Team members: Nikita Sergeev BS20-AI n.sergeev@innopolis.university Github link

Current progress:

1) Data collection from github:

At the first stage of the project I collected commits data from github. For this purpose I clonned **100 most staarred python repositories**. Code to get list of repos via API:

```
for page in range(1, 11):
results = requests.get(
f'https://api.github.com/search/repositories?
q=language:python&sort=stars&order=desc&per_page=100&page={page}').json()
for repo in results['items']:
d_tmp = {'repository_ID': repo['id'],
'name': repo['name'],
'URL': repo['html_url'],
'created_date': datetime.strptime(repo['created_at'], '%Y-%m-%dT%H:%M:%SZ'),
'commits_api_link': repo['commits_url'].split('{')[0],
'number_of_stars': repo['stargazers_count']}
df = pd.concat([df, pd.DataFrame(d_tmp, index=[0])], ignore_index=True)
```

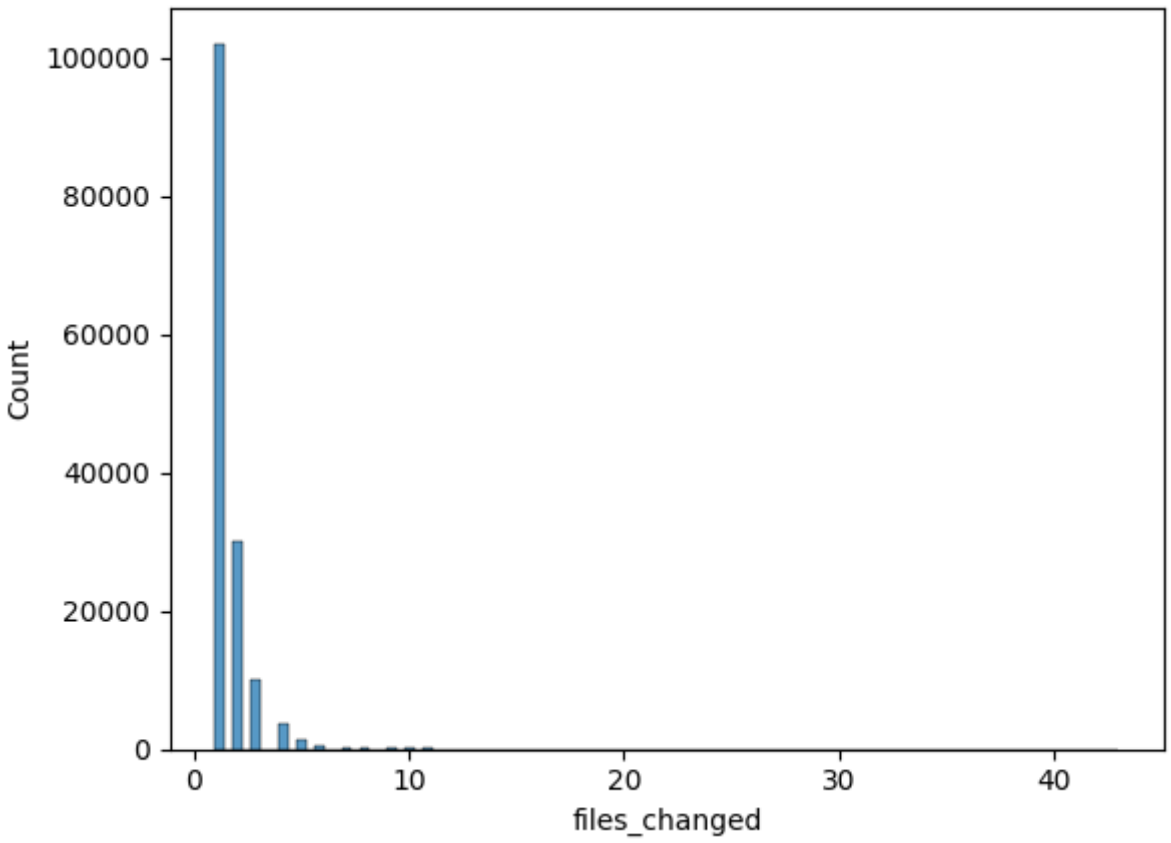
After this using `python git` package I clonned all of this projects, to store them locally. And the last step of getting commits data it's parse diff tree with the `git` lib to get all needed info about commits in the repo. Usefull features in my case are:

- repo name
- original commit message
- hash (unique identifier of the commit to remove dublicates in the future)
- commit changes
- number of files in the commit, which were changed
- len of the changes (in symbols)

2) Dataset cleaning and data preprocessing:

For the data preprocessing of did the following:

1. Take only commits with 1 file changed. As it's ok for the most of commits from the dataset

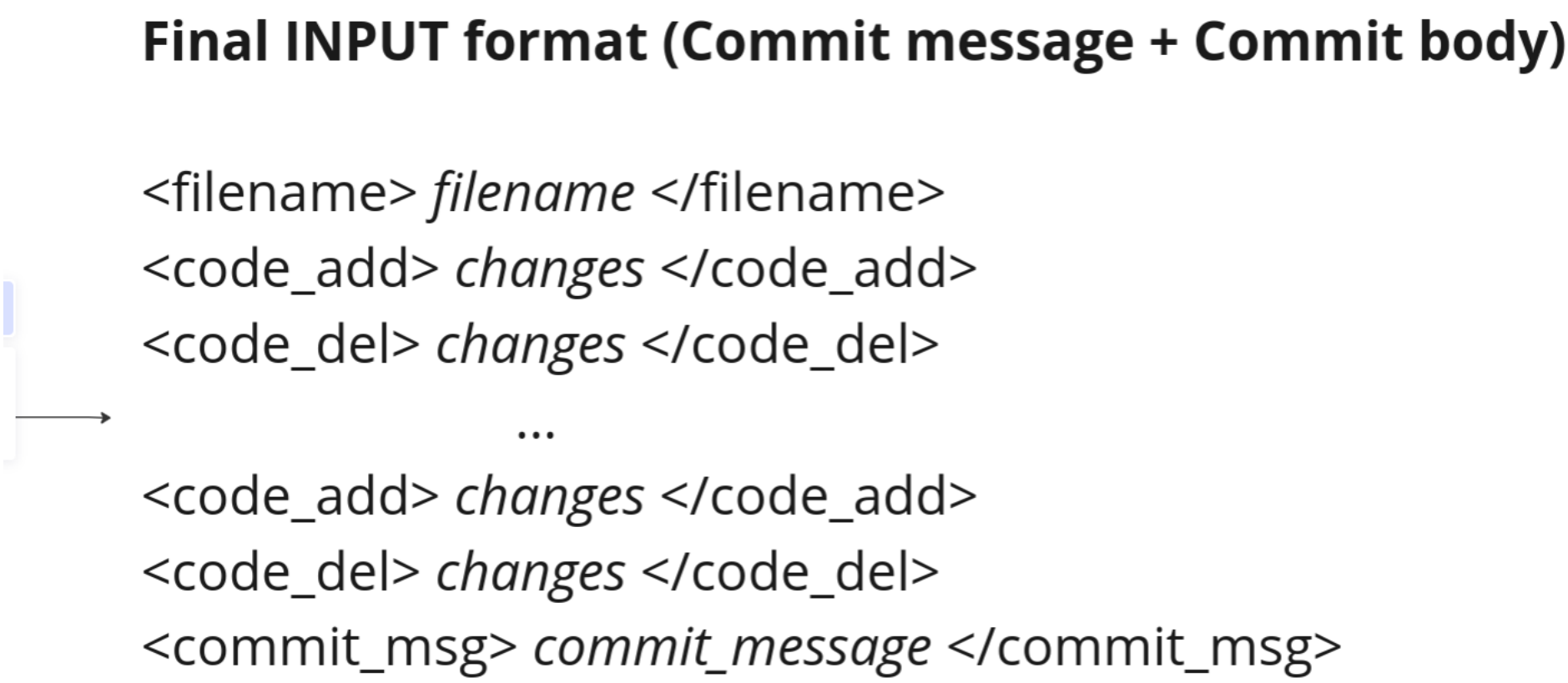


2. Remove samples what are to far from the average (to big code diffs, to big commit msg, non-ascii chars)
3. Take only the samples starting from the verb(fix, update, etc.)
4. Add special tokens to commit message and code diffs for future model training Final format of the commit diffs:

```
<filename> a/tests/test_constraint.py b/tests/test_constraint.py<filename>
<code_del>--- a/tests/test_constraint.py<code_del>
<code_add>+++ b/tests/test_constraint.py<code_add>
@@ -87,10 +87,15 @@ def test_accurate_approximation_when_known():
         n_iter=10,
     )

<code_del>-    params = optimizer.res[0]["params"]<code_del>
<code_del>-    x, y = params['x'], params['y']<code_del>
<code_add>+    # Exclude the last sampled point, because the constraint is not
fitted on that.<code_add>
<code_add>+    res = np.array([[r['target'], r['constraint'], r['params']['x'],
r['params']['y']] for r in optimizer.res[:-1]])<code_add>
<code_add>+<code_add>
<code_add>+    xy = res[:, [2, 3]]<code_add>
<code_add>+    x = res[:, 2]<code_add>
<code_add>+    y = res[:, 3]<code_add>

<code_del>-    assert constraint_function(x, y) ==
approx(conmod.approx(np.array([x, y])), rel=1e-5, abs=1e-5)<code_del>
<code_add>+    assert constraint_function(x, y) == approx(conmod.approx(xy),
rel=1e-5, abs=1e-5)<code_add>
<code_add>+    assert constraint_function(x, y) ==
approx(optimizer.space.constraint_values[:-1], rel=1e-5, abs=1e-5)<code_add>
```

```
 def test_multiple_constraints():

<commit_msg>
```

Commits message at the end need to help decoder model, wich predicts the next word to faster understand what it's should de with this special token.



Detailed description of the data preprocessing and filtering avilable by the following link https://miro.com/app/board/uXjVMwvUK5I=/

Dataset is available in:

- Kaggle - https://www.kaggle.com/datasets/naryqqq/commits-dataset
- HuggingFace - https://huggingface.co/datasets/narySt/github_commits (tokenized wersion of the dataset)

Future work plans:

Were was performed some works on commit message generation using BERT model - https://arxiv.org/abs/2105.14242 Possible next steps for this project:

- Reproduce the experiment from the CommitBERT paper
- Try some encoder-decoder model, pretrained for code (codeT5 for example)
- Try some decoder only LLM pretrained on code (CodeLLaMa or WizardCoder)
- Research about some other methods for commit messages generation using Deep Learning.
- Elaborate on metrics which will best suits for this tasks.