons*. It is based on the assumption that the probability of player ***i*** beating player ***j*** is a function of their abilities ϑ_i and ϑ_j

How does Bradley-Terry work?

The Bradley-Terry model is a probabilistic method for *paired comparisons*. It is based on the assumption that the probability of player ***i*** beating player ***j*** is a function of their abilities ϑ_i and ϑ_j :

$$P(i \text{ beats } j) = \text{logistic}(\vartheta_i - \vartheta_j)$$



How does Bradley-Terry work?

The Bradley-Terry model is a probabilistic method for *paired comparisons*. It is based on the assumption that the probability of player ***i*** beating player ***j*** is a function of their abilities ϑ_i and ϑ_j :

$$P(i \text{ beats } j) = \text{logistic}(\vartheta_i - \vartheta_j)$$

We are interested in learning the latent ability ϑ_i for each player from the data (i.e. matches outcome).

Available data

The dataset comes from Jeff Sackmann github repo.

We will focus on main tour from 2021 to 2023 but exclude team, national events, players with less that 10 matches (203 players).

Available data

The dataset comes from Jeff Sackmann github repo.

We will focus on main tour from 2021 to 2023 but exclude team, national events, players with less that 10 matches (203 players).

```
df = load_data(
    start=2021,
    end=2023
)
# Loaded 6134 matches
df.tail()
```

Available data

The dataset comes from Jeff Sackmann github repo.

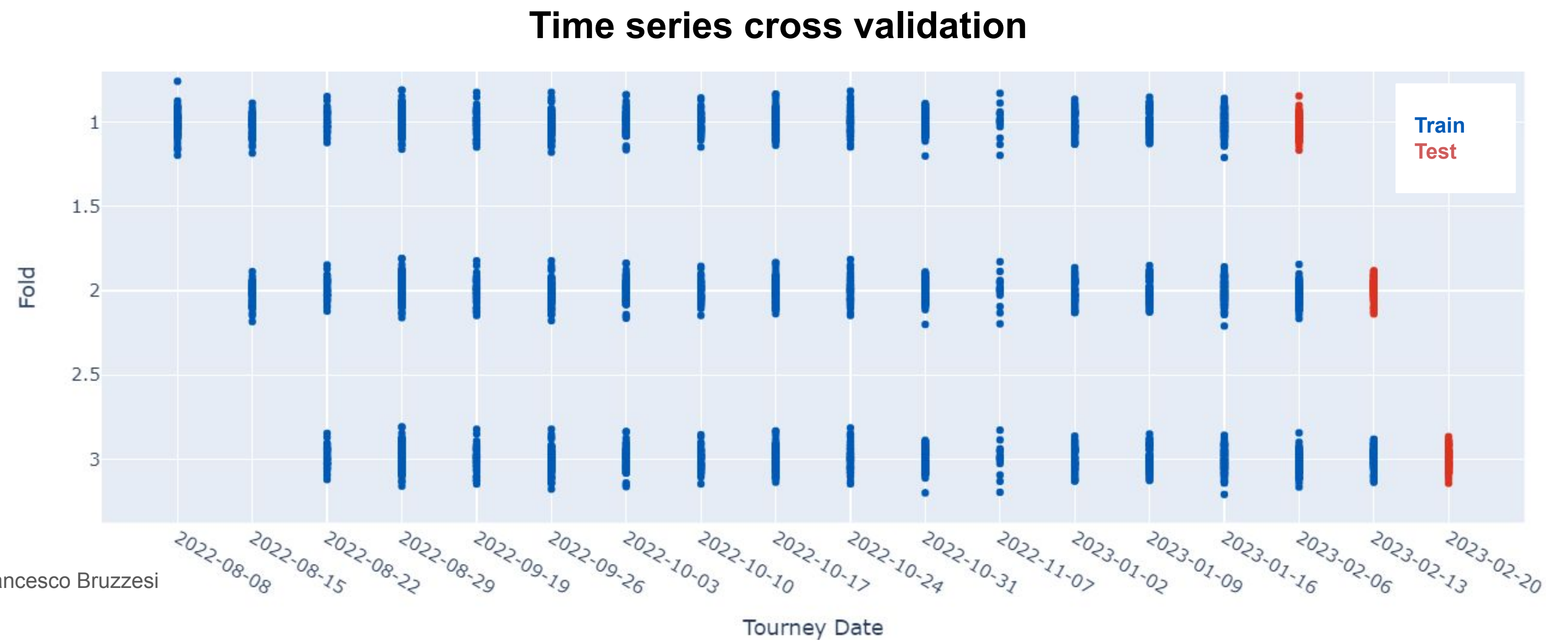
We will focus on main tour from 2021 to 2023 but exclude team, national events, players with less that 10 matches (203 players).

```
df = load_data(
    start=2021,
    end=2023
)
# Loaded 6134 matches
df.tail()
```

Tourney	Date	Surface	Winner Name	Loser Name	Winner Rank	Loser Rank
Hamburg	2023-07-24	Clay	A. Zverev	A. Fils	19	71
Umag	2023-07-24	Clay	A. Popyrin	M. Arnaldi	90	76
Atlanta	2023-07-24	Hard	T. Fritz	A. Vukic	9	82
Hamburg	2023-07-24	Clay	A. Zverev	L. Djere	19	57
Umag	2023-07-24	Clay	A. Popyrin	S. Wawrinka	90	72

Cross Validation

Given a week to forecast, we train with observations one year prior.



Baseline

A naive model assumes that the player with the better ranking will win the match. This will establish a baseline for the “predictive power” of the current ranking system.

Baseline

A naive model assumes that the player with the better ranking will win the match. This will establish a baseline for the “predictive power” of the current ranking system.

```
(df
  .assign(best_rank_win = lambda t: t["winner_rank"]<t["loser_rank"])


)
```

Baseline

A naive model assumes that the player with the better ranking will win the match. This will establish a baseline for the “predictive power” of the current ranking system.

```
(df
 .assign(best_rank_win = lambda t: t["winner_rank"]<t["loser_rank"])
 .loc[lambda t: t["year"].ge(2022), "best_rank_win"]

)
```

Baseline

A naive model assumes that the player with the better ranking will win the match. This will establish a baseline for the “predictive power” of the current ranking system.

```
(df
 .assign(best_rank_win = lambda t: t["winner_rank"]<t["loser_rank"])
 .loc[lambda t: t["year"].ge(2022), "best_rank_win"]
 .mean()
)
# 0.618
```

“Ability Based” model

“Ability Based” model

```
import pymc as pm

with pm.Model() as base_model:
```

“Ability Based” model

```
import pymc as pm

with pm.Model() as base_model:

    X, y = pm.MutableData("X", X_train), pm.MutableData("y", y_train)
    player1, player2 = X[:, 0], X[:, 1]
```

sample values

```
X = array([
    [63, 47],
    [90, 46],
    ...,
    [89, 61]
]) # shape=(n_matches, 2)
y = array([1, 1, ..., 1]) # shape=(n_matches, )
```

“Ability Based” model

```
import pymc as pm

with pm.Model() as base_model:

    X, y = pm.MutableData("X", X_train), pm.MutableData("y", y_train)
    player1, player2 = X[:, 0], X[:, 1]

    a_m = pm.Normal("ability_m", mu=0.0, sigma=1., shape=(n_players_,))
    a_sd = pm.HalfCauchy("ability_sd", beta=1.0)
```

sample values

```
a_m = array([0.5, -0.4, .., 0.1]) # shape=(n_players, )
a_s = array([2.0]) # shape=(1, )
```

“Ability Based” model

```python
import pymc as pm

with pm.Model() as base_model:

    X, y = pm.MutableData("X", X_train), pm.MutableData("y", y_train)
    player1, player2 = X[:, 0], X[:, 1]

    a_m = pm.Normal("ability_m", mu=0.0, sigma=1., shape=(n_players_,))
    a_sd = pm.HalfCauchy("ability_sd", beta=1.0)

    player_ability = pm.Deterministic("player_ability", a_m*a_sd)
    delta_ability = player_ability[player1] - player_ability[player2]
```

sample values

```python
player_ability = array([3.3, -0.7, .., 1.1]) # shape=(n_players, )
delta_ability = array([2.1, -1.2, ..., 0.2]) # shape=(n_matches, )
```

“Ability Based” model

```
import pymc as pm

with pm.Model() as base_model:

    X, y = pm.MutableData("X", X_train), pm.MutableData("y", y_train)
    player1, player2 = X[:, 0], X[:, 1]

    a_m = pm.Normal("ability_m", mu=0.0, sigma=1., shape=(n_players_,))
    a_sd = pm.HalfCauchy("ability_sd", beta=1.0)

    player_ability = pm.Deterministic("player_ability", a_m*a_sd)
    delta_ability = player_ability[player1] - player_ability[player2]

    prob = pm.Deterministic("prob", pm.invlogit(delta_ability))
    _ = pm.Bernoulli("result", p=prob, observed=y)
```

sample values

```
prob = array([0.91, 0.87, .., 0.38])
result = array([1, 1, ..., 0]) # shape=(n_matches, )
```

Fit the model

In order to fit the model, we use the *inference magic button*.

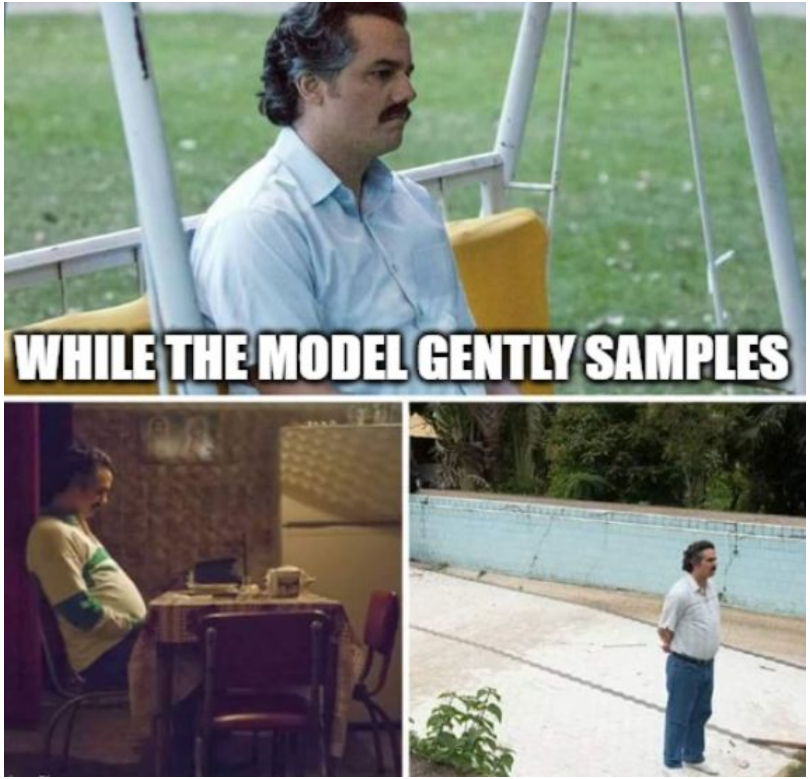
```
with base_model:
```

Fit the model

In order to fit the model, we use the *inference magic button*.

```
with base_model:
    base_trace = pm.sample(
        draws=1000,
        tune=1000,
        chains=4,
        nuts_sampler="numpyro",
        nuts_sampler_kwargs={"chain_method": "parallel"},
        ...
    )
```

Fit the model



Posterior trace

Trace contains all sample stats and posterior information

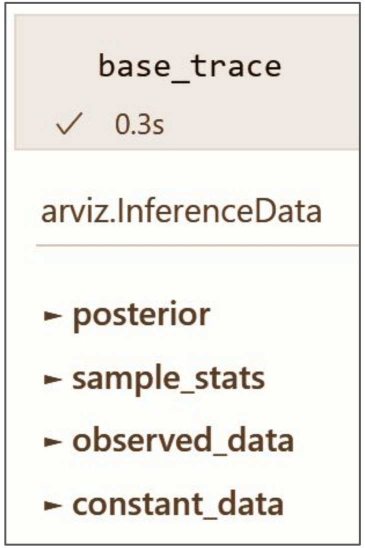
Posterior trace

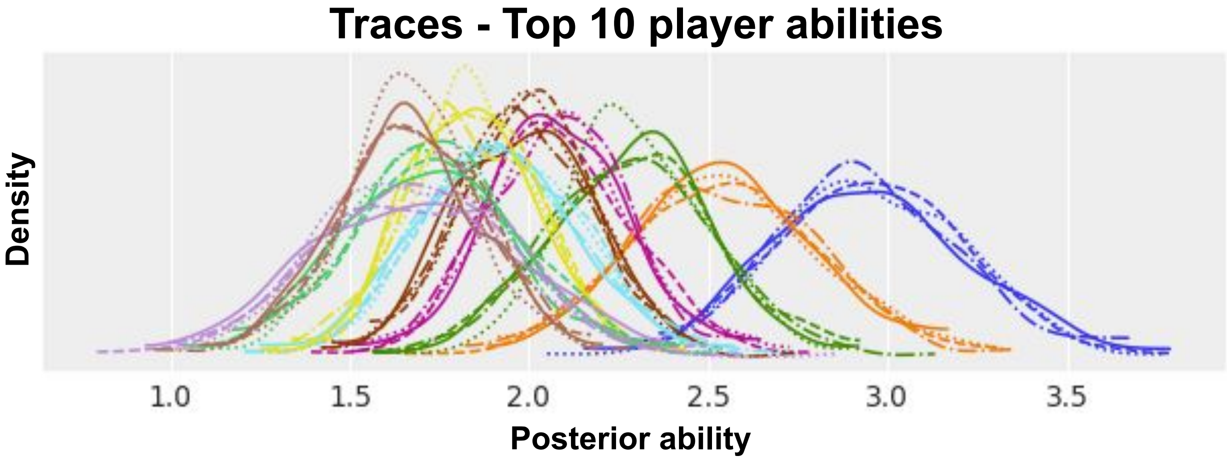
Trace contains all sample stats and posterior information

```
base_trace
✓ 0.3s

arviz.InferenceData

► posterior
► sample_stats
► observed_data
► constant_data
```

Posterior trace

Trace contains all sample stats and posterior information





Predict on new data

Finally we can predict on out of sample data

Predict on new data

Finally we can predict on out of sample data

```
with base_model:
    test_size = X_test.shape[0]
    pm.set_data({
        "X": X_test,
        "y": np.empty(test_size, dtype=int)
    })
```

Predict on new data

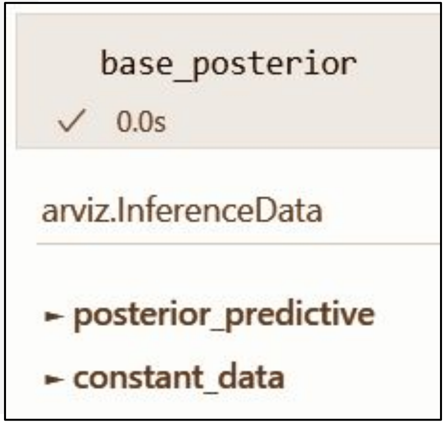
Finally we can predict on out of sample data

```
with base_model:
    test_size = X_test.shape[0]
    pm.set_data({
        "X": X_test,
        "y": np.empty(test_size, dtype=int)
    })

    base_posterior = pm.sample_posterior_predictive(
        trace=base_trace, var_names=[...]
    )
```

Predict on new data

Finally we can predict on out of sample data

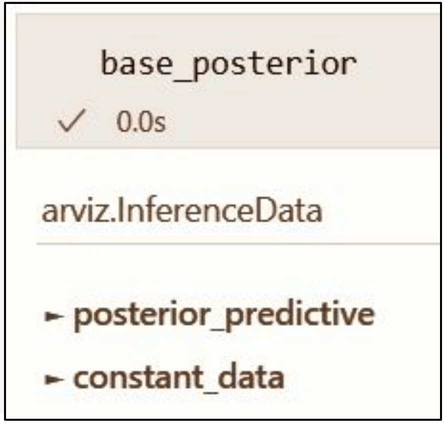
```
with base_model:
    test_size = X_test.shape[0]
    pm.set_data({
        "X": X_test,
        "y": np.empty(test_size, dtype=int)
    })

    base_posterior = pm.sample_posterior_predictive(
        trace=base_trace, var_names=[...]
    )
```



Predict on new data

Finally we can predict on out of sample data

```python
with base_model:
    test_size = X_test.shape[0]
    pm.set_data({
        "X": X_test,
        "y": np.empty(test_size, dtype=int)
    })

    base_posterior = pm.sample_posterior_predictive(
        trace=base_trace, var_names=[...]
    )
```



Model performance: 62.3% accuracy (0.5% above *baseline*)

“Ability, by Surface, with Decay”

“Ability, by Surface, with Decay”

```
with pm.Model() as model:

    X, y = pm.MutableData("X", X), pm.MutableData("y", y)

    player1, player2 = X[:, 0].astype(int), X[:, 1].astype(int)
    surface, sample_weights = X[:, 2].astype(int), X[:, 3]
```

“Ability, by Surface, with Decay”

```
with pm.Model() as model:

    X, y = pm.MutableData("X", X), pm.MutableData("y", y)

    player1, player2 = X[:, 0].astype(int), X[:, 1].astype(int)
    surface, sample_weights = X[:, 2].astype(int), X[:, 3]

    ability_m = pm.Normal("ability_m", 0.0, 1.0, shape=(n_players_,))
    ability_sd = pm.HalfCauchy("ability_sd", beta=1.0)
    base_ability = pm.Deterministic("base_ability", ability_m * ability_sd)
    base_delta = player_ability[player1] - player_ability[player2]
```

“Ability, by Surface, with Decay”

```
with pm.Model() as model:

    X, y = pm.MutableData("X", X), pm.MutableData("y", y)

    player1, player2 = X[:, 0].astype(int), X[:, 1].astype(int)
    surface, sample_weights = X[:, 2].astype(int), X[:, 3]

    ability_m = pm.Normal("ability_m", 0.0, 1.0, shape=(n_players_,))
    ability_sd = pm.HalfCauchy("ability_sd", beta=1.0)
    base_ability = pm.Deterministic("base_ability", ability_m * ability_sd)
    base_delta = player_ability[player1] - player_ability[player2]

    surface_m = pm.Normal("surface_m", 0.0, 1.0, shape=(n_players_, n_surfaces_))
    surface_sd = pm.HalfCauchy("surface_sd", beta=1.0)
    surface_factor = pm.Deterministic("surface_factor", surface_m * surface_sd)
    surface_delta = player_surface[player1, surface] - player_surface[player2, surface]
```

“Ability, by Surface, with Decay”

```
with pm.Model() as model:

    X, y = pm.MutableData("X", X), pm.MutableData("y", y)

    player1, player2 = X[:, 0].astype(int), X[:, 1].astype(int)
    surface, sample_weights = X[:, 2].astype(int), X[:, 3]

    ability_m = pm.Normal("ability_m", 0.0, 1.0, shape=(n_players_,))
    ability_sd = pm.HalfCauchy("ability_sd", beta=1.0)
    base_ability = pm.Deterministic("base_ability", ability_m * ability_sd)
    base_delta = player_ability[player1] - player_ability[player2]

    surface_m = pm.Normal("surface_m", 0.0, 1.0, shape=(n_players_, n_surfaces_))
    surface_sd = pm.HalfCauchy("surface_sd", beta=1.0)
    surface_factor = pm.Deterministic("surface_factor", surface_m * surface_sd)
    surface_delta = player_surface[player1, surface] - player_surface[player2, surface]

    prob = pm.Deterministic("prob", pm.invlogit(delta_ability + delta_surface))
    logp = sample_weights * pm.logp(pm.Bernoulli.dist(p=prob), y)
     _ = pm.Potential("error", logp)
```

“Ability, by Surface, with Decay”

As before, we proceed to:

- Fit the model
- Check there are no issues in convergence
- Run the full backtest/cross validation

Model performance: 63.2% accuracy (1.4% above *baseline*)

“Ability, by Surface, with Decay”

Ranking

Ranking(s) resulting from the model is not too different when compared to the actual one, with a few *big* adjustments

Ranking

Ranking(s) resulting from the model is not too different when compared to the actual one, with a few *big* adjustments

Ranking at 2023-07-24

player_name	rank	core_rank	hard_rank	clay_rank	grass_rank	core_ability	hard_ability	clay_ability	grass_ability
Carlos Alcaraz	1	2	2	2	1	2.730000	2.700000	2.730000	2.960000
Novak Djokovic	2	1	1	1	2	2.890000	3.030000	2.850000	2.930000
Daniil Medvedev	3	3	3	3	3	2.190000	2.420000	2.170000	2.130000
Casper Ruud	4	16	23	8	20	0.730000	0.690000	1.240000	0.690000
Stefanos Tsitsipas	5	7	9	7	7	1.290000	1.380000	1.450000	1.180000
Holger Rune	6	6	7	6	6	1.480000	1.480000	1.590000	1.530000
Andrey Rublev	7	5	6	5	5	1.510000	1.500000	1.610000	1.570000
Jannik Sinner	8	4	4	4	4	1.840000	1.910000	1.870000	1.880000
Taylor Fritz	9	13	10	14	15	0.890000	1.280000	0.920000	0.790000
Frances Tiafoe	10	8	12	11	8	1.100000	1.190000	1.090000	1.180000

Ranking

Ranking(s) resulting from the model is not too different when compared to the actual one, with a few *big* adjustments

Ranking at 2023-07-24

player_name	rank	core_rank	hard_rank	clay_rank	grass_rank	core_ability	hard_ability	clay_ability	grass_ability
Carlos Alcaraz	1	2	2	2	1	2.730000	2.700000	2.730000	2.960000
Novak Djokovic	2	1	1	1	2	2.890000	3.030000	2.850000	2.930000
Daniil Medvedev	3	3	3	3	3	2.190000	2.420000	2.170000	2.130000
Casper Ruud	4	16	23	8	20	0.730000	0.690000	1.240000	0.690000
Stefanos Tsitsipas	5	7	9	7	7	1.290000	1.380000	1.450000	1.180000
Holger Rune	6	6	7	6	6	1.480000	1.480000	1.590000	1.530000
Andrey Rublev	7	5	6	5	5	1.510000	1.500000	1.610000	1.570000
Jannik Sinner	8	4	4	4	4	1.840000	1.910000	1.870000	1.880000
Taylor Fritz	9	13	10	14	15	0.890000	1.280000	0.920000	0.790000
Frances Tiafoe	10	8	12	11	8	1.100000	1.190000	1.090000	1.180000

Ranking

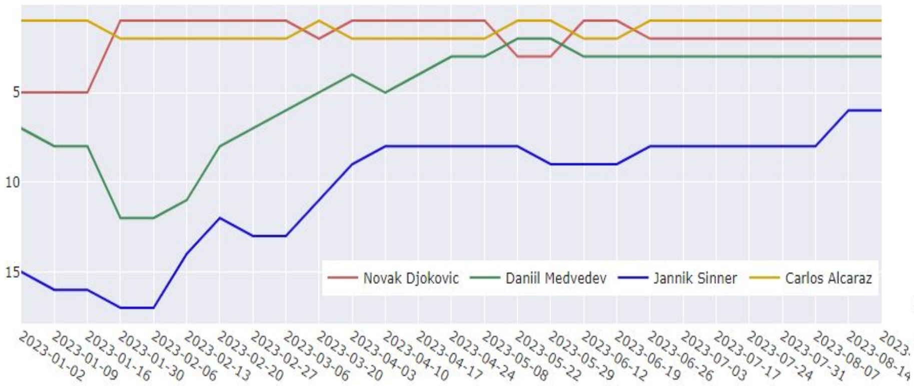
Ranking(s) resulting from the model is not too different when compared to the actual one, with a few *big* adjustments

Ranking at 2023-07-24

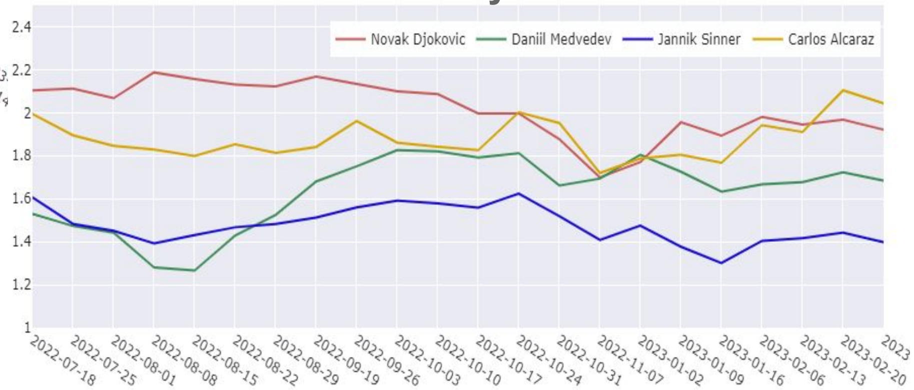
player_name	rank	core_rank	hard_rank	clay_rank	grass_rank	core_ability	hard_ability	clay_ability	grass_ability
Carlos Alcaraz	1	2	2	2	1	2.730000	2.700000	2.730000	2.960000
Novak Djokovic	2	1	1	1	2	2.890000	3.030000	2.850000	2.930000
Daniil Medvedev	3	3	3	3	3	2.190000	2.420000	2.170000	2.130000
Casper Ruud	4	16	23	8	20	0.730000	0.690000	1.240000	0.690000
Stefanos Tsitsipas	5	7	9	7	7	1.290000	1.380000	1.450000	1.180000
Holger Rune	6	6	7	6	6	1.480000	1.480000	1.590000	1.530000
Andrey Rublev	7	5	6	5	5	1.510000	1.500000	1.610000	1.570000
Jannik Sinner	8	4	4	4	4	1.840000	1.910000	1.870000	1.880000
Taylor Fritz	9	13	10	14	15	0.890000	1.280000	0.920000	0.790000
Frances Tiafoe	10	8	12	11	8	1.100000	1.190000	1.090000	1.180000

Comparison over time

Rank over time



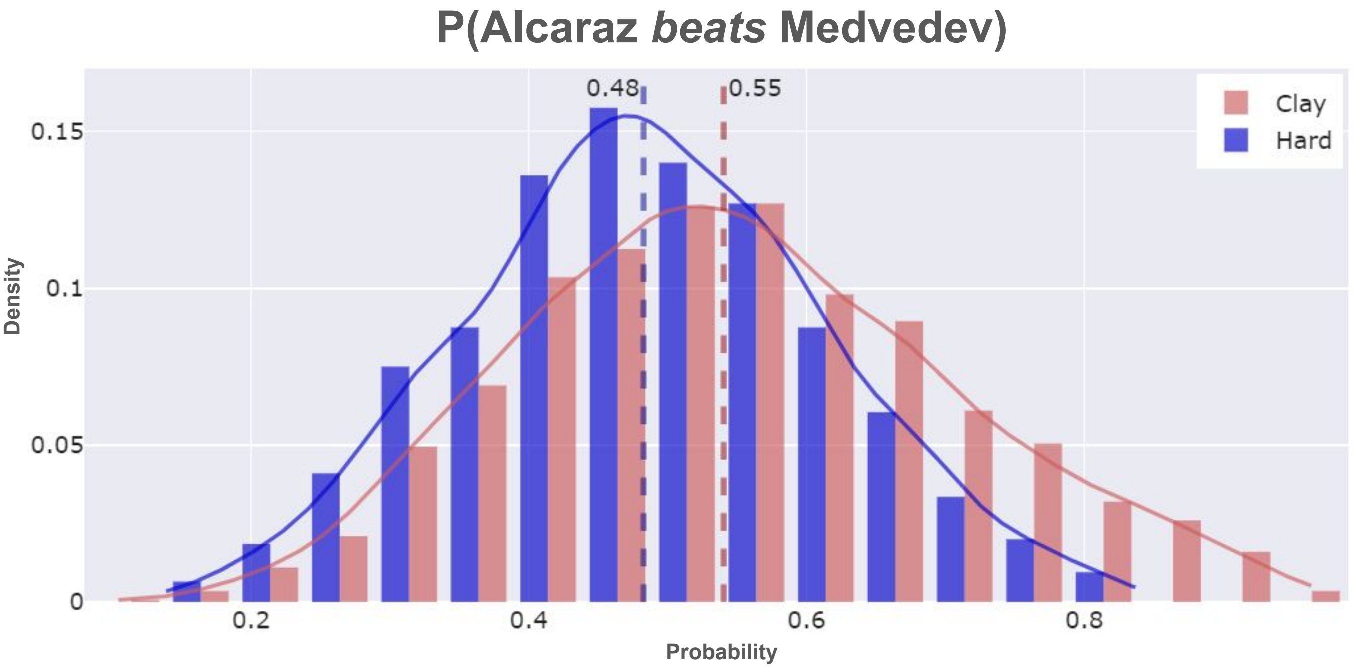
Posterior ability over time



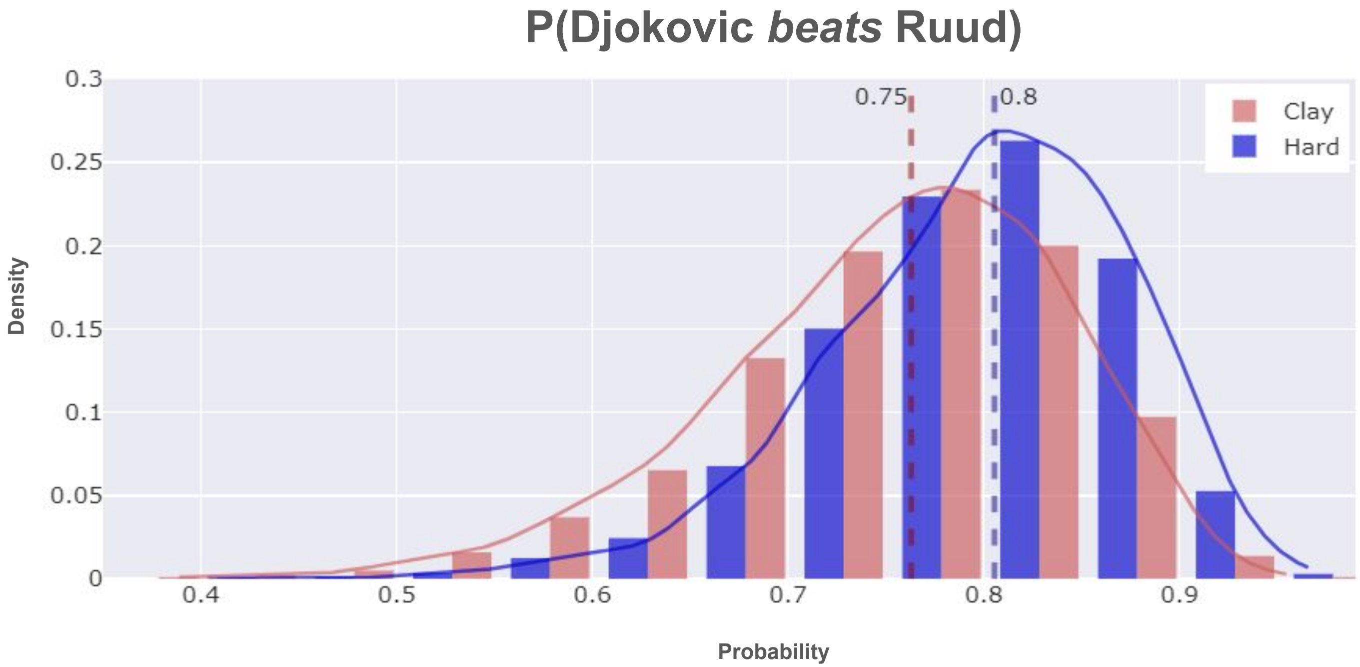
Why Bayesian?

Why Bayesian?

Why Bayesian?



Why Bayesian?



Extensions & other applications

It’s possible to:

- Include other factors (e.g. fatigue)
- Consider priors more robust to outliers (e.g. Student T)
- Use a different model architecture (e.g. a hierarchical model, gaussian processes)
- Extend the period of analysis
- Compare with other models (e.g. ELO Rating)

Where to find me?