stereotype-simulations-summary

November 28, 2023

1 Generational Stereotype Simulations

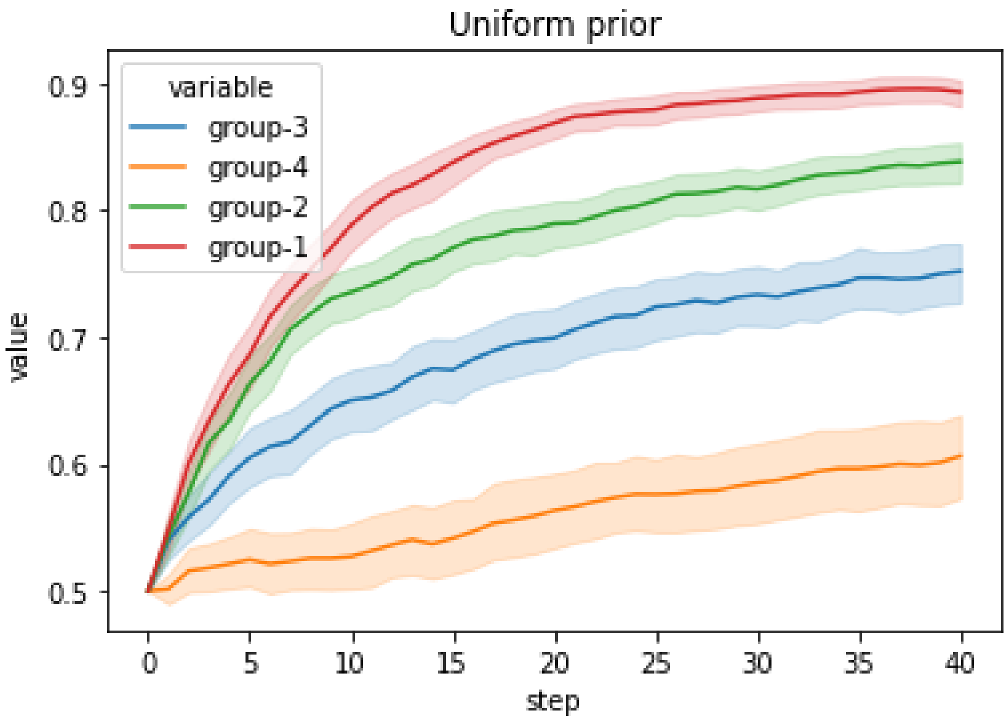
This notebook: - Replicates results from Bai's prior work - Explores social learning in this context - Demonstrates that several hypothesized effects emerge in generational simulations

[1]:
```
import seaborn as sns
import pandas as pd
import matplotlib.pylab as plt

from individual_learner import LearnerAgent, FEATURES, BIASED_PRIOR_PARAMS
import simulations
```

[2]:
```
%load_ext autoreload
%autoreload 2
```

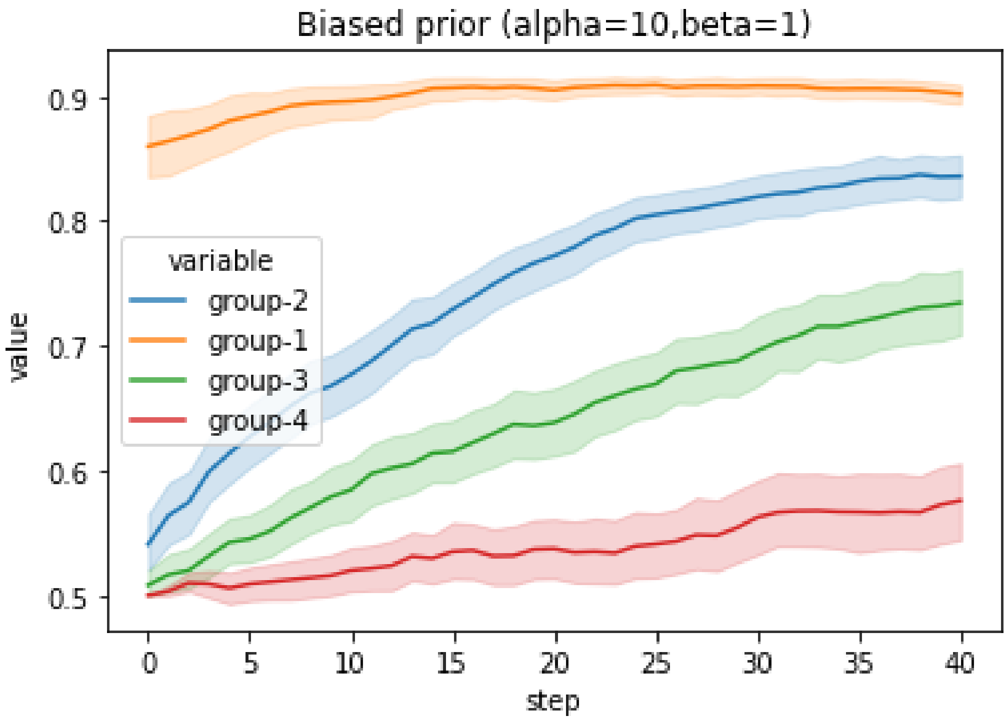
[3]:
```
results = simulations.individual_learner_simulations(100,40)
sns.lineplot(data=pd.melt(results, ['step']), x='step', y='value',␣
 ↪hue='variable')
plt.title("Uniform prior")
```

[3]:
```
Text(0.5, 1.0, 'Uniform prior')
```



[4]:
```
results = simulations.individual_learner_simulations(100,40,␣
  ↪prior=BIASED_PRIOR_PARAMS)
sns.lineplot(data=pd.melt(results, ['step']), x='step', y='value',␣
  ↪hue='variable')
plt.title("Biased prior (alpha=10,beta=1)")
```

[4]:
```
Text(0.5, 1.0, 'Biased prior (alpha=10,beta=1)')
```



2 Social Learning

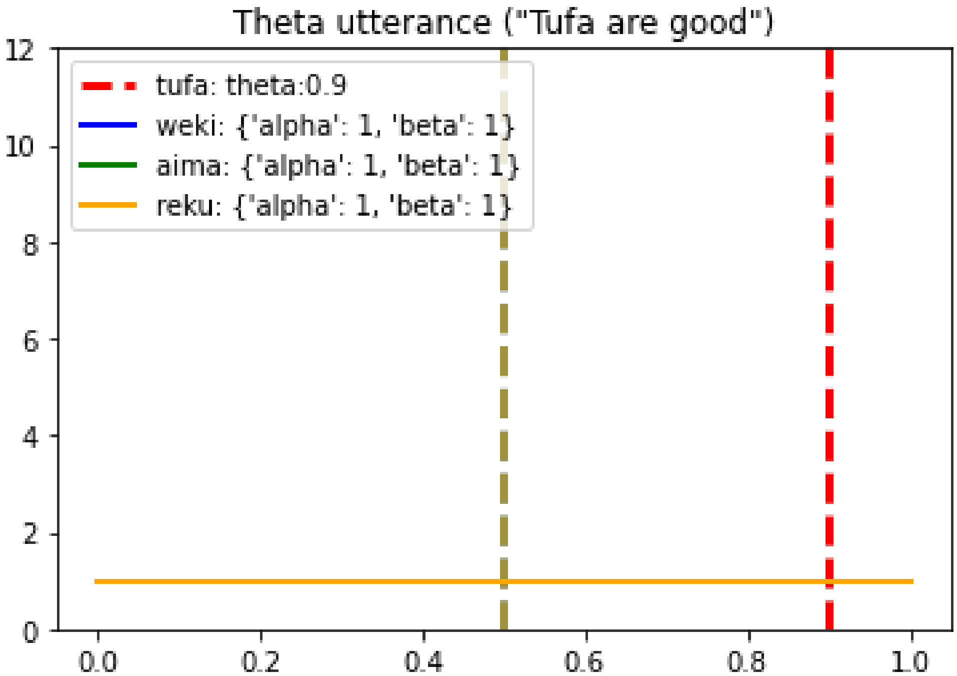
2.1 Listener agents

This section tests speakers which choose a fixed utterance instead of passing the full posteriors.

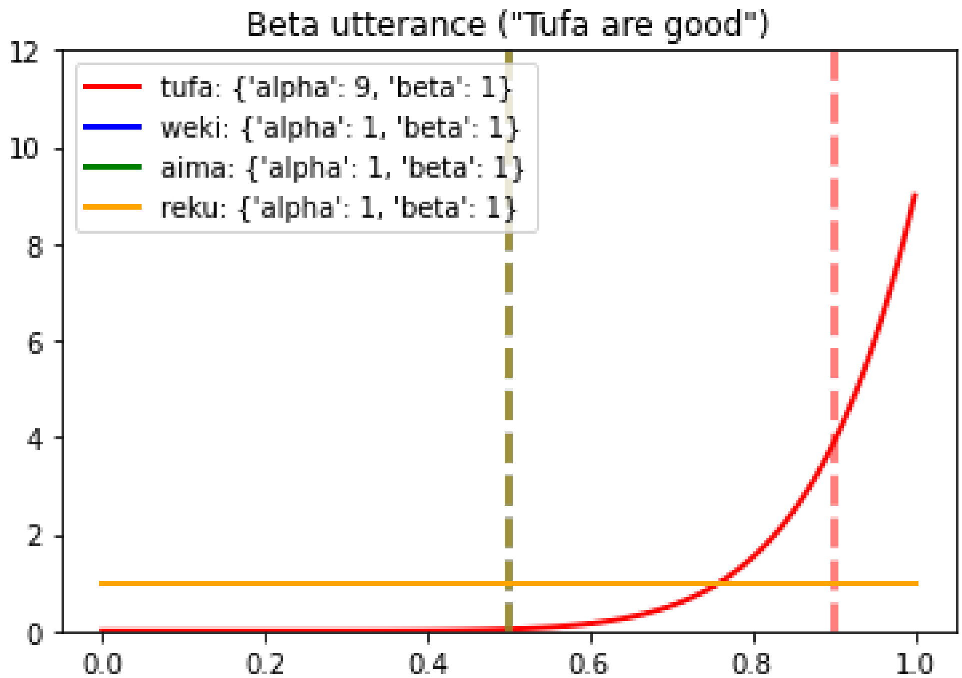
Utterances are either about mean of a population (the θ itself) or some underlying observations (α, β).

θ-based utterances are of the form $\langle$Tufa, .9$\rangle$, while α, β based ones are of the form $\langle$Tufa, $(8, 0)\rangle$.

[5]:
```
theta_utterance = ('tufa', .90)

LearnerAgent().social_learning_from_utterance(theta_utterance).plot_beliefs()
plt.title('Theta utterance ("Tufa are good")');
```



[6]:
```
beta_utterance = ('tufa', (8, 0))

language_conditioned = LearnerAgent().
 ↪social_learning_from_utterance(beta_utterance)
language_conditioned.plot_beliefs()
plt.title('Beta utterance ("Tufa are good")');
```



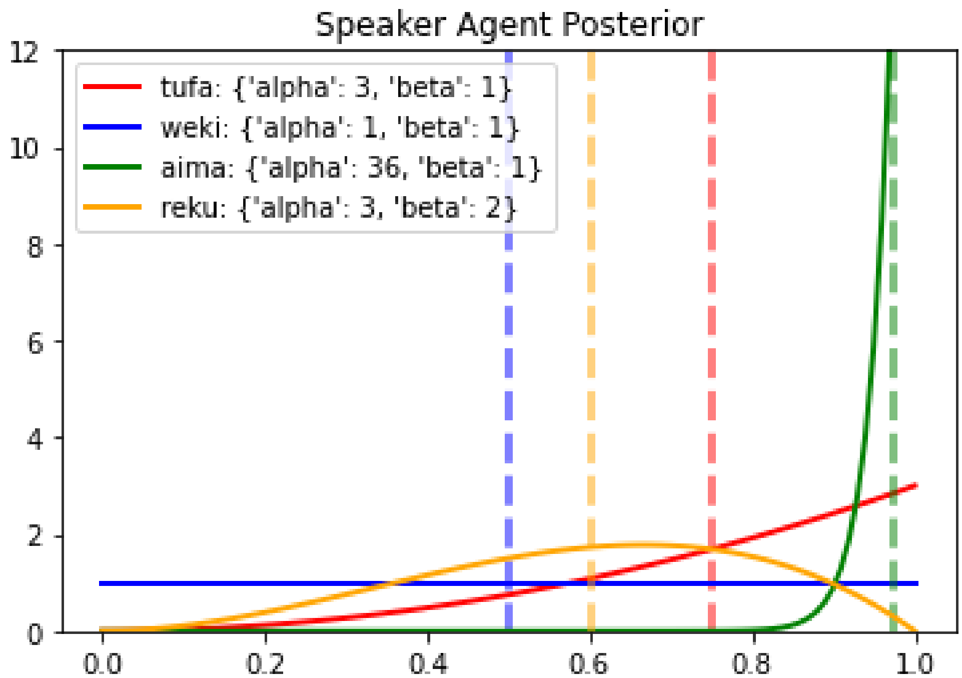
2.2 Speaker Agents

We first have an individual agent gain experience, then evaluate the utility of different utterances conditioned on their posterior.

[7]:
```
from speakers import ExpectedDecisionTheoreticUtility, BETA_UTTERANCES
import visualizations as viz
```

[33]:
```
individual_learner = LearnerAgent()

for i in range(0, 40):
    individual_learner.thompson_sampling()

individual_learner.plot_beliefs()
plt.title('Speaker Agent Posterior')
```

[33]: Text(0.5, 1.0, 'Speaker Agent Posterior')

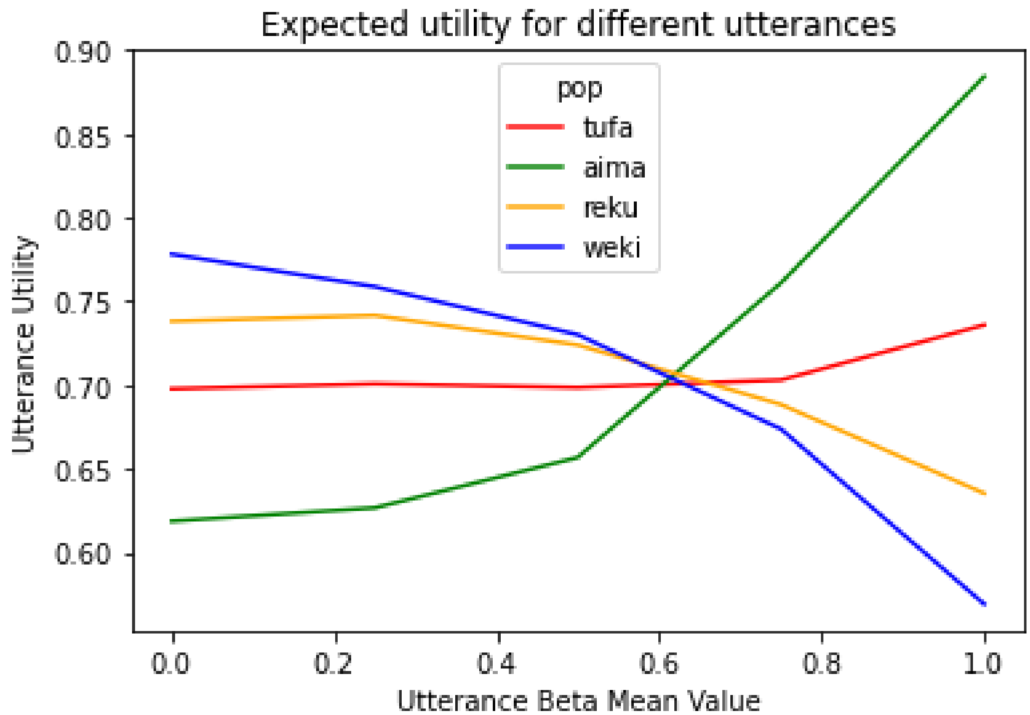


2.2.1 Decision-Theoretic Utility

The speaker's utility for an utterance is the expected value of the *listener's* policy, under the *speaker's* posterior:

$U_{\mathrm{u}} = V_{\pi_L}(\boldsymbol{\theta}_S) = \sum_k \pi_L(a_k \mid u) * \theta_k$.

[36]:
```
dt_speaker = ExpectedDecisionTheoreticUtility(individual_learner,␣
 ↪BETA_UTTERANCES)

viz.visualize_utterance_utilities(dt_speaker, BETA_UTTERANCES)
plt.title("Expected utility for different utterances");

plt.xlabel('Utterance Beta Mean Value')
plt.ylabel('Utterance Utility');
```

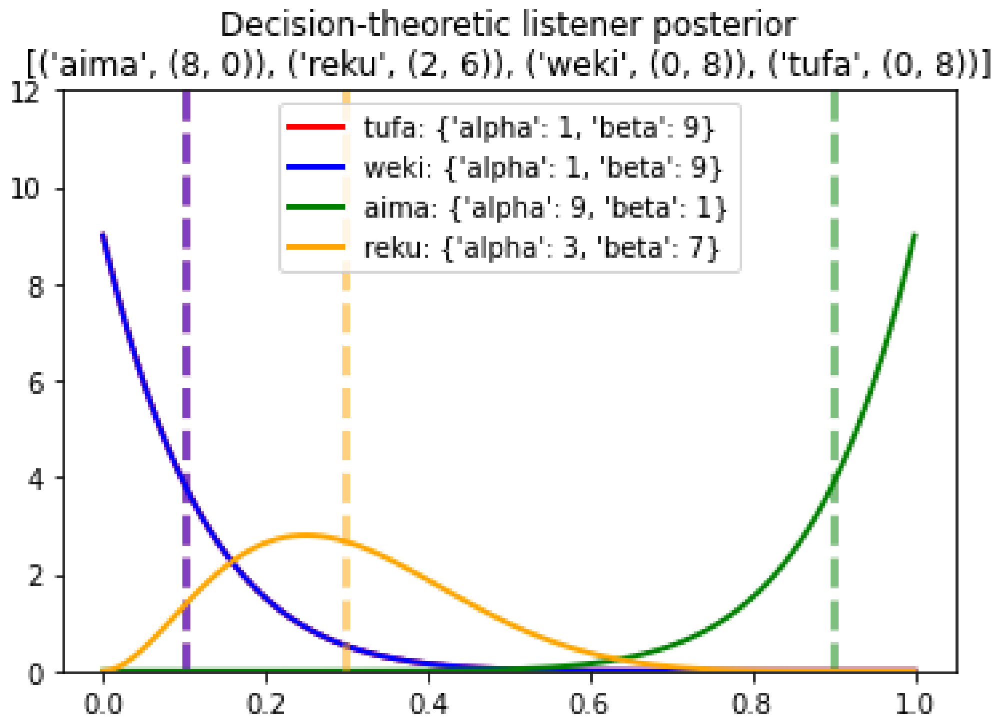


We can use this utility to choose up to k utterances, which determine the listener's posterior beliefs:

[37]:
```
print(dt_speaker.choose_multiple_utterances(1))
print(dt_speaker.choose_multiple_utterances(3))
```

```
[('aima', (8, 0))]
[('aima', (8, 0)), ('tufa', (6, 2)), ('reku', (2, 6))]
```

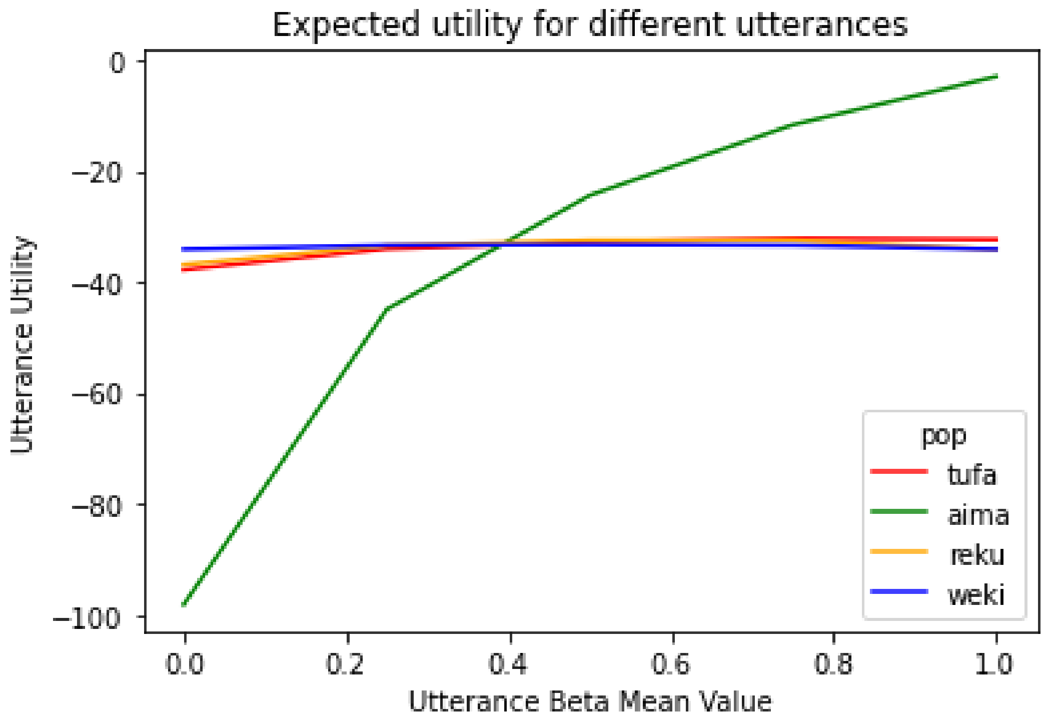
[38]:
```
utts = dt_speaker.choose_multiple_utterances(4)
listener = LearnerAgent()

for u in utts:
    listener = listener.social_learning_from_utterance(u)
listener.plot_beliefs()
plt.title(f'Decision-theoretic listener posterior\n{utts}');
```



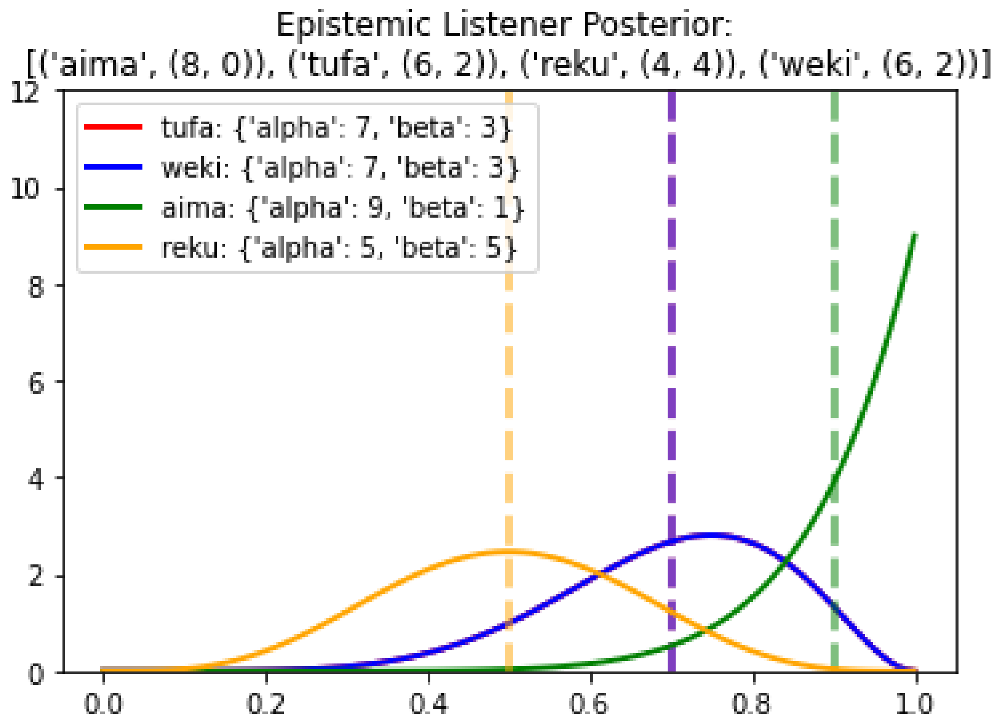
2.2.2 Epistemic Utility

The epistemic speaker tries to minimize the KL divergence between their beliefs and the listener's.

[39]:
```
from speakers import EpistemicUtility

epistemic_speaker = EpistemicUtility(individual_learner, alpha=10)

viz.visualize_utterance_utilities(epistemic_speaker, BETA_UTTERANCES)
plt.title("Expected utility for different utterances");

plt.xlabel('Utterance Beta Mean Value')
plt.ylabel('Utterance Utility');
```

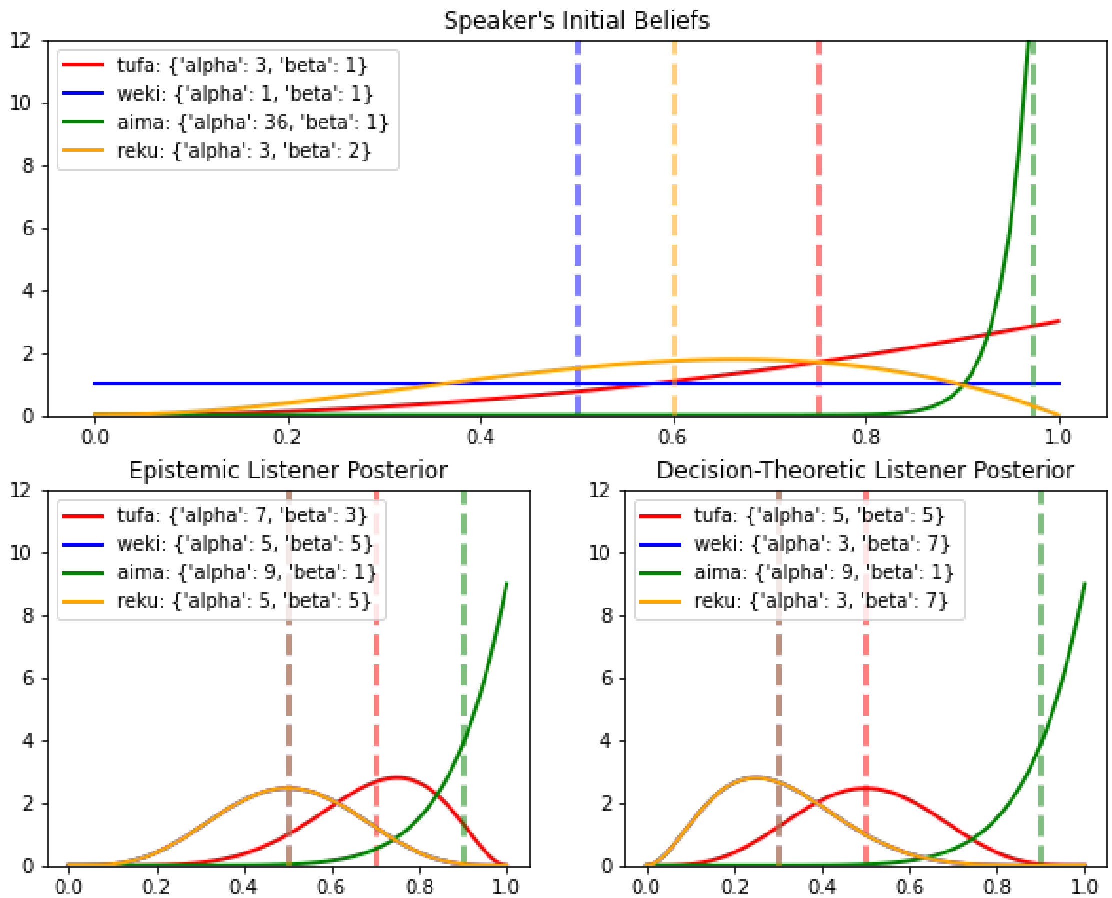


[40]:
```
utts = epistemic_speaker.choose_multiple_utterances(4, visualize=False)

listener = LearnerAgent()

for u in utts:
    listener = listener.social_learning_from_utterance(u)
listener.plot_beliefs()
plt.title(f'Epistemic Listener Posterior: \n{utts}');
```



[41]:
```python
from matplotlib import gridspec

gs = gridspec.GridSpec(2, 2)
fig = plt.figure(figsize=(10, 8))

ax1 = fig.add_subplot(gs[0,0:2])
ax2 = fig.add_subplot(gs[1,0])
ax3 = fig.add_subplot(gs[1,1])

listener = LearnerAgent()
epistemic_speaker.individual_learner.plot_beliefs(ax=ax1)
ax1.set_title("Speaker's Initial Beliefs")

utts = epistemic_speaker.choose_multiple_utterances(4, visualize=False)
for u in utts:
    listener = listener.social_learning_from_utterance(u)
listener.plot_beliefs(ax=ax2)
ax2.set_title(f'Epistemic Listener Posterior');

listener = LearnerAgent()
utts = dt_speaker.choose_multiple_utterances(4, visualize=False)
for u in utts:
    listener = listener.social_learning_from_utterance(u)
```

```
listener.plot_beliefs(ax=ax3)
ax3.set_title(f'Decision-Theoretic Listener Posterior');

plt.show()
```



3 Generational Simulations

3.0.1 Simulation 1: "Ordering Effect"

The "ordering effect" is the hypothesis that agents which recieve social information first (*prior* to indiviual learning) will display *more bias* than agents which recieve social information second.

The "Social first" conditions should show: - **lower** Herfindahl score (less diversity in interaction) - **higher** reward stddev (amplified social stereotypes)

We test two communication variants: "full beta", in which agents directly transfer their beta distributions, and "dt_beta" in which agents produce a single beta utterance using decision-theoretic utility.

[15]:

```
n_chains = 1000
n_generations = 10
n_rounds = 10

social_first = [True, False]
transmissions = ['full_beta', 'dt_beta', 'epistemic_beta']

results_df_list = []
for social in social_first:
    for t in transmissions:

        res = simulations.generational_simulations(n_chains=n_chains,
                                                   n_generations=n_generations,
                                                   n_rounds=n_rounds,
                                                   social_first=social,
                                                   transmission=t,
                                                   n_utterances=1)
        results_df_list.append(res)
```

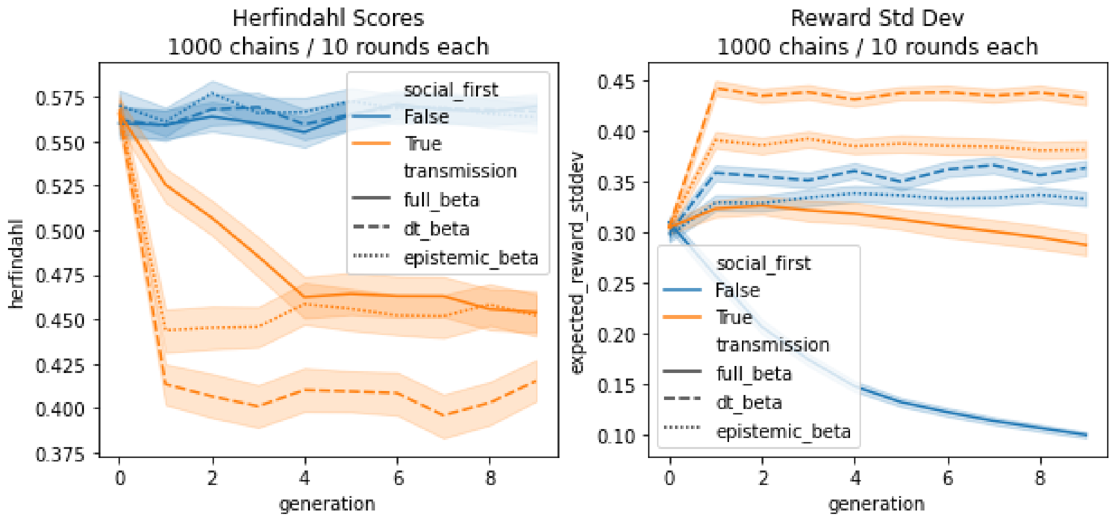
```
full_beta, social first, 1 utts, no initial bias --> 1000 chains took 0.26
minutes.
dt_beta, social first, 1 utts, no initial bias --> 1000 chains took 1.9 minutes.
epistemic_beta, social first, 1 utts, no initial bias --> 1000 chains took 0.46
minutes.
full_beta, individual first, 1 utts, no initial bias --> 1000 chains took 0.28
minutes.
dt_beta, individual first, 1 utts, no initial bias --> 1000 chains took 1.92
minutes.
epistemic_beta, individual first, 1 utts, no initial bias --> 1000 chains took
0.48 minutes.
```

[16]:

```
results = pd.concat(results_df_list)
```

[20]:

```
fig, ax = plt.subplots(1, 2, figsize=(10, 4))

sns.lineplot(data=results, x='generation', y='herfindahl', hue='social_first',␣
 ↪style='transmission', ax=ax[0])
ax[0].set_title(f'Herfindahl Scores\n{n_chains} chains / {n_rounds} rounds␣
 ↪each');
ax[0].legend(loc=1)

sns.lineplot(data=results, x='generation', y='expected_reward_stddev',␣
 ↪hue='social_first', style='transmission', ax=ax[1])
ax[1].set_title(f'Reward Std Dev\n{n_chains} chains / {n_rounds} rounds each');
plt.legend(loc=3);
```



Discussion **Social first shows more bias**. When learners *don't* recieve social information first, their behaviors are unaffected (left: Herfindahl score remains the same). As a result, the chains accumulate information about different sub-populations. When passing full betas, this reduces bias (right: reward stddev decreases with time). Notably, when producing utterances, this reduces bias *relative to social-first*, but overall still produces bias.

In "social first", the agents effectively start Thompson sampling with a strong prior. Chains "lock in" to a belief state and then propagate that.

Utterances show more bias than full parameter passing. Utterance-based Herfindahl scores drop immediately (left) and bias spikes (right). In contrast, with full betas, the Herfindahl drops gradually; and the bias actually decreases a bit over time. This is particularly interesting given that agents can communicate about only a single population in the "utterance" condition.

3.0.2 Simulation 2: "Multiple Utterances"

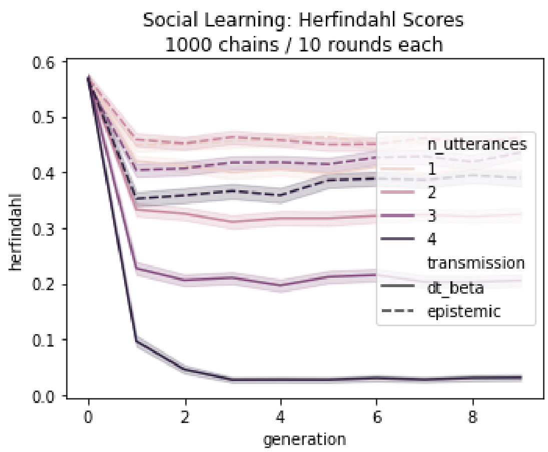
Multiple utterances allow agents to pass information about >1 population. We expect that **bias will increase** with the number of utterances agents produce.

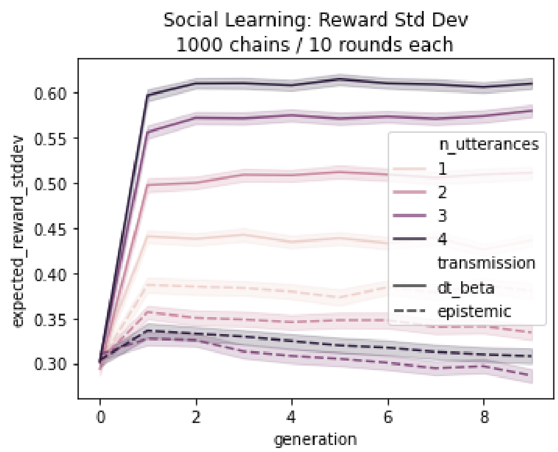
[22]:
```
n_chains = 1000
n_generations = 10
n_rounds = 10

social_first = True
transmission = ['dt_beta', 'epistemic']

n_utterances = [1, 2, 3, 4]

multi_utterance_results_df_list = []
```

```
for n_utt in n_utterances:
    for t in transmission:

        res = simulations.generational_simulations(n_chains=n_chains,
                                                   n_generations=n_generations,
                                                   n_rounds=n_rounds,
                                                   social_first=social_first,
                                                   transmission=t,
                                                   n_utterances=n_utt)
        multi_utterance_results_df_list.append(res)

full_results = pd.concat(multi_utterance_results_df_list)
```

```
dt_beta, social first, 1 utts, no initial bias --> 1000 chains took 1.89
minutes.
epistemic, social first, 1 utts, no initial bias --> 1000 chains took 0.46
minutes.
dt_beta, social first, 2 utts, no initial bias --> 1000 chains took 3.1 minutes.
epistemic, social first, 2 utts, no initial bias --> 1000 chains took 0.59
minutes.
dt_beta, social first, 3 utts, no initial bias --> 1000 chains took 3.86
minutes.
epistemic, social first, 3 utts, no initial bias --> 1000 chains took 0.69
minutes.
dt_beta, social first, 4 utts, no initial bias --> 1000 chains took 4.26
minutes.
epistemic, social first, 4 utts, no initial bias --> 1000 chains took 0.75
minutes.
```

[29]:
```
fig, ax = plt.subplots(1, 2, figsize=(12, 4))

sns.lineplot(data=full_results, x='generation', y='herfindahl',␣
 ↪hue='n_utterances',
             style='transmission', ax=ax[0])
ax[0].set_title(f'Social Learning: Herfindahl Scores\n{n_chains} chains /␣
 ↪{n_rounds} rounds each');
ax[0].legend(loc=5);

sns.lineplot(data=full_results, x='generation', y='expected_reward_stddev',␣
 ↪hue='n_utterances',
             style='transmission', ax=ax[1])

ax[1].set_title(f'Social Learning: Reward Std Dev\n{n_chains} chains /␣
 ↪{n_rounds} rounds each');
ax[1].legend(loc=5);
```





Discussion We confirmed that more utterances results in more bias.

3.0.3 Simulation 3: "Initialization Bias"

[30]:
```
n_chains = 1000
n_generations = 10
n_rounds = 10

social_first = [True, False]
initial_bias = [True, False]
transmissions = ['full_beta', 'dt_beta']

results_df_list = []
for social in social_first:
    for t in transmissions:
        for b in initial_bias:

            res = simulations.generational_simulations(n_chains=n_chains,
                                                                            ␣
 ↪n_generations=n_generations,
                                                                            n_rounds=n_rounds,
                                                                            social_first=social,
                                                                            transmission=t,
                                                                            initial_bias=b)
            results_df_list.append(res)
```

```
full_beta, social first, 1 utts, initial bias --> 1000 chains took 0.26 minutes.
full_beta, social first, 1 utts, no initial bias --> 1000 chains took 0.26
minutes.
dt_beta, social first, 1 utts, initial bias --> 1000 chains took 1.88 minutes.
dt_beta, social first, 1 utts, no initial bias --> 1000 chains took 1.89
```

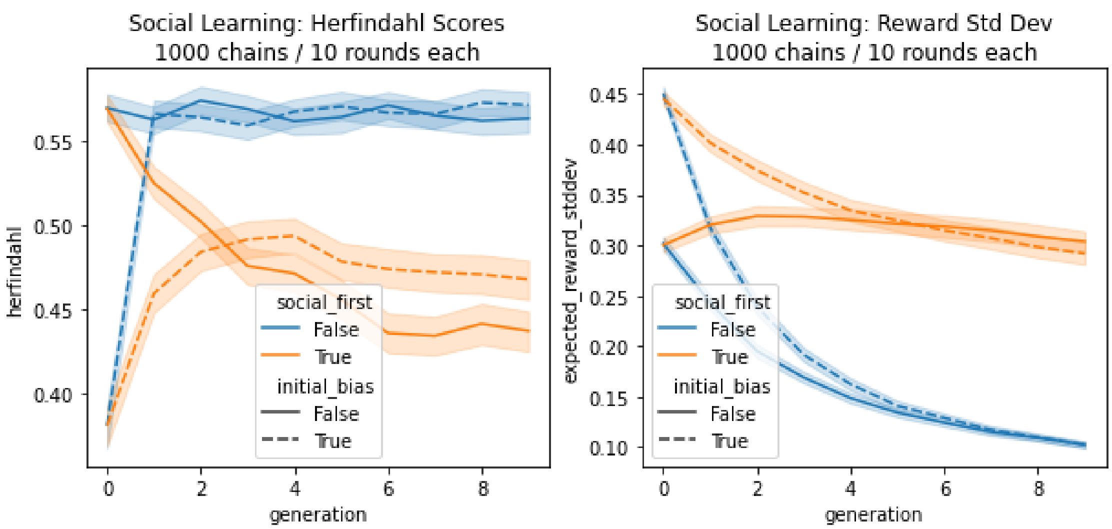
```
minutes.
full_beta, individual first, 1 utts, initial bias --> 1000 chains took 0.28
minutes.
full_beta, individual first, 1 utts, no initial bias --> 1000 chains took 0.28
minutes.
dt_beta, individual first, 1 utts, initial bias --> 1000 chains took 1.91
minutes.
dt_beta, individual first, 1 utts, no initial bias --> 1000 chains took 1.92
minutes.
```

[31]:

```
initial_bias_res = pd.concat(results_df_list)
```

[32]:

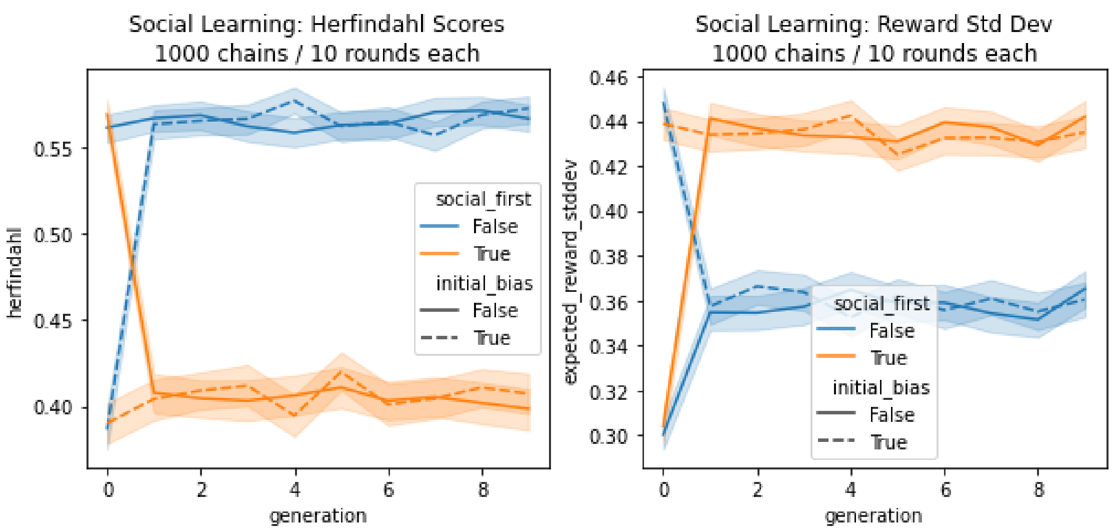
```
fig, ax = plt.subplots(1, 2, figsize=(10, 4))

to_plot = initial_bias_res[initial_bias_res.transmission == 'full_beta']

sns.lineplot(data=to_plot, x='generation', y='herfindahl', hue='social_first',␣
 ↪style='initial_bias', ax=ax[0])
ax[0].set_title(f'Social Learning: Herfindahl Scores\n{n_chains} chains /␣
 ↪{n_rounds} rounds each');

sns.lineplot(data=to_plot, x='generation', y='expected_reward_stddev',␣
 ↪hue='social_first', style='initial_bias', ax=ax[1])
ax[1].set_title(f'Social Learning: Reward Std Dev\n{n_chains} chains /␣
 ↪{n_rounds} rounds each');
```



[35]:

```
fig, ax = plt.subplots(1, 2, figsize=(10, 4))

to_plot = initial_bias_res[initial_bias_res.transmission == 'dt_beta']
```

```
sns.lineplot(data=to_plot, x='generation', y='herfindahl', hue='social_first',␣
↪style='initial_bias', ax=ax[0])
ax[0].set_title(f'Social Learning: Herfindahl Scores\n{n_chains} chains /␣
↪{n_rounds} rounds each');

sns.lineplot(data=to_plot, x='generation', y='expected_reward_stddev',␣
↪hue='social_first', style='initial_bias', ax=ax[1])
ax[1].set_title(f'Social Learning: Reward Std Dev\n{n_chains} chains /␣
↪{n_rounds} rounds each');
```




```
[ ]:
```