Web Interface for Best-Worst-Scaling Documentation

Release 0.0.1

D.Nguyen, M.Charniuk, S.Safdel

Feb 28, 2020

CONTENTS

Welcome to the Project's documentation!

This is a documentation for the source code to the web application for **Best-Worst-Scaling**. The source code includes scripts for frontend and backend.

Backend is mainly developed with Python3.6 using Flask and its extensions. The application is also integrated with Amazon Crowdsourcing Platform using its API.

For the frontend development, besides HTML, CSS, JavaScript and Bootstrap4, Jinja2 is used for dynamic rendering of the most of the templates.

CHAPTER

ONE

LINKS

Source code

CHAPTER TWO

CONTENTS

2.1 Backend

2.1.1 Configurations

Module `config`

This module defines different Config objects for different servers.

class `config.`**`Config`**

Base Configurations used for all servers.

Parameters

- **`SECRET_KEY`** (*`str`*) – secret key of the web application
- **`BASE_DIR`** (*`str`*) – base directory of the application
- **`SQLALCHEMY_TRACK_MODIFICATIONS`** (*`bool`*) – whether to track modification when using SQLAlchemy, *default:* `False`
- **`SQLALCHEMY_DATABASE_URI`** (*`str`*) – directory of the local SQL database, *default:* `SQLite database`
- **`MTURK_URL`** (*`str`*) – endpoint of Amazon Crowdsourcing Platform, *default:* `None`
- **`MTURK_SHOW_UP_URL`** (*`str`*) – link to where the project is uploaded (mainly in production environment), *default:* real page

`init_app`(*app*)

Application initialization

class `config.`**`DevelopmentConfig`**

Extends *`Config`*. Configurations used during development.

Parameters

- **`FLASK_ENV`** (*`str`*) – environment of the application, *default:* `development`
- **`DEBUG`** (*`bool`*) – whether to debug during running the application, *default:* `True`

- **SQLALCHEMY_DATABASE_URI** (*str*) – SQL database used for development
- **MTURK_URL** (*str*) – endpoint of Amazon Crowdsourcing Platform, used in Development environment, *default:* sandbox in us-east-1 region
- **AWS_ACCESS_KEY_ID** (*str*) – IAM AWS credentials - key id, *default:* `None`
- **AWS_SECRET_ACCESS_KEY** (*str*) – IAM AWS credentials - secret access key, *default:* `None`
- **MTURK_SHOW_UP_URL** – link to where the project is uploaded, in development environment, *default:* Sandbox

class config.**TestingConfig**

Configurations used during testing.

Parameters

- **DEBUG** (*bool*) – whether to debug during running the application, *default:* `False`
- **TESTING** (*bool*) – whether to set this application in testing environment, *default:* `True`
- **SQLALCHEMY_DATABASE_URI** (*str*) – SQL database used for testing
- **WTF_CSRF_ENABLED** (*bool*) – whether to enable CSRF Token for different input forms in HTML, *default:* `False`

config.**config**

Global variable for environment configurations. Can import this instead of configuration objects.

2 environments are available:

- For development `config['development']`
- For testing `config['testing']`

Default, `config['default']`, is development environment.

2.1.2 Web Application

Package `project`

This package defines the application.

project.__init__.**create_app**(*config_env*)

Generate application based on configuration environment

Parameters **config_env** (`config.Config`) – which configuration environment is used to generate application

Returns application

Return type Flask

`project.__init__.`**`init_extensions`**`(`*app*`)`

Bind each Flask extension to the Flask application instance. Configure Login manager for multi-login system.

Parameters **`app`** (Flask) – application

`project.__init__.`**`register_blueprints`**`(`*app*`)`

Register each Blueprints with the Flask application instance.

Parameters **`app`** (Flask) – application

2.1.2.1 Generators

Module `project.generator`

This module provides some classes to generate and handle different types of data during running the application. This can be used outside the application as it works independently.

`class` `project.generator.`**`BaseGenerator`**`(`*tuples*`)`

This class contains a base function *`get_frequency()`* to count occurencies of each items in all tuples.

Parameters **`tuples`** (*`list(tuple or set or list)`*) – list of tuples/lists/sets of items

`frequencies`

container in form of a dictionary to save all items and frequencies inside given lists of sets

Type collections.Counter

Examples

```
>>> tuples = [('A','B','C'), ('B','C','D'),('D','A','C')]
>>> base = BaseGenerator(tuples=tuples)
>>> base.frequencies
Counter({'C': 3, 'A': 2, 'B': 2, 'D': 2})
```

`get_frequency``(`*tuples*`)`

Count the number of tuples each item is in.

Parameters **`tuples`** (*`list(tuple or set or list)`*) – set of tuples

Returns frequencies of items in all tuples

Return type collections.Counter

class `project.generator.`**`DataGenerator`**(*num_iter=100*, *batch_size=20*, *minimum=5*)

Extend `BaseGenerator`. Create an object of input data for the survey based on input file(s).

Parameters

- **`num_iter`** (`int, optional`) – number of necessary iterations to generate tuples, *default:* `100`
- **`batch_size`** (`int, optional`) – size of a normal batch, *default:* `20`
- **`minimum`** (`int, optional`) – minimum size of a batch to be formed if the rest items do not meet the normal size, *default:* `5`

`items`

the unique given items

Type set

`tuples`

list of all unique generated tuples with the best results after all iterations

Type list

`batches`

all batches prepared for questionnaire

Type dict

`num_iter`

number of necessary iterations to generate tuples, *default:* `100`

Type int

`batch_size`

size of a normal batch, *default:* `20`

Type int

`minimum`

minimum size of a batch to be formed if the rest items do not meet the normal `batch_size`, *default:* `5`

Type int

`factor`

to decide the number of tuples to be generated - *n_tuples* = `factor` * *len(* `items` *)*, *default:* `2` if fewer than 10000 items

Type int or float

`tuple_size`

size of each tuple, *default:* `4` if fewer than 1000 items else `5`

Type int

Examples

```
>>> example = open('../examples/movie_reviews_examples.txt','rb
↪')
>>> data = DataGenerator()
>>> data.generate_items(example)
>>> data.generate_data()
>>> data.items # items read from input example
{'interesting', 'excited', 'annoyed', 'boring', 'aggressive',
↪'joyful', 'fantastic', 'indifferent'}
>>> data.tuples # tuples generated from the items (change each␣
↪time calling this function)
[['interesting', 'indifferent', 'excited', 'joyful'], [
↪'indifferent', 'boring', 'aggressive', 'joyful'], [
↪'interesting', 'fantastic', 'annoyed', 'indifferent'], [
↪'joyful', 'fantastic', 'annoyed', 'indifferent'], ['fantastic
↪', 'annoyed', 'aggressive', 'indifferent'], ['fantastic',
↪'boring', 'indifferent', 'joyful'], ['excited', 'boring',
↪'aggressive', 'interesting'], ['interesting', 'aggressive',
↪'annoyed', 'joyful'], ['interesting', 'fantastic', 'boring',
↪'aggressive'], ['excited', 'fantastic', 'indifferent', 'joyful
↪'], ['excited', 'boring', 'annoyed', 'joyful'], ['interesting
↪', 'fantastic', 'excited', 'indifferent'], ['excited',
↪'aggressive', 'annoyed', 'interesting'], ['fantastic', 'boring
↪', 'aggressive', 'annoyed'], ['interesting', 'fantastic',
↪'aggressive', 'joyful'], ['excited', 'boring', 'annoyed',
↪'indifferent']]
>>> data.batches # batches generated from the tuples (change␣
↪each time calling this function)
{1: [['interesting', 'indifferent', 'excited', 'joyful'], [
↪'indifferent', 'boring', 'aggressive', 'joyful'], [
↪'interesting', 'fantastic', 'annoyed', 'indifferent'], [
↪'joyful', 'fantastic', 'annoyed', 'indifferent'], ['fantastic
↪', 'annoyed', 'aggressive', 'indifferent']], 2: [['fantastic',
↪ 'boring', 'indifferent', 'joyful'], ['excited', 'boring',
↪'aggressive', 'interesting'], ['interesting', 'aggressive',
↪'annoyed', 'joyful'], ['interesting', 'fantastic', 'boring',
↪'aggressive'], ['excited', 'fantastic', 'indifferent', 'joyful
↪']], 3: [['excited', 'boring', 'annoyed', 'joyful'], [
↪'interesting', 'fantastic', 'excited', 'indifferent'], [
↪'excited', 'aggressive', 'annoyed', 'interesting'], [
↪'fantastic', 'boring', 'aggressive', 'annoyed'], ['interesting
↪', 'fantastic', 'aggressive', 'joyful'], ['excited', 'boring',
↪ 'annoyed', 'indifferent']]}
>>> data.get_frequency(data.tuples) # get frequency of each␣
↪item in all generated tuples
Counter({'indifferent': 9, 'fantastic': 9, 'interesting': 8,
↪'joyful': 8, 'aggressive': 8, 'annoyed': 8, 'excited': 7,
↪'boring': 7})
```

generate_batches()

Split the whole set of tuples into batches.

Returns

update all batches prepared for questionnaire (attribute `batches`).

Return type dict(int = list)

Raises **`ValueError`** – if there is no attribute `tuples`.

`generate_data`()

Generate data including tuples and batches. This method calls `generate_tuples()` and `generate_batches()`.

`generate_items`(*file_name*)

Read uploaded *txt*-file. Accept only one file each time.

Parameters **`file_name`** (FileStorage or io.BufferedReader) – uploaded file

Returns update list of items with this file (attribute `items`).

Return type list

`generate_tuples`()

Generate tuples, this is a reimplementation of *generate-BWS-tuples.pl* in source code.

The tuples are generated by random sampling and satisfy the following criteria:

1. no two items within a tuple are identical;
2. each item in the item list appears approximately in the same number of tuples;
3. each pair of items appears approximately in the same number of tuples.

Returns update list of all unique generated tuples with the best results after all (attribute `tuples`).

Return type list

Raises **`ValueError`** – if the number of `items` is fewer than `tuple_size`.

class `project.generator.`**`ScoreGenerator`**(*tuples*, *best*, *worst*)

Create an object to calculate the scores of given items based on annotations.

Parameters

- **`tuples`** (`list`) – list of tuples
- **`best`** (`list`) – list of items annotated as '**best**'
- **`worst`** (`list`) – list of items annotated as '**worst**'

`frequencies`

frequency of each item in all tuples

Type dict or collections.Counter

best

frequency of each item annotated as '**best**'

Type dict or collections.Counter

worst

frequency of each item annotated as '**worst**'

Type dict or collections.Counter

Examples

```
>>> tuples = [('A','B','C'), ('B','C','D'),('D','A','C')]
>>> best = ['A','B','A']
>>> worst = ['B','D','C']
>>> generator = ScoreGenerator(tuples, best, worst)
>>> generator.scoring()
[('A', 1.0), ('B', 0.0), ('C', -0.3333333333333333), ('D', -0.
↪5)]
```

scoring()

Calculate scores of the items using formula of Orme 2009.

Returns descendingly sorted list of tuples `(item, score)` based on scores

Return type list(tuple(str, float))

References

More about research with Best-Worst-Scaling

2.1.2.2 Models

Module `project.models`

This module defines the database tables used for the web application.

See also:

A quick example how to define a table with Flask_SQLALchemy

class `project.models.`**Annotator**(*keyword=None, name=None, project=None*)

Extend UserMixin and db.Model.

Store data of each (local) annotator from project.

id

automatically defined annotator-id

Type db.Integer

`keyword`
logged in keyword for annotator

Type db.String

`name`
pseudoname chosen by annotator to avoid more annotators having access to annotator system using the same keyword

Type db.String

`project_id`
id of the project the annotator takes part in

Type db.Integer

`project`
`many-to-one` relationship with `Project`

Type db.relationship

`batches`
`many-to-many` relationship with `Batch`

Type db.relationship

`get_id``()`
Override UserMixin.get_id() method to manage login in multi-login system.

`class` `project.models.`**`Batch`**`(`*size*, *keyword=None*, *hit_id=None*, *project=None*`)`
Extend db.Model.

Store data of each created batch from project.

`id`
automatically defined batch-id

Type db.Integer

`size`
batch size

Type db.Integer

`keyword`
keyword for this batch (only used in case of MTurk)

Type db.String

`hit_id`
endpoint to this batch in annotator system with option MTurk

Type db.String

`project_id`
id of this batch's project

Type db.Integer

project
`many-to-one` relationship with *Project*

Type db.relationship

class `project.models.`**Data**(*best_id=None*, *worst_id=None*, *annotator=None*, *tuple_=None*)

Extend db.Model.

Store data of each created tuple from project.

id
automatically defined data-id

Type db.Integer

best_id
id of item chosen as '**best**' in table *Item*

Type db.Integer

worst_id
id of item chosen as '**worst**' in table *Item*

Type db.Integer

anno_id
id of annotator who submits/saves this data (only in local annotator system)

Type db.Integer

tuple_id
id of the tuple this data uses

Type db.Integer

annotator
`many-to-one` relationship with *Annotator*

Type db.relationship

tuple_
`many-to-one` relationship with *Tuple*

Type db.relationship

class `project.models.`**Item**(*item*)

Extend db.Model.

Store data of each uploaded item from project.

id
automatically defined item-id

Type db.Integer

item
(raw) representation of item in string-format

Type db.String

class project.models.**Project**(*name*, *description*, *anno_number*, *best_def*, *worst_def*, *n_items*, *p_name*, *mturk=False*, *user=None*)

Extend db.Model.

Store data of each project uploaded by users.

id
automatically defined project-id

Type db.Integer

name
project name

Type db.String

description
project description

Type db.String

anno_number
number of expected annotations/annotators for this project

Type db.Integer

best_def
definition of '**best**'

Type db.String

worst_def
definition of '**worst**'

Type db.String

n_items
number of items in project

Type db.Integer

p_name
endpoint to this project

Type db.String

mturk
whether to upload this project on Mechanical Turk

Type db.Boolean

user_id
id of user this project belongs to

Type db.Integer

user
`many-to-one` relationship with *User*

Type db.relationship

class project.models.**Tuple**(*batch=None*)
Extend db.Model.

Store data of each created tuple from project.

id
automatically defined tuple-id

Type db.Integer

batch_id
id of this tuple's batch

Type db.Integer

batch
`many-to-one` relationship with *Batch*

Type db.relationship

items
`many-to-many` relationship with *Item*

Type db.relationship

class project.models.**User**(*username*, *email*, *password*)
Extend UserMixin and db.Model.

Store data of each user who uses the system to get the scores of their items calculated.

id
automatically defined user-id

Type db.Integer

username
username

Type db.String

email
user's email

Type db.String

password
use method generate_password_hash() to create user's hashed password

Type db.String

check_password(*password*)
Use check_password_hash() to check if given password from user and the password saved in table are the same.

Parameters **password** (*str*) – given password from user

Returns `True` if these two are the same, else `False`.

Return type bool

get_id()

Override UserMixin.get_id() method to manage login in multi-login system.

2.1.2.3 Validators

Module `project.validators`

This module provides validators for all forms used within all systems.

- Call the classes the same way with classes in Module wtforms.validators.
- Use functions mostly inside overwritten validate() or validate_<fieldname>().

class project.validators.**InputValid**(*model*, *field*, *message='Invalid object'*)

Make sure input data is valid.

Parameters

- **model** (db.Model) – table of database as model
- **field** (db.Column) – attribute inside the table used
- **message** (*str*) – message if this validator fails

Raises ValidationError if input data is invalid.

class project.validators.**NotEqualTo**(*other_fieldname*, *message=None*)

Use the same structure like default class EqualTo.

Make sure 2 fields are not the same.

Parameters

- **other_fieldname** (db.Column) – an another attribute of the table used
- **message** (*str*) – message if this validator fails

Raises ValidationError if this validator fails.

class project.validators.**Unique**(*model*, *field*, *message='This element already exists.'*)

Check if input data is unique.

Parameters

- **model** (db.Model) – table of database as model
- **field** (db.Column) – attribute inside the table used
- **message** (*str*) – message if this validator fails

Raises ValidationError if input data is not unique.

`project.validators.`**`allowed_file`**(*filename*, *allowed={'txt'}*)

Check if uploaded file(s) have/has the right extension.

Parameters

- **`filename`** (*`str`*) – name of the uploaded file
- **`allowed`** (*`set`*) – list of allowed extensions, *default:* `{'txt'}`

Returns `True` if this file is an allowed file, else `False`.

Return type bool

Examples

```
>>> allowed_file('example.txt')
True
>>> allowed_file('example.csv')
False
>>> allowed_file('example.csv', allowed=set(['csv']))
True
```

2.1.2.4 Annotator Subsystem

Subpackage `project.annotator`

This subpackage defines Annotator-System for annotators to annotate datas.

Main functions:

- Login
- View all batches
- Annotate batch

Two Blueprints define two annotator systems:

- Local: at `/annotator`
- With Mechanical Turk: at `/mturk`

Forms

Module `project.annotator.forms`

This module defines forms used inside of the Annotator-System.

`class` `project.annotator.forms.`**`AnnoCheckinForm`**(*formdata=<object object>*, **kwargs)

Extend FlaskForm. Define form for an annotator login-system.

keyword
keyword to log in

Type PasswordField

name
pseudoname given by annotator

Type StringField

validate()
Override validate().

Check if the person using this keyword is the first one who logged in by checking his given (pseudo)name.

class `project.annotator.forms.`**TupleForm**(*formdata=<object object>*, ***kwargs*)
Extend FlaskForm. Define form for a tuple with two choices for **best** and **worst** item.

best_item
answer for the question of best item

Type RadioField

worst_item
answer for the question of worst item

Type RadioField

Helper Functions

Module `project.annotator.helpers`

This module defines some helper functions to deal with problems of each subroute inside of the Annotator-System.

`project.annotator.helpers.`**batches_list**(*project='batch'*, *n_batches=5*)
Create buttons corresponding to number of batches inside the given project.

Parameters

- **project** (*str*) – name of the project, *default:* `batch`
- **n_batches** (*int*) – number of batches inside this project, *default:* `5`

Returns list of tuples `(project, batch_id, batch_name)`

Return type list

Example

```
>>> batches_list()
[('batch', 1, 'Batch 1'), ('batch', 2, 'Batch 2'), ('batch', 3,
↪'Batch 3'), ('batch', 4, 'Batch 4'), ('batch', 5, 'Batch 5')]
>>> batches_list(project='test', n_batches=3)
[('test', 1, 'Batch 1'), ('test', 2, 'Batch 2'), ('test', 3,
↪'Batch 3')]
```

Routes Management

Accounts

Module `project.annotator.account`

This module defines routes to manage accounts of annotators.

`project.annotator.account.`**`login`**`()`

Manage account login with keyword and (pseudo)name inside annotator system at `/annotator`.

Returns project site `/annotator/<p_name>` if account is valid.

Error: Error message emerges if keyword or name is invalid.

Annotations

Module `project.annotator.annotation`

This module defines routes to manage the annotations.

`project.annotator.annotation.`**`batch`**`(`*p_name*, *batch_id*`)`

Represent a batch within the project in the local system for an annotator to annotate at `/annotator/<p_name>/batch-<int:batch_id>`.

Parameters

- **`p_name`** (*`str`*) – name of the project
- **`batch_id`** (*`int`*) – id of the batch

Returns project site with all batches at `/annotator/<p_name>`, if the annotation is saved or the valid annotation is submitted.

Error: Error message emerges if invalid annotation is submitted.

`project.annotator.annotation.`**`hit`**(*p_name*, *hit_id*)

Represent a HIT within the project directed from MTurk for an annotator (turker) to annotate at `/mturk/<p_name>/<hit_id>`.

Parameters

- **`p_name`** (*`str`*) – name of the project
- **`hit_id`** (*`str`*) – id of the HIT

Returns keyword of the HIT, if valid annotation is submitted.

Error: Error message emerges if invalid annotation is submitted.

Views

Module `project.annotator.views`

This module defines routes to manage views for annotators.

`project.annotator.views.`**`project`**(*p_name*)

View project with all batches corresponding to given keyword at `/annotator/<p_name>`.

Parameters **`p_name`** (*`str`*) – project name

Returns Status of each batch in project if they are submitted by this annotator or not.

2.1.2.5 User Subsystem

Subpackage `project.user`

This subpackage defines User-System.

Main functions:

- Signup-Login
- View profiles, projects
- Upload projects
- Get outputs

A Blueprint defines this user system: under `/user`

Forms

Module `project.user.forms`

This module defines forms used inside User-System.

class `project.user.forms.`**`LoginForm`**(*formdata=<object object>*, ***kwargs*)

Extend FlaskForm. Define login form.

`username`

username

Type StringField

`password`

password of user

Type PasswordField

`remember`

whether to remember data of this account

Type BooleanField

`validate`()

Override validate().

Check if given password is valid with given valid username.

class `project.user.forms.`**`ProjectInformationForm`**(*formdata=<object object>*, ***kwargs*)

Extend FlaskForm. Define form to upload a project.

`upload`

files of items

Type MultipleFileField

`name`

project name

Type StringField

`description`

project description

Type TextAreaField

`anno_number`

number of expected annotators for this project

Type IntegerField

`best_def`

Definition of '**best**' in this project

Type StringField

`worst_def`

Definition of '**worst**' in this project

Type StringField

mturk
Whether to upload this project on Mechanical Turk

Type BooleanField

aws_access_key_id
IAM AWS access key id, *only provide if set* `mturk` == `True`

Type StringField, optional

aws_secret_access_key
IAM AWS secret access key, *only provide if set* `mturk` == `True`

Type StringField, optional

keywords
keywords to describe the project on Mechanical Turk, *default:* `e.g. quick, sentiment, labeling`, *only provide if set* `mturk` == `True`

Type StringField, optional

reward
reward for an annotator/turker to annotate a HIT, *default:* `0.15`, *only provide if set* `mturk` == `True`

Type StringField, optional

lifetime
duration of the project to be availabe on Mechanical Turk, *default:* `1`, *only provide if set* `mturk` == `True`

Type IntegerField, optional

lifetimeunit
duration unit for `lifetime`, *default:* `month`, *only provide if set* `mturk` == `True`

Type SelectField, optional

hit_duration
duration of a HIT annotation for each annotator, *default:* `1`, *only provide if set* `mturk` == `True`

Type IntegerField, optional

duration_unit
duration unit for `hit_duration`, *default:* `month`, *only provide if set* `mturk` == `True`

Type SelectField, optional

validate_upload(*self*)
Validate uploaded files `upload`.

Check if uploaded file(s) has/have one of the allowed extensions (*default:* `{'txt'}`).

class `project.user.forms.`**`RegisterForm`**(*formdata=<object object>, **kwargs*)

Extend FlaskForm. Define registration form.

`username`

username

Type StringField

`email`

user email

Type StringField

`password`

user password

Type PasswordField

`validate_username`(*self*)

Validate `username`.

Username is not allowed to have space or special character.

Helper Functions

Module `project.user.helpers`

This module provides some helper functions to deal with problems of each subroute inside User-System.

`project.user.helpers.`**`convert_into_seconds`**(*duration, unit*)

Convert given duration and unit into seconds.

Parameters

- **`duration`** (`int`) – duration
- **`unit`** (`str`) – acronym for duration unit, use: `m` - *month*, `d` - *day*, `h` - *hour*, `min` - *minute*

Returns converted duration in seconds

Return type int

Examples

```
>>> convert_into_seconds(2,'m')  # month
5184000
>>> convert_into_seconds(2,'d')  # day
172800
>>> convert_into_seconds(2,'h')  # hour
7200
```

(continues on next page)

(continued from previous page)

```
>>> convert_into_seconds(2,'min') # minute
120
```

project.user.helpers.**generate_keyword**(*chars=None*, *k_length=None*)
Generate keyword for annotators and batches.

Parameters

- **chars** (*str*) – type of characters used to generate keyword, *default:* `string.ascii_letters+string.digits`
- **k_length** (*int*) – length of the keyword, *default:* `random.randint(8,12)`

Returns generated keyword

Return type str

Examples

```
>>> generate_keyword()
'WfgdmWPZ7fx'
>>> generate_keyword(chars=string.digits)
'15151644097'
>>> generate_keyword(chars=string.ascii_letters, k_length=3)
'RIF'
```

project.user.helpers.**is_not_current_user**(*user*, *current_name*)
Inside the User-System, if a user is logged in and authenticated, this user is not allowed to see profile of an another user by typing the route! If this meets the conditions which means the current user tries to access to the account of an another user, this current user will be redirected to the login page and asked to log in with the used username.

Parameters

- **user** (werkzeug.LocalProxy) – this is actually the attribute current_user
- **current_name** (*str*) – name of the other user to be typed in the route

Returns `True` if this is not the current user else `False`

Return type bool

project.user.helpers.**upload_file**(*files*)
Upload all files and store in container for later use.

Parameters **files** (list(FileStorage)) – list of uploaded files

Returns object that contains list of items, batches and tuples if the number of items meeth the required condition

Return type *generator.DataGenerator*

Warning:

- Returns 1 if there are items but the number is fewer than 5.
- Returns None if there is no item at all.

Routes Management

Accounts

Module `project.user.account`

This module defines routes to manage accounts of users.

`project.user.account.`**`login`**`()`

Manage a user login within the user system at `/user/login`.

Returns user profile page at `/user/<some_name>`, if account is valid.

Error: Error message emerges if there is invalid username or password.

`project.user.account.`**`logout`**`()`

Manage an account logout at `/user/logout`.

Returns User Homepage at `/user`.

Error: Error message emerges if there is no currently logged in user.

`project.user.account.`**`signup`**`()`

Manage signup of a user account within the user system at `/user/signup`.

Returns user homepage if valid account is created.

Error: Error message emerges if there is invalid username or email.

Inputs

Module `project.user.inputs`

This module defines routes to manage input new inputs of projects from users.

`project.user.inputs.`**`upload_project`**`()`

Provide information of a new project from user at `/user/upload-project`.

Returns user profile page at `/user/<some_name>` if new valid project is submitted.

Note: Upload project on Mechanical Turk Platform or use local annotator system.

Error: Error message emerges if there are invalid fields or there is no logged in user.

Outputs

Module `project.user.outputs`

This module defines routes to manage outputs for users.

`project.user.outputs.`**`get_keywords`**(*some_name*, *p_name*)

Collect keywords for annotators and save at `/user/<some_name>/<p_name>/keywords.txt`.

Parameters

- **`some_name`** (*`str`*) – username
- **`p_name`** (*`str`*) – project name as endpoint

Returns Keywords for annotators within the project in .txt-file if this project is on the local system, else no keyword (this project is created on Mechanical Turk).

Error: Error message emerges if user of this project is not logged in.

`project.user.outputs.`**`get_report`**(*some_name*, *p_name*)

Collect the annotated data and save at `/user/<some_name>/<p_name>/report.txt`.

Parameters

- **`some_name`** (*`str`*) – user name
- **`p_name`** (*`str`*) – project name

Returns Report in .txt-file with all the submitted data.

Error: Error message emerges if user of this project is not logged in.

`project.user.outputs.`**`get_scores`**(*some_name*, *p_name*)

Collect the annotated data, calculate the BWS-score for each item and save them as `/user/<some_name>/<p_name>/scores.txt`.

Parameters

- **`some_name`** (*`str`*) – user name
- **`p_name`** (*`str`*) – project name

Returns Items with scores in .txt-file, if at least one batch or HIT is submitted, else no result.

Error: Error message emerges if user of this project is not logged in.

Views

Module `project.user.views`

This module defines routes to manage views for users.

`project.user.views.`**`main`**`()`

View homepage of user system at `/user`.

`project.user.views.`**`profile`**`(`*some_name*`)`

View profile of an account within user system at `/user/<some_name>`.

Parameters **`some_name`** (*`str`*) – name of the user

Returns Project site if account is validated.

Note: Show table of all user projects.

Error: Error message emerges if there is no logged in user.

`project.user.views.`**`project`**`(`*some_name*, *p_name*`)`

View user project at `/user/<some_name>/<p_name>`.

Parameters

- **`some_name`** (*`str`*) – user name.
- **`p_name`** (*`str`*) – project name.

Returns project information.

Error: Error message emerges if there is no logged in user.

2.2 Frontend

2.2.1 Templates

Different templates within web application.

2.2.1.1 Main System

start.html

Template for web homepage at `/`.

questions.xml

Template for keyword page on Mechanical Turk.

2.2.1.2 Annotator Subsystem

batch.html

Template for each batch/HIT page with multiple choice questions at `/annotator/<project_name>/batch-<batch_id>` or `/mturk/<project_name>/<hit_id>`.

index.html

Template for annotator homepage at `/annotator`.

project.html

Template for project page with all batches at `/annotator/<project_name>`.

2.2.1.3 User Subsystem

index.html

Template for user homepage at `/user`.

login.html

Template for user login page at `/user/login`.

profile.html

Template for user profile page with all projects at `/user/<username>`.

project.html

Template for each project page with its information at `/user/<username>/<project_name>`.

signup.html

Template for user signup page at `/user/signup`.

upload-project.html

Template for the page to create a new project at `/user/upload-project`.

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX