Ergo Documentation

Ergo contributors

Oct 21, 2020

USAGE

Ergo is a Python library for integrating model-based and judgmental forecasting.

CHAPTER

ONE

GETTING STARTED

1. To get started with a template to work from, load this Colab notebook.
2. For more information about ergo, see the README.
3. See the sections below to learn more about using ergo.
4. To learn about contributing, read our CONTRIBUTING.md.

CHAPTER

TWO

METACULUS

2.1 Metaculus

class Metaculus(*api_domain='www'*, *username=None*, *password=None*)

The main class for interacting with Metaculus

Parameters

- **api_domain** (Optional[str]) – A Metaculus subdomain (e.g., www, pandemic, finance)
- **username** (Optional[str]) – A Metaculus username (deprecated)
- **password** (Optional[str]) – The password for the given Metaculus username (deprecated)

get_question(*id*, *name=None*)

Load a question from Metaculus

Parameters

- **id** (int) – Question id (can be read off from URL)
- **name** – Name to assign to this question (used in models)

Return type MetaculusQuestion

get_questions(*question_status='all'*, *player_status='any'*, *cat=None*, *pages=1*, *fail_silent=False*, *load_detail=True*)

Retrieve multiple questions from Metaculus API.

Parameters

- **question_status** (Literal['all', 'upcoming', 'open', 'closed', 'resolved', 'discussion']) – Question status
- **player_status** (Literal['any', 'predicted', 'not-predicted', 'author', 'interested', 'private']) – Player's status on this question
- **cat** (Optional[str]) – Category slug
- **pages** (int) – Number of pages of questions to retrieve

Return type List[MetaculusQuestion]

2.2 MetaculusQuestion

class MetaculusQuestion(*id*, *metaculus*, *data*, *name=None*)

A forecasting question on Metaculus

Parameters

- **id** (int) – Question id
- **metaculus** (Any) – Metaculus API instance
- **data** (Dict) – Question JSON retrieved from Metaculus API
- **name** – Name to assign to question (used in models)

Variables

- **activity** –
- **anon_prediction_count** –
- **author** –
- **author_name** –
- **can_use_powers** –
- **close_time** – when the question closes
- **comment_count** –
- **created_time** – when the question was created
- **id** – question id
- **is_continuous** – is the question continuous or binary?
- **last_activity_time** –
- **page_url** – url for the question page on Metaculus
- **possibilities** –
- **prediction_histogram** – histogram of the current community prediction
- **prediction_timeseries** – predictions on this question over time
- **publish_time** – when the question was published
- **resolution** –
- **resolve_time** – when the question will resolve
- **status** –
- **title** –
- **type** –
- **url** –
- **votes** –

static get_central_quantiles(*df*, *percent_kept=0.95*, *side_cut_from='both'*)

Get the values that bound the central (percent_kept) of the sample distribution, i.e., cutting the tails from these values will give you the central. If passed a dataframe with multiple variables, the bounds that encompass all variables will be returned.

Parameters

- **df** (Union[DataFrame, Series, DeviceArray, ndarray]) – pandas dataframe of one or more column of samples
- **percent_kept** (float) – percentage of sample distrubtion to keep
- **side_cut_from** (str) – which side to cut tails from, either 'both','lower', or 'upper'

Returns lower and upper values of the central (percent_kept) of the sample distribution.

refresh_question()
Refetch the question data from Metaculus, used when the question data might have changed

sample_community()
Get one sample from the distribution of the Metaculus community's prediction on this question (sample is denormalized/on the the true scale of the question)

set_data(*key*, *value*)
Set key on data dict

Parameters

- **key** (str) –
- **value** (Any) –

static to_dataframe(*questions*, *columns=['id', 'title', 'resolve_time']*)
Summarize a list of questions in a dataframe

Parameters

- **questions** (List[MetaculusQuestion]) – questions to summarize
- **columns** (List[str]) – list of column names as strings

Return type DataFrame

Returns pandas dataframe summarizing the questions

2.3 ContinuousQuestion

class ContinuousQuestion(*id*, *metaculus*, *data*, *name=None*)
A continuous Metaculus question – a question of the form, what's your distribution on this event?

change_since(*since*)
Calculate change in community prediction median between the argument and most recent prediction

Parameters **since** (datetime) – datetime

Returns change in median community prediction since datetime

community_dist()
Get the community distribution for this question NB: currently missing the part of the distribtion outside the question range

Return type PointDensity

Returns the (true-scale) community distribution as a histogram.

community_dist_in_range()
A distribution for the portion of the current normalized community prediction that's within the question's range, i.e. 0...(len(self.prediction_histogram)-1).

Returns distribution on integers

denormalize_samples(*samples*)
Map samples from the Metaculus normalized scale to the true scale :param samples: samples on the normalized scale :return: samples from a distribution answering the prediction question

(true scale)

property has_predictions
Are there any predictions for the question yet?

property high_open
Are you allowed to place probability mass above the top of this question's range?

Return type bool

property latest_community_percentiles

Returns Some percentiles for the metaculus commununity's latest rough prediction. *prediction_histogram* returns a more fine-grained histogram of the community prediction

property low_open
Are you allowed to place probability mass below the bottom of this question's range?

Return type bool

normalize_samples(*samples*)
Map samples from their true scale to the Metaculus normalized scale :param samples: samples from a distribution answering the prediction question

(true scale)

Returns samples on the normalized scale

property p_outside
How much probability mass is outside this question's range?

Return type Optional[float]

prepare_logistic(*normalized_dist*)
Transform a single logistic distribution by clipping the parameters and adding scale information as needed for submission to Metaculus. The loc and scale have to be within a certain range for the Metaculus API to accept the prediction.

Parameters **dist** – a (normalized) logistic distribution

Return type Logistic

Returns a transformed logistic distribution

prepare_logistic_mixture(*normalized_dist*)
Transform a (normalized) logistic mixture distribution as needed for submission to Metaculus.

Parameters **normalized_dist** (LogisticMixture) – normalized mixture dist

Return type LogisticMixture

Returns normalized dist clipped and formatted for the API

property question_range
Range of answers specified when the question was created

sample_community()
Sample an approximation of the entire current community prediction, on the true scale of the question. The main reason that it's just an approximation is that we don't know exactly where probability mass outside of the question range should be, so we place it arbitrarily

Return type float

Returns One sample on the true scale

sample_normalized_community()
Sample an approximation of the entire current community prediction, on the normalized scale. The main reason that it's just an approximation is that we don't know exactly where probability mass outside of the question range should be, so we place it arbitrarily.

Return type float

Returns One sample on the normalized scale

show_community_prediction(*percent_kept=0.95*, *side_cut_from='both'*, *num_samples=1000*, ***kwargs*)
Plot samples from the community prediction on this question

Parameters

- **percent_kept** (float) – percentage of sample distrubtion to keep
- **side_cut_from** (str) – which side to cut tails from, either 'both','lower', or 'upper'
- **num_samples** (int) – number of samples from the community
- **kwargs** – additional plotting parameters

show_prediction(*samples*, *plot_samples=True*, *plot_fitted=False*, *percent_kept=0.95*, *side_cut_from='both'*, *show_community=False*, *num_samples=1000*, ***kwargs*)
Plot prediction on the true question scale from samples or a submission object. Optionally compare prediction against a sample from the distribution of community predictions

Parameters

- **samples** – samples from a distribution answering the prediction question (true scale). Can either be a 1-d array corresponding to one model's predictions, or a pandas DataFrame with each column corresponding to a distinct model's predictions
- **plot_samples** (bool) – boolean indicating whether to plot the raw samples
- **plot_fitted** (bool) – boolean indicating whether to compute Logistic Mixture Params from samples and plot the resulting fitted distribution. Note this is currently only supported for 1-d samples
- **percent_kept** (float) – percentage of sample distrubtion to keep
- **side_cut_from** (str) – which side to cut tails from, either 'both','lower', or 'upper'
- **show_community** (bool) – boolean indicating whether comparison to community predictions should be made
- **num_samples** (int) – number of samples from the community
- **kwargs** – additional plotting parameters

submit_from_samples(*samples*, *verbose=False*)
Submit prediction to Metaculus based on samples from a prediction distribution

Parameters **samples** – Samples from a distribution answering the prediction question

Return type Response

Returns logistic mixture params clipped and formatted to submit to Metaculus

2.4 LinearQuestion

class LinearQuestion(*id*, *metaculus*, *data*, *name=None*)
A continuous Metaculus question that's on a linear (as opposed to a log) scale"

get_true_scale_logistic(*normalized_dist*)
Convert a normalized logistic distribution to a logistic on the true scale of the question.

Parameters **normalized_dist** (`Logistic`) – normalized logistic distribution

Return type `Logistic`

Returns logistic distribution on the true scale of the question

get_true_scale_mixture(*normalized_dist*)
Convert a normalized logistic mixture distribution to a logistic on the true scale of the question.

Parameters **normalized_dist** (`LogisticMixture`) – normalized logistic mixture dist

Return type `LogisticMixture`

Returns same distribution rescaled to the true scale of the question

2.5 LogQuestion

class LogQuestion(*id*, *metaculus*, *data*, *name=None*)

2.6 LinearDateQuestion

class LinearDateQuestion(*id*, *metaculus*, *data*, *name=None*)

date_to_timestamp(*date*)
Turn a date string in %Y-%m-%d format into a timestamp. Metaculus uses this format for dates when specifying the range of a date question. We're assuming Metaculus is interpreting these date strings as UTC.

Returns A Unix timestamp

sample_community()
Sample an approximation of the entire current community prediction, on the true scale of the question.

Returns One sample on the true scale

2.7 BinaryQuestion

class BinaryQuestion(*id*, *metaculus*, *data*, *name=None*)
A binary Metaculus question – how likely is this event to happen, from 0 to 1?

change_since(*since*)
Calculate change in community prediction between the argument and most recent prediction

Parameters **since** (`datetime`) – datetime

Returns change in community prediction since datetime

sample_community()
Sample from the Metaculus community distribution (Bernoulli).

Return type bool

score_my_predictions()
Score all of my predictions according to the question resolution (or according to the current community prediction if the resolution isn't available)

Returns List of ScoredPredictions with Brier scores

score_prediction(*prediction*, *resolution*)
Score a prediction relative to a resolution using a Brier Score.

Parameters

- **prediction** – how likely is the event to happen, from 0 to 1?
- **resolution** (float) – how likely is the event to happen, from 0 to 1? (0 if it didn't, 1 if it did)

Return type ScoredPrediction

Returns ScoredPrediction with Brier score, see https://en.wikipedia.org/wiki/Brier_score#Definition 0 is best, 1 is worst, 0.25 is chance

submit(*p*)
Submit a prediction to my Metaculus account

Parameters **p** (float) – how likely is the event to happen, from 0 to 1?

Return type Response

CHAPTER

THREE

FORETOLD

3.1 Foretold

class Foretold(*token=None*)
Interface to Foretold

get_question(*id*)
Retrieve a single question by its id

get_questions(*ids*)

Retrieve many questions by their ids

ids (List[string]): List of foretold question ids (should be less than 500 per request)

Returns: List of questions corresponding to the ids, or None for questions that weren't found.

3.2 ForetoldQuestion

class ForetoldQuestion(*id*, *foretold*, *data=None*)
"Information about foretold question, including aggregated distribution

plotCdf()
Plot the CDF

quantile(*q*)
Quantile of distribution

sample_community()
Sample from CDF

submit_from_samples(*samples*, *length=20*)
Submit a prediction to Foretold based on the given samples

Parameters

- **samples** (Union[ndarray, Series]) – Samples on which to base the submission
- **length** (int) – The length of the CDF derived from the samples

Return type Response

CHAPTER

FOUR

PREDICTIT

This module lets you get question and prediction information from PredictIt via the API (https://predictit.freshdesk.com/support/solutions/articles/12000001878)

4.1 PredictIt

class PredictIt
The main class for interacting with PredictIt.

get_market(*id*)
Return the PredictIt market with the given id. A market's id can be found in the url of the market.

Parameters **id** (int) – market id

Return type *PredictItMarket*

Returns market

property markets
Generate all of the markets currently in PredictIt.

Return type Generator[*PredictItMarket*, None, None]

Returns iterator of PredictIt markets

refresh_markets()
Refetch all of the markets from the PredictIt API.

4.2 PredictItMarket

class PredictItMarket(*predictit*, *data*)
A PredictIt market.

Parameters

- **predictit** (*PredictIt*) – PredictIt API instance
- **data** (Dict) – Market JSON retrieved from PredictIt API

Variables

- ***predictit*** (PredictIt) – PredictIT API instance
- **api_url** (*str*) – url of the PredictIt API for the given question
- **id** (*int*) – id of the market

- **name** (*str*) – name of the market
- **shortName** (*str*) – shortened name of the market
- **image** (*str*) – url of the image resource of the market
- **url** (*str*) – url of the market in PredictIt
- **status** (*str*) – status of the market. Closed markets aren't included in the API, so always "Open"
- **timeStamp** (*datetime.datetime*) – last time the market was updated. The API updates every minute, but timestamp can be earlier if it hasn't been traded in

get_question(*id*)
Return the specified question given by the id number.

Parameters **id** (int) – question id

Return type *PredictItQuestion*

Returns question

property questions
Generate all of the questions in the market.

Return type Generator[*PredictItQuestion*, None, None]

Returns generator of questions in market

refresh()
Refetch the market data from PredictIt, used when the question data might have changed.

4.3 PredictItQuestion

class PredictItQuestion(*market*, *data*)
A single binary question in a PredictIt market.

Parameters

- **market** (*PredictItMarket*) – PredictIt market instance
- **data** (Dict) – Contract JSON retrieved from PredictIt API

Variables

- **market** (PredictItMarket) – PredictIt market instance
- **id** (*int*) – id of the contract
- **dateEnd** (*datetime.datetime*) – end-date of a market, usually None
- **image** (*str*) – url of the image resource for the contract
- **name** (*str*) – name of the contract
- **shortName** (*str*) – shortened name of the contract
- **status** (*str*) – status of the contract. Closed markets aren't included in the API, so always "Open"
- **lastTradePrice** (*float*) – last price the contract was traded at
- **bestBuyYesCost** (*float*) – cost to buy a single Yes share
- **bestBuyNoCost** (*float*) – cost to buy a single No share

- **bestSellYesCost** (*float*) – cost to sell a single Yes share
- **bestSellNoCost** (*float*) – cost to sell a single No share
- **lastClosePrice** (*float*) – price the contract closed at the previous day
- **displayOrder** (*int*) – position of the contract in PredictIt. Defaults to 0 if sorted by lastTradePrice

refresh()
Refetch the market data from PredictIt and reload the question.

sample_community()
Sample from the PredictIt community distribution (Bernoulli).

Return type bool

Returns true/false

static to_dataframe(*questions*, *columns=None*)
Summarize a list of questions in a dataframe

Parameters

- **questions** (List[*PredictItQuestion*]) – questions to summarize
- **columns** – list of column names as strings

Return type DataFrame

Returns pandas dataframe summarizing the questions

CHAPTER

FIVE

INFERENCE

5.1 tag

tag(*value*, *name*)

5.2 run

run(*model*, *num_samples=5000*, *ignore_untagged=True*, *rng_seed=0*)

Run model forward, record samples for variables. Return dataframe with one row for each execution.

Return type `DataFrame`

CHAPTER

SIX

DISTRIBUTIONS

6.1 normal

normal(*mean=0*, *stdev=1*, ***kwargs*)

6.2 normal_from_interval

normal_from_interval(*low*, *high*, ***kwargs*)

6.3 lognormal

lognormal(*loc=0*, *scale=1*, ***kwargs*)

6.4 lognormal_from_interval

lognormal_from_interval(*low*, *high*, ***kwargs*)

6.5 uniform

uniform(*low=0*, *high=1*, ***kwargs*)

6.6 beta

beta(*a=1*, *b=1*, ***kwargs*)

6.7 beta_from_hits

beta_from_hits(*hits*, *total*, ***kwargs*)

6.8 categorical

categorical(*ps*, ***kwargs*)

6.9 halfnormal

halfnormal(*stdev=1*, ***kwargs*)

6.10 halfnormal_from_interval

halfnormal_from_interval(*high*, ***kwargs*)

6.11 random_choice

random_choice(*options*, *ps=None*)

6.12 random_integer

random_integer(*min*, *max*, ***kwargs*)

Return type `int`

6.13 flip

flip(*p=0.5*, ***kwargs*)

CHAPTER

SEVEN

CONTRIBUTE TO ERGO CORE

To get started:

1. `git clone https://github.com/oughtinc/ergo.git`
2. `poetry install`
3. `poetry shell`

7.1 `poetry`

Ergo uses poetry to manage its dependencies and environments.

Follow these directions to install poetry if you don't already have it.

Troubleshooting: If you get `Could not find a version that satisfies the requirement jaxlib ...` after using poetry to install, this is probably because your virtual environment has old version of pip due to how poetry choses pip versions.

Try:

1. `poetry run pip install -U pip`
2. `poetry install` again

7.2 Before submitting a PR

1. Run `poetry install` to make sure you have the latest dependencies
2. Format code using `make format` (black, isort)
3. Run linting using `make lint` (flake8, mypy, black check)
4. Run tests using `make test`
    - To run the tests in `test_metaculus.py`, you'll need our secret *.env* file. If you don't have it, you can ask us for it, or rely on Travis CI to run those tests for you.
5. Generate docs using `make docs`, load `docs/build/html/index.html` and review the generated docs
6. Or run all of the above using `make all`

7.3 Conventions

Import `numpy` as follows:

```
import jax.numpy as np
import numpy as onp
```

CHAPTER

EIGHT

CONTRIBUTE TO ERGO NOTEBOOKS

8.1 How to change a notebook and make a PR

1. Open the notebook in JupyterLab or Colab (*Run a notebook in Colab or JupyterLab*)
2. Make your changes
3. Follow our *Notebook Style*
4. Run the notebook in Colab. Save the .ipynb file (with output) in `ergo/notebooks`
5. Run *make scrub*. This will produce a scrubbed version of the notebook in *ergo/notebooks/scrubbed/*.

    1. You can *git diff* the scrubbed version against the previous scrubbed version to more easily see what you changed

        2. You may want to use nbdime for better diffing

6. You can now make a PR with your changes. If you make a PR in the original ergo repo (not a fork), you can then use the auto-comment from ReviewNB to more thoroughly vet your changes

CHAPTER

NINE

RUN A NOTEBOOK IN COLAB OR JUPYTERLAB

9.1 Colab

1. Go to https://colab.research.google.com/:
1. click “GitHub” on the “new notebook” dialog, then enter the notebook URL. Or:
2. go to “Upload” and upload the notebooks `ipynb` file. Or:
2. Install and use the Open in Colab Chrome extension

9.2 JupyterLab

1. `git clone https://github.com/oughtinc/ergo.git`
2. `poetry install`
3. `poetry shell`
4. `jupyter lab`

CHAPTER

TEN

NOTEBOOK STYLE

How to clean up a notebook for us to feature:

1. Make sure that the notebook meets a high standard in general:
    1. high-quality code
    2. illuminating data analysis
    3. clear communication of what you're doing and your findings
    4. as short as possible, but no shorter
    5. this random style guide I found in a few minutes of Googling seems good, but it's not our official style guide or anything
2. Do the following specific things to clean up:
    1. as much as possible, avoid showing extraneous output from cells
        1. you can use the `%%capture` magic to suppress all output from a cell (helpful if a function in the cell prints something)
        2. you can add a `;` at the end of the last line in a cell to suppress printing the return value of the line
        3. think about what cells the reader really needs to see vs. which ones just have to be there for setup or whatnot. Collapse the latter.
    3. use the latest version of `ergo`
    4. make sure that any secrets like passwords are removed from the notebook
    5. Pull out any code not central to the main point of the model into a module in `ergo/contrib/`. See *Notebook contrib folder* for details.

The featured notebooks in our README should be exemplars of the above, so refer to those to see what this looks like in practice.

CHAPTER

ELEVEN

NOTEBOOK CONTRIB FOLDER

11.1 Adding new packages

For modules providing functionality specific to the questions addressed in a notebook, create a new package in contrib `/ergo/contrib/{your_package}` and include an `__init__.py` file. You can then access it in your notebook with:

```
from ergo.contrib.{your_package} import {module_you_want}
```

For modules providing more general functionality of use across notebooks (and perhaps a candidate for inclusion in core ergo), you can use `/ergo/contrib/utils`. You can either add a new module or extend an existing one. You can then access it with:

```
from ergo.contrib.utils import {module_you_want}
```

11.2 Adding dependencies

1. Usual poetry way with –optional flag

```
poetry add {pendulum} --optional
```

2. You can then (manually in the `pyproject.toml`) add it to the 'notebook' group

(Look for "extras" in `pyproject.toml`)

```
[tool.poetry.extras]
notebooks = [
             "pendulum",
             "scikit-learn",
             "{your_dependency}"
            ]
```

(To my knowledge) there is no way currently to do this second step with the CLI.

This allows people to then install the additional notebook dependencies with:

```
poetry install -E notebooks
```

CHAPTER

TWELVE

LOADING DATA FROM GOOGLE SHEETS

Three methods for loading data from google sheets into a Colab Notebook

12.1 Method 1 (Public CSV)

If you're willing to make your spreadsheet public, you can publish it as a CSV file on Google Sheets. Go to File > Publish to the Web, and select the CSV format. Then you can copy the published url, and load it in python using pandas.

```
import pandas as pd
df = pd.read_csv(url)
```

12.2 Method 2 (OAuth)

This method requires the user of the colab to authorize it every time the colab runs, but can work with non-public sheets

```
# Authentication
import google
google.colab.auth.authenticate_user()
google_sheets_credentials = GoogleCredentials.get_application_default()
gc = gspread.authorize(google_sheets_credentials)

# Load spreadsheet
wb = gc.open_by_url(url)
sheet = wb.worksheet(sheet)
values = sheet.get_all_values()
```

12.3 Method 3 (Service Account)

This method requires your to follow the instructions at https://gspread.readthedocs.io/en/latest/oauth2.html to create a google service account. You then need to share the google sheet with the service account email address.

```
# Need a newer version of gspread than included by default in Colab
!pip install --upgrade gspread

service_account_info = {} #JSON for google service account
```

(continues on next page)

(continued from previous page)

```
import gspread
from google.oauth2.service_account import Credentials

scope = ['https://spreadsheets.google.com/feeds',
        'https://www.googleapis.com/auth/drive']

credentials = Credentials.from_service_account_info(service_account_info,␣
↪scopes=scope)

gc = gspread.authorize(credentials)

# Load spreadsheet
wb = gc.open_by_url(url)
sheet = wb.worksheet(sheet)
values = sheet.get_all_values()
```

PYTHON MODULE INDEX

e

INDEX

P

Q

R

S

T

U