Practice of Efficient Data Collection via Crowdsourcing: Aggregation, Incremental Relabelling, and Pricing

Setting up and running label collection projects

Instruction

Yandex Toloka for requesters:
https://ya.cc/tlk

Tutorial slides:
https://research.yandex.com/tutorials/crowd/wsdm-2020
# Table of contents

Suggested pipeline ........................................................................................................................................... 3
Project creation. Main steps .................................................................................................................................. 4
Key types of instances in Yandex.Toloka ............................................................................................................. 4
**Project #1** Does the photo contain an item? .................................................................................................... 5
Project creation .................................................................................................................................................. 6
Pool creation ...................................................................................................................................................... 9
Preparing and uploading a file with tasks ........................................................................................................... 11
Receiving responses ........................................................................................................................................... 14
**Project #2** Find a similar item in an online store .............................................................................................. 15
Project creation ................................................................................................................................................ 16
Pool creation ..................................................................................................................................................... 19
Preparing and uploading a file with tasks ........................................................................................................... 22
**Project #3** Does the item found look similar to the initial one? ........................................................................ 23
Project creation (similar to the 1st project) ....................................................................................................... 24
Pool creation ..................................................................................................................................................... 27
Preparing and uploading a file with tasks ........................................................................................................... 29
Receiving responses ........................................................................................................................................... 33
Upload reviewed results ....................................................................................................................................... 34
Review assignments in the interface (another way of results validation) ............................................................. 35
**Project #4** Which item is more similar? ........................................................................................................... 36
Project creation ................................................................................................................................................ 37
Pool creation ..................................................................................................................................................... 41
Preparing and uploading a file with tasks ........................................................................................................... 43
Receiving responses ........................................................................................................................................... 44
**Appendix:** Expanded code of the projects .................................................................................................... 45
Suggested pipeline

Tasks for performers:

Project #1
- Does a photo contain a specific item?
  - If yes

Project #2
- Find a similar item in the online store
  - If yes
  - If no

Project #3
- Is the found item (project 2) similar to the initial one? (post-verification)
  - If no
  - If yes

Project #4
- Which of the found items is more similar to the initial one?
  - OR

a set of photos

photos with items

photos with incorrect similar items

photos with correct similar items

Reject the result in Project #2
Do not pay for it

Accept the result in Project #2
Pay for it

photos w/ best substitute
Project creation. Main steps

Create a project

Add a task pool

Upload file with tasks

Start the pool

Receive and aggregate the results

Define input and output data, task interface and instruction

Set up user filters and quality control

Key types of instances in Yandex.Toloka

**Project**
- Defines the structure of tasks
- Defines how to perform them

**Pool**
- Is a batch of tasks
- Defines access of performers

**Task**
- A particular input data
- Results for it from performers

- Configure in a project:
  - Input and output data types
  - Task interface
  - Task instruction

- Configure in a pool:
  - Performer filters
  - Quality control mechanisms
  - Overlap settings
Project #1 Does the photo contain an item?

Important: Before you start using Toloka, make sure that the English language is selected.
Project creation

1. Click the button + Create project

2. Choose the **Image classification** template.

3. Enter a clear project name and description. **Important:** It will be visible for **real people.**
4. Write short and simple instructions. To include an image in the instruction just paste the link from the dataset provided by pressing button.

5. Define parameters for the input and output data:

   - The “image” input data field with the link type will be used to pass the image links to the performers. You will be able to upload the file with links to the pool later.
   - The "result" field will be used to receive performer’s responses.
   - The "like" field in the template is used to pass the response to the question “Do you like the photo?”. Our project doesn’t require this checkbox, so you don’t need an output field for it. Let’s remove it.
6. Create the task interface in the HTML block.
   • Delete the line with the checkbox component:
     ```html
     {{field type="checkbox" name="like" label="Do you like the photo?" hotkey="q"}}
     ```
   • Add a question: does the image include a certain object? Example:
     ```html
     <div>Are there shoes in the picture?</div>
     ```
   • Replace "label" with your response options (change “Good” to “Yes” and “Bad” to “No”). Example:
     ```html
     {{field type="radio" name="result" value="Yes" label="Yes"}}
     {{field type="radio" name="result" value="No" label="No"}}
     {{field type="radio" name="result" value="404" label="Picture not found"}}
     ```

7. Leave JavaScript and CSS block unchanged.
8. Click the **Preview** button to see the performer’s view of the task.
   *You will see tasks with standard pictures on the page. You will set the number of tasks per page when configuring a pool.*
9. Select the radio buttons in the preview and make sure that the task can be completed.
10. Click **Save** button to save the project.
    **Note.** To edit project parameters, click the button in the list of projects or **Project actions → Edit** on the project page.
Pool creation

1. Click **Add pool**.
2. Give the pool any convenient name and description. You are the only one who can see them.
3. Specify the **pool parameters**:
   - Set the price per task page (for example, $0.01).
   - Overlap. This is the number of users who will complete the same task. For example, 3.
   - Time allowed for completing a task page (for example, 600 seconds).

![Pool creation interface](image)

4. Set up user **filters**.
   - Select English-speaking performers using the **Language = English** filter.
   - Optionally, specify the percentage of top-rated performers in the **Speed / Quality ratio**. **Important**: This can slow down pool completion.
5. Set up **quality control: Golden Set aka Control tasks.** Ban performers who give incorrect responses to control tasks. Example:

```plaintext
This rule will be triggered when the performer completes 3 control tasks in the pool. If the performer gives at least 3 responses to the control tasks and the percentage of correct responses is less than 60%, they lose access to the project for 10 days. If the percentage of correct responses is over 60%, the performer can pass to the next task page. The rule will be triggered after the next control task.

Optionally, add **other quality control rules.**
```

6. Save the pool.
Preparing and uploading a file with tasks

1. Download TSV file with images by link or QR-code on the card with your login/password that you were provided at the beginning of the practice session.
2. **Upload pool tasks** from this file.
   Important: If you changed the name of the input field, change it in the file as well

Select **Smart mixing** and specify the number of tasks per page.
For example: 9 main tasks and 1 control task.

3. **Create control tasks**.

   **Note.** Control tasks are tasks with the correct response known in advance. They are used to track the performer's quality of responses. The performer's response is compared to the response you provided. If they match, it means the performer answered correctly.
   
   - Click **Edit** → **Create control tasks**.
Enter correct responses for your control tasks. In small pools, control tasks should account for 10-20% of all tasks.

**Tip.** Make sure to include different variations of correct responses in equal amounts. Open the Control tasks → Distribution of correct responses for control tasks tab.
- Save the markup and check the number of control tasks.

4. Start the pool.

**Important.** Remember that real Toloka performers will complete the tasks. Double check that everything is correct with configuration of your project before you start the pool.
Receiving responses

Disclaimer: Aggregation takes from 5 to 20 minutes. During this time, you can start configuring your next project. Refresh the Operations page to check progress.

1. Wait until the pool is completed. Refresh the pool page to check progress.
2. Click the arrow next to the Download results button and run aggregation using the Dawid-Skene model.

3. Go to the operations list and wait until aggregation finishes.
   Note. During this time, you can start working on your next project. Refresh the Operations page to check progress.

4. When aggregation is complete, download the TSV file with the results.

5. Use this file to prepare data for Project #2.
Project #2 Find a similar item in an online store

Important: If you just started using Toloka, make sure that the English language is selected.
Project creation

Disclaimer: In this particular example we ask the performers to search for the clothing items on ASOS. However, you can use any other online store. In this case you must change ASOS to the store of your choice in HTML and JS fields of the project.

1. Click the + Create project button and choose the Product photo search template.

2. Enter a clear project name and description. It will be shown to performers.

<table>
<thead>
<tr>
<th>PROJECT NAME</th>
<th>Find same or similar shoes in an online store</th>
</tr>
</thead>
<tbody>
<tr>
<td>DESCRIPTION</td>
<td>Go to the online shop and provide the link and the image of the same or similar pair of shoes</td>
</tr>
<tr>
<td>INSTRUCTIONS</td>
<td>Look at the clothes a person is wearing in the picture. Then go to ASOS [link provided] and search for the same pair of shoes (it should be similar at least in category, style, color and length).</td>
</tr>
</tbody>
</table>

3. Write short and simple instructions.
4. Define specifications for the input and output data:

   Input:
   - The “image” input data field with the link type will be used to pass the image links to the performers.

   Output:
   - In the output field add “found_image”, to give the performers space to upload the image.
   - In the output field add “button” to check if the performer actually went to the required website.
   - In the output field add “found_link”, here we will check whether the pattern of the url from the required website (a.g. online store, ASOS) matches the pattern of the submitted link. If you have chosen a different store then change "pattern": "https://www.asos.com/.*", to "pattern": "your_store.*"
The code for specifications is:

Input data:
{
"image": {
"type": "url",
"hidden": false,
"required": true
}
}

Output data:
{
"button": {
"type": "boolean",
"hidden": false,
"required": true,
"allowed_values": [true]
},
"found_link": {
"type": "string",
"hidden": false,
"pattern": "https://www.asos.com/.*",
"required": true
},
"found_image": {
"type": "file",
"hidden": false,
"required": true
}
}

5. Create the task interface.

The editor for data search is automatically included in the template but you still need to add extra lines in the interface to adjust the template to your needs.

- **Delete the whole HTML code** in the template, and instead add the following code to show the initial item to the performers:

```html
{{img src=image width="50%" height="400px"}}<div class='answers'>
<p>Find the same <b>shoes</b> on ASOS</p>{{field type="button-clicked" name="button" label="ASOS" href="https://www.asos.com" action=true}}
<p>Shoes must be the same color and the same style.</p>
<p>Paste the link here</p>{{field width="100%" type="input" name="found_link"}}
<p>Upload the image here</p>{{field width="100%" type="file-img" name="found_image" preview=true}}
</div>
</div>
```
6. Now we need to check whether performer is going to submit a valid link and an image. To check it, **DO NOT delete any of JS code but change** the following JS block on the 5th line

```javascript
if (!solution.output_values.image && !solution.output_values.no_image) {
    to

    if (!solution.output_values.found_image) {
```

Be extra careful while changing the code! In case you are having trouble previewing call our team or check the expanded code in Appendix.

7. Add the following code in CSS field to set the images sizes proportionally:

```css
.task {
    display: block;
    height: 500px;
    width: 800px;
}
.img {
    float: left;
    width: 50%;
}
.answers {
    float: left;
    width: 40%;
    margin: 5%;
}
```

8. Click the **Preview** button to see the performer’s view of the task. In this particular case you will not be able to submit the assignment in **Preview** as the image you will be trying to upload cannot be uploaded while the pool is still closed.

9. Click **Save** button to save the project.
Pool creation

1. Click **Add pool**.

2. Give the pool any convenient name and description. You are the only one who can see them.

3. Specify the **pool parameters**:
   - Price per task page (for example, $0.03),
   - Overlap. This is the number of users who will complete the same task. Put overlap at least 3 so that the performers can bring you different links and images.
   - Time allowed for completing a task page (for example, 300 seconds).
   - **Turn on the Non-automatics acceptance** option and enter the number of days for checking in the **Deadline** field (for example, 7).

4. Set up user **filters**.
   - Select English-speaking performers using the **Language = English** filter.
   - Create the "**Found_shoes**" skill that will be assigned to users after they complete the pool tasks. You will use this skill to prevent these users from checking tasks in the next project. Click **Create skill**.
• Enter the skill name and add a description if desired. You are the only one who will see it. Leave the skill private, as it is by default and click Add.

Optionally, specify the percentage of top-rated performers in the Speed / Quality ratio.

**Important:** This can slow down pool completion.

5. Set up quality control:
   • Resend the rejected tasks for completion. Add the Recompletion of rejected assignments:

   - Submitted responses. Add a rule to mark users who completed at least one task in the pool.
Tip. If the skill you created doesn’t appear in the drop-down list, save the pool, and then open it for editing again.

- Optionally, add other quality control rules.
  
  Tip. Control tasks and majority vote are not used in this type of project, because the links provided by the performers must exactly match the reference, which is practically impossible.

6. Save the pool.
Preparing and uploading a file with tasks

1. Open the file with aggregated results from the project #1.

2. Select only images suitable for highlighting (OK answers or another value if you have changed it in the "result" field). Use a text editor or a spreadsheet editor.

3. Copy the column with the selected links to a new page or document and give a name to the INPUT:image column.
   
   **Important:** If you changed the input field name in the project to something other than “image”, change the name in the file as well: INPUT:<your field name>.

4. Save the file in TSV format.

5. **Upload the file** to the pool by selecting Set manually. Set 1 task per page.

6. Start the pool.
Project #3 Does the item found look similar to the initial one?

Important: If you just started using Toloka, make sure that the English language is selected.
Project creation (similar to the 1st project)

1. Click the button **Create project**

   ![Yandex Toloka](image1)

2. Choose the **Image classification** template.

   ![Templates](image2)

3. Enter a clear project name and description. **Important:** It will be visible for **real people**.

   ![Project Name](image3)

4. Write short and simple instructions.
5. Define parameters for the **input and output data**:

   **Input:**
   - The “image” input data field with the "url" type will be used to pass the initial image links to the performers.
   - The "found_link" field with the "url" type will allow performers can go to the website.
   - The "assignment_id" field with the "string" type, will be used to pass the number of the completed task.
Output:
- Leave “result” as it is.

The code for specifications is:

Input data:
```
{
  "image": {
    "type": "url",
    "hidden": false,
    "required": true
  },
  "found_link": {
    "type": "url",
    "hidden": false,
    "required": true
  },
  "assignment_id": {
    "type": "string",
    "hidden": true,
    "required": true
  }
}
```

Output data:
```
{
  "result": {
    "type": "string",
    "hidden": false,
    "required": true
  }
}
```

6. **Delete the whole HTML code** in the template, and add the following one.

```
{{img src=image height="400px"}}
{{iframe src=found_link height="600px"}}
<p>Check that the uploaded image matches the product in the store.</p>
<!--{button label="Check the item" href=found_link action=true}-->
<p>Are these <b>shoes</b> similar to each other?</p>
<p>Shoes must be the same color and the same style.</p>
{{field type="radio" name="result" value="Yes" label="Yes"}}
{{field type="radio" name="result" value="No" label="No"}}
```

7. Leave the JS block unchanged.
8. In the CSS block paste the following:

```
.task {
  display: block;
}
.img {
  float: left;
  width: 50%;
}
.iframe {
  float: left;
  width: 50%;
}
```
9. Click the **Preview** button to see the performer’s view of the task.

![Image of a cat on a computer screen]

10. Select the radio buttons in the preview and make sure that the task can be completed.

11. Click **Save** button to save the project.

   **Note.** To edit project parameters, click the button in the list of projects or **Project actions** → **Edit** on the project page.
Pool creation

1. Click **Add pool**.
2. Give the pool any convenient name and description. You are the only one who can see them.
3. Specify the **pool parameters**:
   - Set the price per task page (for example, $0.01).
   - Overlap. Specify 3.
   - Time allowed for completing a task page (for example, 300 seconds).
   - Keep task order for your convenience.

![Pool parameters interface](image)

4. Set up user **filters**.
   - Select English-speaking performers using the "Language = English" filter. Prevent performers who completed previous tasks from checking this one. To do this, set a filter with the **"Found_shoes" skill**:
     - The **"Found_shoes" skill = absent (empty field)**

![User filter interface](image)

Optionally, specify the percentage of top-rated performers in the **Speed / Quality ratio**. **Important**: This can slow down pool completion.
5. Set up **quality control**: **Golden Set aka Control tasks**. Ban performers who give incorrect responses to control tasks. Example:

**Fast responses.** You can ban the performers who suspiciously fast responses. This way you can get rid of cheaters in your pool. Example:

Optionally, add **other quality control rules**.

6. Save the pool.
Preparing and uploading a file with tasks

1. Wait until the pool of project #2 on “finding similar shoes” is completed.
2. Open the pool page in Project #2 and click the Download results button

- Clear the Accepted checkbox and select Submitted.
- Clear link, user ID, status, start time and Separate assignments with empty row checkboxes. This will give you a list of unreviewed tasks.

3. Keep and rename the following columns:
   - Keep the name of the "INPUT:image" column as it is.
   - Change the name of the "OUTPUT:found_link" column to "INPUT:found_link". To check this image for correctness in project 3.
   - Change the "ASSIGNMENT:assignment_id" column name to INPUT:assignment_id" to later track and match the assignment number.

**Make sure you have the columns exactly as below**

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>INPUT:image</td>
<td>INPUT:found_link</td>
<td>INPUT:assignment_id</td>
</tr>
</tbody>
</table>
Save the file in TSV format.

4. Open the pool page in Project #3.
5. Upload the file to the pool by selecting Smart mixing. Set the number of main and control tasks per page (for example, 9 and 1).

6. Create control tasks.

*Note. Control tasks are tasks with the correct response known in advance. They are used to track the performer's quality of responses. The performer's response is compared to the response you provided. If they match, it means the performer answered correctly.*

Click **Edit → Create control tasks.**
- Check the "result" output field that is used to match the user response to the control one, select the response and click **Save and go to next**.

Enter correct responses for your control tasks. In small pools, control tasks should account for 10-20% of all tasks.

**Tip.** Make sure to include different variations of correct responses in equal amounts. Open the **Control tasks → Distribution of correct responses for control tasks** tab.
• Save the markup and check the number of control tasks.

7. Start the pool.

**Important.** Remember that real Toloka performers will complete the tasks. Double check that everything is correct with configuration of your project before you start the pool.
Receiving responses

Disclaimer: Aggregation takes from 5 to 20 minutes. During this time, you can start configuring your next project. Refresh the Operations page to check progress.

1. Wait until the pool is completed.
2. Click the arrow next to the **Download results** button and run aggregation using the **Dawid-Skene model**.

3. Go to the operations list and wait until aggregation finishes.

4. Download the responses.
Upload reviewed results

As you set post verification in the pool settings in Project #2, you need to check the performers' responses within the time limit set in the Deadline field.

1. Open the file with aggregated results in a spreadsheet editor.

2. Add the following columns:
   - "ACCEPT:verdict" — The result of verification.
   - "ACCEPT:comment" — Comments for performers if responses were rejected (for example, which part of the instructions wasn't followed).

3. Change the name of the "INPUT:assignment_id" column to "ASSIGNMENT:assignment_id".

4. Delete all other columns

5. Fill in the "ACCEPT:verdict" and "ACCEPT:comment" columns:
   - If the aggregated result for the task is OK, put "+" then the task will be accepted.
   - If the result is BAD or 404, put "-" then the task will be rejected. Enter the reason for rejection in the "ACCEPT:comment" field. For example: The item provided is incorrect or improper.

6. Now you can delete the other columns. Save the edited TSV file.

```
<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASSIGNMENT:assignment_id</td>
<td>ACCEPT:verdict</td>
<td>ACCEPT:comment</td>
</tr>
<tr>
<td>00009c4245--5e27381d139bba</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>00009c4245--5e27337bb0e86f7-</td>
<td>-</td>
<td>The item provided is incorrect</td>
</tr>
<tr>
<td>00009c4245--5e2737692f3cb8c+</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>00009c4245--5e2739bcb0e86f6+</td>
<td>+</td>
<td></td>
</tr>
</tbody>
</table>
```

7. Open the pool page in Project #2.
8. Click Review assignments on the pool page above the progress bar.
9. Click Upload review results.

10. Select the file and upload it to Toloka.
11. Check that all tasks have changed their status to accepted or rejected.
12. You rejected tasks and set up the rule to send them for re-completion while configuring Project 2.

If have enough time you can do as many more reiterations as needed in order to receive as much clean data as you can. The steps are the following: The pool will open again, and these tasks will be resent to other performers. After the pool is marked up, download the new results and submit them for review. Download the reviewed results. Repeat these steps until all the images from the second project are correctly marked up.

But if you do not have enough time, move on to the next project and you can complete the reiterations later at your own pace.

Review assignments in the interface (another way of results validation)

You can also review assignments by yourself and see the results of the crowdsourcing pipeline that you have created.
1. Open the pool page in Project #2.
2. Click the Review assignments button on the pool page.

- Choose an assignment then click Accept or Reject.
- For rejected assignments, enter a comment (explain why you decline it).
Important: Before you start using Toloka, make sure that the English language is selected.
Project creation

1. Click the button + Create project

2. Choose the Side-by-side image comparison template.

1. Enter a clear project name and description. Important: It will be visible for real Toloka performers.
2. Write short and simple instructions. Example:

11. Define parameters for the input and output data:

- The “image” field is the initial image from the dataset.
- The “left_link” field is a link that the performer provided to match the item from the initial image.
- The “right_link” field is another link the performer provided.

You will be able to upload the file with links to the pool later.
- The “result” field will be used to receive performer’s responses.

The code for specifications is:

Input data:
12. Create the task interface in the HTML block.

```html
<div class="header">
  <div class="left caption">
    {{button label="Go to site" href=left_link size="L"}}
    <p class="url">{{left_link}}</p>
  </div>
  <div class="right caption">
    {{button label="Go to site" href=right_link size="L"}}
    <p class="url">{{right_link}}</p>
  </div>
</div>

{{img src=image}}

<div class="content clearfix">
  <div class="left page">
    {{iframe src=uploaded_link_left width="100%" height="700px" real-size=true screenshot=true}}
  </div>
  <div class="right page">
    {{iframe src=uploaded_link_right width="100%" height="700px" real-size=true screenshot=true}}
  </div>
</div>

<div class="footer">
  {{field type="radio" name="result" label="The left one is better" value=result_left hotkey="1"}}
  {{field type="radio" name="result" label="The right one is better" value=result_right hotkey="2"}}
</div>
```
13. **DO NOT delete any of JS code** but add the following JS block right before `OnRender`

```javascript
getTemplateData: function() {
    var data = TolokaHandlebarsTask.prototype.getTemplateData.apply(this, arguments),
        input = this.getTask().input_values,
        left_link = input.left_link,
        right_link = input.right_link,
        uploaded_link_left = '',
        uploaded_link_right = ''
    if (Math.floor(Math.random() * 2)) {
        uploaded_link_left = left_link
        uploaded_link_right = right_link
    } else {
        uploaded_link_left = right_link
        uploaded_link_right = left_link
    }
    data.uploaded_link_left = uploaded_link_left;
    data.uploaded_link_right = uploaded_link_right;
    data.result_left = uploaded_link_left;
    data.result_right = uploaded_link_right;

    return data;
},
```

14. In the CSS block add:

```css
.task {
    display: block;
    text-align:center;
}
.header {
    overflow: hidden;
    background-color: #FFCC00;
}
.caption {
    width: 50%;
}
.url {
    white-space: nowrap;
    overflow: hidden;
    text-overflow: ellipsis;
    max-width: calc(100% - 182px);
    display: inline-block;
    vertical-align: bottom;
}
.button {
    margin: 10px;
    max-width: 182px;
}
.button__label {
    white-space: nowrap;
    overflow: hidden;
    text-overflow: ellipsis;
    max-width: 150px;
}
.content {
```
15. Click the **Preview** button to see the performer’s view of the task. 
*You will see standard pictures on the page.*

16. Select the radio buttons in the preview and make sure that the task can be completed. 
17. Click **Save** button to save the project. 
   **Note.** To edit project parameters, click the button in the list of projects or **Project actions → Edit** on the project page.
Pool creation

1. Click **Add pool**.
2. Give the pool any convenient name and description. You are the only one who can see them.
3. Specify the **pool parameters**:
   - Set the price per task page (for example, $0.01).
   - **Overlap.** This is the number of users who will complete the same task. Put a larger number in this task. **For example, 10.**
   - Time allowed for completing a task page (for example, 300 seconds).

![Pool creation interface](image)

4. Set up user **filters**.
   - Select English-speaking performers using the **Language = English** filter.

![User filter interface](image)

- Optionally, specify the percentage of top-rated performers in the **Speed / Quality ratio.** **Important:** This can slow down pool completion.
5. Set up **quality control**:

   **Fast responses.** You can ban the performers who suspiciously fast responses. This way you can get rid of cheaters in your pool. Example:

![Fast responses](image)

   Optionally, add **other quality control rules**.

6. Save the pool.
Preparing and uploading a file with tasks

1. Take the downloaded TSV file with validated responses from Project 3.
2. Now you need to generate pairs for each INPUT:image so that you will be able to compare two found images with the initial one and decide which one is more similar than another.

   You can either generate the pairs by hand, using MS Excel or any other editor or you can automatically do it. We recommend using Python and Jupyter Lab.

   You can consult with our results
   https://tlk.s3.yandex.net/wsdm2020/SbS_Toloka_prep&aggr_data.ipynb

3. **Upload pool tasks** from this file.
   Important: If you changed the name of the input field, change it in the file as well

   ![Image](image.png)

   Upload the file to the pool by selecting **Set manually** and specify the number of tasks per page.
   You can experiment with the number of tasks!

4. Start the pool.
   Important. Remember that real Toloka performers will complete the tasks. Double check that everything is correct with configuration of your project before you start the pool.
Receiving responses

1. Wait until the pool is completed. Refresh the pool page to check progress.
2. Download the accepted results. Select the URLs, user IDs and assignment IDs like in the picture below. **Do not forget to clear “Separate assignments with empty row” box!**

![Download results screenshot](image)

3. Try to run Bradley Terry model on these results (you can consult with our results [https://tlk.s3.yandex.net/wsdm2020/SbS_Toloka_prep&aggr_data.ipynb](https://tlk.s3.yandex.net/wsdm2020/SbS_Toloka_prep&aggr_data.ipynb))

**GOOD LUCK!!!**
Appendix: Expanded code of the projects

Project 1

Specifications:

Input:

```json
{
  "image": {
    "type": "url",
    "hidden": false,
    "required": true
  }
}
```

Output:

```json
{
  "result": {
    "type": "string",
    "hidden": false,
    "required": true
  }
}
```

HTML:

```html
{{img src=image width="100%" height="400px"}}
<div>Are there <b>shoes</b> in the picture?</div>
<div>{{field type="radio" name="result" value="Yes" label="Yes" hotkey="1"}} {{field type="radio" name="result" value="No" label="No" hotkey="2"}}</div>
```

JS:

```javascript
exports.Task = extend(TolokaHandlebarsTask, function (options) {
  TolokaHandlebarsTask.call(this, options);
}, {
  onRender: function() {
    // DOM element for task is formed (available via #getDOMElement())
  },
  onDestroy: function() {
    // Task is completed. Global resources can be released (if used)
  }
});

function extend(ParentClass, constructorFunction, prototypeHash) {
  constructorFunction = constructorFunction || function () {};
  prototypeHash = prototypeHash || {};
  if (ParentClass) {
    constructorFunction.prototype = Object.create(ParentClass.prototype);
  }
  for (var i in prototypeHash) {
```

45
```
constructorFunction.prototype[i] = prototypeHash[i];
}
return constructorFunction;

Project 2
Specifications:

Input:
{
    "image": {
        "type": "url",
        "hidden": false,
        "required": true
    }
}

Output:
{
    "button": {
        "type": "boolean",
        "hidden": false,
        "required": true,
        "allowed_values": [true]
    },
    "found_link": {
        "type": "string",
        "hidden": false,
        "pattern": "https://www.asos.com/.*",
        "required": true
    },
    "found_image": {
        "type": "file",
        "hidden": false,
        "required": true
    }
}

HTML:
{{img src=image width="50%" height="400px"}}
<div class='answers'>
    <p>Find the same <b>shoes</b> on ASOS</p> {{field type="button-clicked" name="button" label="ASOS" href="https://www.asos.com" action=true}}
    <p>Shoes must be the same color and the same style.</p>
    <p>Paste the link here</p> {{field width="100%" type="input" name="found_link"}}
    <p>Upload the image here</p>
    <div>{{field width="100%" type="file-img" name="found_image" preview=true}}</div>
</div>

JS:
exports.Task = extend(TolokaHandlebarsTask, function(options) {
    TolokaHandlebarsTask.call(this, options);
}, {
    validate: function(solution) {
        if (!solution.output_values.found_image) {
            return {
                task_id: this.getTask().id,
                errors: {
                    '__TASK__': {
                        message: 'Upload photo or mark that there is no photo'
                    }
                }
            };
        } else {
            return TolokaHandlebarsTask.prototype.validate.apply(this, arguments);
        }
    },
    onRender: function() {
        // DOM element for task is formed (available via #getDOMElement())
    },
    onDestroy: function() {
        // Task is completed. Global resources can be released (if used)
    }
});

function extend(ParentClass, constructorFunction, prototypeHash) {
    constructorFunction = constructorFunction || function() {};
    prototypeHash = prototypeHash || {};
    if (ParentClass) {
        constructorFunction.prototype = Object.create(ParentClass.prototype);
    }
    for (var i in prototypeHash) {
        constructorFunction.prototype[i] = prototypeHash[i];
    }
    return constructorFunction;
}

CSS:

.task {
    display: block;
    height: 500px;
    width: 800px;
}

.img {
    float: left;
    width: 50%;
}

.answers {
    float: left;
    width: 40%;
    margin: 5%;
}
Project 3

Specifications:

Input:

```json
{
    "image": {
        "type": "url",
        "hidden": false,
        "required": true
    },
    "found_link": {
        "type": "url",
        "hidden": false,
        "required": true
    },
    "assignment_id": {
        "type": "string",
        "hidden": true,
        "required": true
    }
}
```

Output:

```json
{
    "result": {
        "type": "string",
        "hidden": false,
        "required": true
    }
}
```

HTML:

```html
{{img src=image height="400px"}} {{iframe src= found_link height="600px"}}
<p>Check that the uploaded image matches the product in the store.</p>
<p>Are these <b>shoes</b> similar to each other?</p>
<p>Shoes must be the same color and the same style.</p>
{{field type="radio" name="result" value="Yes" label="Yes"}}
{{field type="radio" name="result" value="No" label="No"}}
```

JS:

```javascript
exports.Task = extend(TolokaHandlebarsTask, function (options) {
    TolokaHandlebarsTask.call(this, options);
}, {
    onRender: function() {
        // DOM element for task is formed (available via #getDOMElement())
    },
    onDestroy: function() {
        // Task is completed. Global resources can be released (if used)
    }
});
```

```javascript
function extend(ParentClass, constructorFunction, prototypeHash) {
    constructorFunction = constructorFunction || function () {};
    prototypeHash = prototypeHash || {};
    if (ParentClass) {
        constructorFunction.prototype = Object.create(ParentClass.prototype);
    }
    for (var i in prototypeHash) {
        constructorFunction.prototype[i] = prototypeHash[i];
    }
```

48
return constructorFunction;

CSS:

.task {
  display: block;
}
.img {
  float: left;
  width: 50%;
}
.iframe {
  float: left;
  width: 50%;
}

Project 4:

Specifications:

Input:

```json
{
  "image": {
    "type": "url",
    "hidden": false,
    "required": true
  },
  "left_link": {
    "type": "url",
    "hidden": false,
    "required": true
  },
  "right_link": {
    "type": "url",
    "hidden": false,
    "required": true
  }
}
```

Output:

```json
{
  "result": {
    "type": "url",
    "hidden": false,
    "required": true
  }
}
```

HTML:

```
<div class="header">
  <div class="left caption">
    {{button label="Go to site" href=left_link size="L"}}</p>
    <p class="url">{{left_link}}</p>
  </div>
  <div class="right caption">
    <p class="url">{{right_link}}</p>
    {{button label="Go to site" href=right_link size="L"}}</p>
  </div>
</div>{{img src=image}}
<div class="content clearfix">
```
<div class="left page">
  {{iframe src=uploaded_link_left width="100%" height="700px" real-size=true screenshot=true}}
</div>

<div class="right page">
  {{iframe src=uploaded_link_right width="100%" height="700px" real-size=true screenshot=true}}
</div>

<div class="footer">
  {{field type="radio" name="result" label="The left one is better" value=result_left hotkey="1"}}
  {{field type="radio" name="result" label="The right one is better" value=result_right hotkey="2"}}
</div>

**JS:**

```javascript
exports.Task = extend(TolokaHandlebarsTask, function(options) {
  TolokaHandlebarsTask.call(this, options);
}, {
  getTemplateData: function() {
    var data = TolokaHandlebarsTask.prototype.getTemplateData.apply(this, arguments),
      input = this.getTask().input_values;
    var left_link = input.left_link;
    var right_link = input.right_link;
    var uploaded_link_left = '',
    uploaded_link_right = '';
    if (Math.floor(Math.random() * 2)) {
      uploaded_link_left = left_link
      uploaded_link_right = right_link
    } else {
      uploaded_link_left = right_link
      uploaded_link_right = left_link
    }
    data.uploaded_link_left = uploaded_link_left;
    data.uploaded_link_right = uploaded_link_right;
    data.result_left = uploaded_link_left;
    data.result_right = uploaded_link_right;
    return data;
  },
  onRender: function() {
    // DOM element for task is formed (available via #getDOMElement())
  },
  onDestroy: function() {
    // Task is completed. Global resources can be released (if used)
  }
});
```

```javascript
function extend(ParentClass, constructorFunction, prototypeHash) {
  constructorFunction = constructorFunction || function() {};
  prototypeHash = prototypeHash || {};
  if (ParentClass) {
    constructorFunction.prototype = Object.create(ParentClass.prototype);
  }
}
```
for (var i in prototypeHash) {
    constructorFunction.prototype[i] = prototypeHash[i];
}
return constructorFunction;

CSS:

.task {
    display: block;
    text-align: center;
}

.header {
    overflow: hidden;
    background-color: #FFCC00;
}

.caption {
    width: 50%;
}

.url {
    white-space: nowrap;
    overflow: hidden;
    text-overflow: ellipsis;
    max-width: calc(100% - 182px);
    display: inline-block;
    vertical-align: bottom;
}

.button {
    margin: 10px;
    max-width: 182px;
}

.content {
    margin: 10px 0;
}

.page {
    display: inline-block;
    width: 50%;
}

.left {
    float: left;
    text-align: left;
}

.right {
    float: right;
    text-align: right;
}

.clearfix {
overflow: hidden;
width: 100%;
}

.image {
  display: inline-block;
  width: 50%;
}