Пояснения к процедуре валидации датасета

В соответствии с критериями приемки нужно подготовить два набора данных по исходному коду: проверочный набор данных и обучающий набор данных.

В проверочный набор данных должно входить не менее 100 сниппетов кода, среди которых 50% сниппетов должны содержать дефект и 50% сниппетов кода должны не содержать никаких дефектов, где под сниппетом понимается код функции или метода. Корректность разметки на дефектный/недефектный должна быть 95%.

Проверочный набор данных должен содержать сниппеты, позволяющие оценить обучающую способность модели на коде из ранее не встречаемых проектов. Проверочный набор данных должен содержать примеры, позволяющие оценить обучающую способность модели на коде проектов из разных доменных областей.

В обучающий набор данных должно входить не менее 1000 сниппетов кода без ограничений на сбалансированность классов. Единственным требованием является корректная на 95% разметка фрагментов кода.

Планируется, что корректные сниппеты кода без дефектов будут взяты среди сниппетов, содержащих стабильный код. Сниппеты кода с дефектами предлагается отобрать вручную среди сниппетов, которые являются bug-сниппетами пар bugfix, ассоциированных с обработкой инцидентов с меткой “bug”, содержащих полный traceback ошибки: датасет PyBugs с данными по последним содержит довольно много полезной информации: url инцидента, traceback ошибки, url пулл-реквеста или sha коммита. Таким образом, большая часть информации о каждом вручную отобранном сниппете уже имеется в собранном датасете PyBugs.

В дополнение к этой информации предлагается также собрать следующую информацию о вручную отобранных сниппетах с ошибками:

1. индекс сниппета с ошибкой в таблице
2. описание ошибки на естественном (английском) языке
3. локализация ошибки
4. тип ошибки

Под индексом понимается индекс в таблице фрейма pandas, который содержится в разосланных pickle-файлах. Локализация должна содержать имя файла, метода/функции и диапазоны номеров строк в исходном тексте сниппета. При классификации ошибок следует использовать классификацию, описанную по ссылке <http://cwe.mitre.org/data/definitions/1000.html>, которая будет описана ниже.

Для чего нужно собирать всю вышеперечисленную информацию? Во-первых, чтобы иметь возможность асессору самостоятельно проверить собранную им информацию по сниппетам с ошибками. Во-вторых, эта информация может облегчить проверку подготовленного валидированного датасета в Хуавей.

Описание сниппетов с дефектами

Для сниппетов с дефектами предполагается собирать следующую информацию.

1. индекс сниппета с ошибкой в таблице
2. описание ошибки на естественном (английском) языке
3. локализация ошибки
4. тип ошибки

Описание ошибки следует делать со ссылками на названия конкретных переменных и функций. Локализация ошибки предполагает, что, вдобавок к уже имеющейся информации о пути к файлу и названию функции/метода нужно сообщить диапазоны строк в исправлявшемся сниппете, которые по вашему мнению содержат ошибку. Чем подробнее будет описана ошибка и дана ее локализация, тем проще будет проверить правильность собранной информации о сниппете.

При работе с очередным сниппетом с ошибкой следует иметь в виду, что нас не интересуют следующие типы ошибок:

1. ошибки несовместимости (dependency compatibility) с другими библиотеками: например, новая версия используемой проектом библиотеки теперь не поддерживает какой-то атрибут, скажем, он либо переименован, либо удален и т.п. ошибки в нашем проекте, возникающие в результате этого, нас не интересуют
2. ошибки обратной совместимости библиотеки (backward compatibility)
3. сложные ошибки, которые трудно описать и отыскивать (т.е. требуются специальные знания о функциональности проекта)

Каждому из вас переслан набор из 1000 сниппетов, которые правились в рамках обработки инцидентов с меткой ошибка. Нужно открыть этот набор данных в виде таблицы. В колонке commit\_message иногда содержится краткое описание сделанных исправлений в коде. По этому описанию можно понять подходит ли нам ошибка, исправленная в сниппете кода (см выше про ошибки, которые нам не нужны).

Прежде чем приступить к просмотру сниппетов, можно удалить из рассмотрения сниппеты, которые отличаются только комментариями или докстрингами. В таблице есть графы before\_merge\_without\_docstrings и after\_merge\_without\_docstrings. Можно удалить из таблицы все строки, в которых графы before\_merge\_without\_docstrings и after\_merge\_without\_docstrings совпадают. Можно также пытаться удалить все сниппеты, отличающиеся форматированием или отступами, например, применив токенизацию (например, с помощью функции tokenize.tokenize) с тем, чтобы удалить лишние пробельные символы, а затем сделать tokenize.untokenize с тем, чтобы вернуться обратно к исходному коду. Либо, как вариант, для удаления пар сниппетов, отличающихся форматированием, можно пробовать применять библиотеку difflib.

Далее нужно заглянуть на страницу инцидента, к которому относится данное исправление (графа url), и попробовать понять суть ошибки из текста описания ошибки и сообщения traceback. Наконец, если суть ошибки не очень ясна, походить по страницам пулл-реквестов (графа pr\_url) и коммитов (доступны со страницы инцидентов), которые тоже могут содержать информацию по ошибке и тому как она исправлялась. Наконец, нужно посмотреть, что исправлялось в сниппете (графы таблицы before\_merge и after\_merge). Далее, исходя из этой информации нужно понять в чем состоит ошибка (графа bug description), понятно описать ее на английском языке, сообщить ее локализацию (графа bug lines) в виде диапазона строк, а также присвоить ей определенный тип (графа bug type) согласно классификации по ссылке <http://cwe.mitre.org/data/definitions/1000.html>. Предлагается максимально подробно (в меру понимания) описать ошибку, сосредотачиваясь больше на причине ошибки, а не на формальном описании, содержащемся в full traceback.

Достаточно часто возникает ситуация, когда данному типу ошибки сопутствует несколько других типов ошибок. Допустим имела место ошибка деления на 0 и фактическое исправление ошибки было произведено в куске кода, где производится вычисление знаменателя (его текст содержится в графе before\_merge). Данная ошибка также отнесена к другой категории ошибок, состоящей в необработке (неправильной обработке) ситуации, когда знаменатель 0 в месте, где происходит фактическое деление. Этот тип ошибки, очевидно, локализован в том месте, где происходит деление, которое, вообще говоря, может находиться в другом сниппете, исходного кода которого нет в таблице, ведь наши данные содержат только фактические исправления и не содержат версий сниппетов кода, которые фактически не менялись.

В связи с этим предлагается добавить в таблицу несколько дополнительных граф: related bug description, related bug filename, related bug function\_name, related bug lines и related bug snippet, которые будут содержать информацию о связанной ошибке. Тут может возникнуть определенная проблема, состоящая в том, что исходный текст сниппета возможно придется искать вручную.

По моему мнению, в качестве основного или связанного типа ошибки стоит выбирать тот тип, который можно конструктивно отыскать с помощью статического анализа кода. Существуют типы ошибок, которые рассматривают как дефект отсутствие обработки граничных значений, а также крайних (ошибочных) ситуаций в коде. Отсутствие такой обработки в принципе может быть обнаружено алгоритмически на основе статического анализа кода. Мы раньше называли такой тип синтетической ошибкой (см предыдущую версию данной инструкции). Однако, как оказалось, существует уже известная достаточно абстрактная классификация дефектов, включающая по крайней мере часть категорий, которые по смыслу близки к тому, что мы называли синтетической ошибкой.

Классификация ошибок

По вышеупомянутой ссылке содержится достаточно известная (<https://en.wikipedia.org/wiki/Common_Weakness_Enumeration>) кроссязычная классификация ошибок. К сожалению, во многом она ориентирована на C и C++. Эта классификация ошибок построена в виде иерархии классов ошибок.

Приведем некоторые примеры. Например, категория Incorrect Calculation содержит в себе классы арифметических ошибок типа деления на 0 или численного переполнения, когда результат вычислений не влезает в разрядную сетку и т.п. Кроме того, категория Improper Control of a Resource Through its Lifetime содержит в себе в качестве подкатегории Incorrect Access of Indexable Resource. Эта подкатегория содержит в себе разные ошибки класса IndexError. Например, категория Buffer Over-read содержит ошибки типа индекс в массиве выходит за максимальную границу диапазона индексов. Также категория Incomplete Internal State Distinction может соответствовать ошибкам, когда происходит работа с уже закрытым файлом.

Также существуют категории ошибок, связанные с некорректным преобразованием типа Incorrect Type Conversion or Cast.

В классификации также существуют категории того, что очень близко к тому, что мы считали синтетической ошибкой. Так, например, в классификации имеется категория Improper Check or Handling of Exceptional Conditions, в которой есть подкатегория Improper Handling of Missing Special Element, которая описывает как дефект отсутствие корректной обработки ситуации, когда у объекта, переданного в функцию, отсутствует определенный атрибут, который используется в этой функции.

При работе с категориями следует обращать внимание на следующую информацию:

1. Description и Extended Description содержат краткое и подробное описание ошибки
2. Раздел Relevant to the view "Research Concepts" содержит родственные категории ошибок (CanAlsoBe, CanFollow), которые возможно также описывают рассматриваемую вами ошибку
3. Applicable Platforms содержит информацию о языках программирования, в которых наиболее часто встречается данная ошибка; тут нужно думать взять ли данную категорию в качестве типа рассматриваемой ошибки или ее родительскую категорию
4. Demonstrative Examples содержит примеры кода на соответствующем языке программирования, содержащего данную ошибку
5. Detection Methods содержит описание методов поиска данной категории ошибок
6. Notes содержит разные важные пояснения и замечания

Таким образом, на основе всей этой информации предлагается присвоить данному сниппету с ошибкой одну или несколько категорий ошибок. На основе информации из раздела Detection Methods, а также на основе собственных знаний, можно пробовать описать способ отыскания такой ошибки, дающий малое количество false positives.

Случай нескольких релевантных категорий ошибок

Ошибки в коде чаще всего связаны с его недопроверенностью, а также с отсутствием обработки различных крайних значений и исключительных ситуаций. Это означает, что ошибки часто сочетаются с другими типами дефектов.

Рассмотрим серию из нескольких примеров.

ПРИМЕР 1. Ошибка деления на 0

У нас есть две функции:

Первая

def rescale(image, scale, order=None, mode='reflect', cval=0, clip=True,

preserve\_range=False, multichannel=False,

anti\_aliasing=None, anti\_aliasing\_sigma=None):

scale = np.atleast\_1d(scale)

if len(scale) > 1:

if ((not multichannel and len(scale) != image.ndim) or

(multichannel and len(scale) != image.ndim - 1)):

raise ValueError("Supply a single scale, or one value per spatial "

"axis")

if multichannel:

scale = np.concatenate((scale, [1]))

orig\_shape = np.asarray(image.shape)

output\_shape = np.round(scale \* orig\_shape)

if multichannel: # don't scale channel dimension

output\_shape[-1] = orig\_shape[-1]

return resize(image, output\_shape, order=order, mode=mode, cval=cval,

clip=clip, preserve\_range=preserve\_range,

anti\_aliasing=anti\_aliasing,

anti\_aliasing\_sigma=anti\_aliasing\_sigma)

Вторая

def resize(image, output\_shape, order=None, mode='reflect', cval=0, clip=True,

preserve\_range=False, anti\_aliasing=None, anti\_aliasing\_sigma=None):

output\_shape = tuple(output\_shape)

output\_ndim = len(output\_shape)

input\_shape = image.shape

if output\_ndim > image.ndim:

# append dimensions to input\_shape

input\_shape = input\_shape + (1, ) \* (output\_ndim - image.ndim)

image = np.reshape(image, input\_shape)

elif output\_ndim == image.ndim - 1:

# multichannel case: append shape of last axis

output\_shape = output\_shape + (image.shape[-1], )

elif output\_ndim < image.ndim - 1:

raise ValueError("len(output\_shape) cannot be smaller than the image "

"dimensions")

if anti\_aliasing is None:

anti\_aliasing = not image.dtype == bool

if image.dtype == bool and anti\_aliasing:

warn("Input image dtype is bool. Gaussian convolution is not defined "

"with bool data type. Please set anti\_aliasing to False or "

"explicitely cast input image to another data type. Starting "

"from version 0.19 a ValueError will be raised instead of this "

"warning.", FutureWarning, stacklevel=2)

factors = (np.asarray(input\_shape, dtype=float) /

np.asarray(output\_shape, dtype=float))

if anti\_aliasing:

if anti\_aliasing\_sigma is None:

anti\_aliasing\_sigma = np.maximum(0, (factors - 1) / 2)

else:

anti\_aliasing\_sigma = \

np.atleast\_1d(anti\_aliasing\_sigma) \* np.ones\_like(factors)

if np.any(anti\_aliasing\_sigma < 0):

raise ValueError("Anti-aliasing standard deviation must be "

"greater than or equal to zero")

elif np.any((anti\_aliasing\_sigma > 0) & (factors <= 1)):

warn("Anti-aliasing standard deviation greater than zero but "

"not down-sampling along all axes")

# Translate modes used by np.pad to those used by ndi.gaussian\_filter

np\_pad\_to\_ndimage = {

'constant': 'constant',

'edge': 'nearest',

'symmetric': 'reflect',

'reflect': 'mirror',

'wrap': 'wrap'

}

try:

ndi\_mode = np\_pad\_to\_ndimage[mode]

except KeyError:

raise ValueError("Unknown mode, or cannot translate mode. The "

"mode should be one of 'constant', 'edge', "

"'symmetric', 'reflect', or 'wrap'. See the "

"documentation of numpy.pad for more info.")

image = ndi.gaussian\_filter(image, anti\_aliasing\_sigma,

cval=cval, mode=ndi\_mode)

# 2-dimensional interpolation

if len(output\_shape) == 2 or (len(output\_shape) == 3 and

output\_shape[2] == input\_shape[2]):

rows = output\_shape[0]

cols = output\_shape[1]

input\_rows = input\_shape[0]

input\_cols = input\_shape[1]

if rows == 1 and cols == 1:

tform = AffineTransform(translation=(input\_cols / 2.0 - 0.5,

input\_rows / 2.0 - 0.5))

else:

# 3 control points necessary to estimate exact AffineTransform

src\_corners = np.array([[1, 1], [1, rows], [cols, rows]]) - 1

dst\_corners = np.zeros(src\_corners.shape, dtype=np.double)

# take into account that 0th pixel is at position (0.5, 0.5)

dst\_corners[:, 0] = factors[1] \* (src\_corners[:, 0] + 0.5) - 0.5

dst\_corners[:, 1] = factors[0] \* (src\_corners[:, 1] + 0.5) - 0.5

tform = AffineTransform()

tform.estimate(src\_corners, dst\_corners)

# Make sure the transform is exactly metric, to ensure fast warping.

tform.params[2] = (0, 0, 1)

tform.params[0, 1] = 0

tform.params[1, 0] = 0

out = warp(image, tform, output\_shape=output\_shape, order=order,

mode=mode, cval=cval, clip=clip,

preserve\_range=preserve\_range)

else: # n-dimensional interpolation

order = \_validate\_interpolation\_order(image.dtype, order)

coord\_arrays = [factors[i] \* (np.arange(d) + 0.5) - 0.5

for i, d in enumerate(output\_shape)]

coord\_map = np.array(np.meshgrid(\*coord\_arrays,

sparse=False,

indexing='ij'))

image = convert\_to\_float(image, preserve\_range)

ndi\_mode = \_to\_ndimage\_mode(mode)

out = ndi.map\_coordinates(image, coord\_map, order=order,

mode=ndi\_mode, cval=cval)

\_clip\_warp\_output(image, out, order, mode, cval, clip)

return out

Во второй функции resize возникает ошибка деления на 0, если в первой функции параметр scale=0.1, а orig\_shape=(3,10). А именно, после округления в первой функции (показано красным) получаем output\_shape = (0,1). Далее в rescale вызывается функция resize с параметром output\_shape, который в функции resize возникает в знаменателе (показано красным). Подытожим, что мы имеем:

1. выражение для factors в функции resize является отношением двух операндов, один из которых вычисляется в другой функции
2. нет никакой проверки на деление на 0 в этом выражении

Эта ошибка исправляется путем определения немного другой формулы для output\_shape в функции rescale. А именно output\_shape = np.maximum(np.round(scale \* orig\_shape), 1). Такое исправление приводит к тому, что output\_shape не содержит нулевых координат, и, как следствие, ошибки деления на 0 не возникает. Таким образом, реальная ошибка деления на 0 в данном примере состоит в том, что параметр output\_shape вычисляется неправильно, и она локализована в месте, где вычисляется знаменатель. При этом знаменатель и само выражение для отношения вычисляются в разных функциях.

Однако, на это можно посмотреть с другой стороны. В месте, где происходит фактическое деление, т.е. в выражении для factors, нет обработки ситуации на наличие нулевых координат в массиве output\_shape. Данная ситуация также считается дефектом. Такой дефект с необработкой нулевого знаменателя часто сопровождает ошибку деления на 0. Поскольку правильность выражения для output\_shape проверить довольно трудно, приходится прибегать к тому, чтобы проверять код, где фактически происходит деление, на предмет наличия проверки знаменателя или обработки соответствующей исключительной ситуации.

Таким образом, несмотря на то, что ошибка содержится в функции rescale, соответствующий дефект содержится в функции resize или в перегруженной функции \_\_truediv\_\_ , где нет соответствующей проверки на деление на 0. Сам факт наличия такой проверки говорит о том, что разработчик знает об этой возможной крайней ситуации и умеет ее обрабатывать.

Безусловно, в коде полно таких ситуаций, где знаменатель не равен 0 ни при каких условиях. Наличие подобного рода проверок везде и всюду затруднило бы читабельность кода. В связи с этим, возможно, что стоит считать дефектом только те ситуации, где операнд, фигурирующий в знаменателе, вычисляется полностью в другой функции.

Приведем также пример кода, содержащего ошибку, но не содержащего дефекта отсутствия проверок.

ПРИМЕР 2. Ошибка IndexError

Если массив, в котором индексируются элементы, и собственно, его индекс не согласованы, возникают ошибки типа IndexError. Имеются две функции:

def \_build\_laplacian(data, spacing, mask, beta, multichannel):

l\_x, l\_y, l\_z = data.shape[:3]

edges = \_make\_graph\_edges\_3d(l\_x, l\_y, l\_z)

weights = \_compute\_weights\_3d(data, spacing, beta=beta, eps=1.e-10,

multichannel=multichannel)

if mask is not None:

# Remove edges of the graph connected to masked nodes, as well

# as corresponding weights of the edges.

mask0 = np.hstack([mask[..., :-1].ravel(), mask[:, :-1].ravel(),

mask[:-1].ravel()])

mask1 = np.hstack([mask[..., 1:].ravel(), mask[:, 1:].ravel(),

mask[1:].ravel()])

ind\_mask = np.logical\_and(mask0, mask1)

edges, weights = edges[:, ind\_mask], weights[ind\_mask]

# Reassign edges labels to 0, 1, ... edges\_number - 1

\_, inv\_idx = np.unique(edges, return\_inverse=True)

edges = inv\_idx.reshape(edges.shape)

# Build the sparse linear system

pixel\_nb = edges.shape[1]

i\_indices = edges.ravel()

j\_indices = edges[::-1].ravel()

data = np.hstack((weights, weights))

lap = sparse.coo\_matrix((data, (i\_indices, j\_indices)),

shape=(pixel\_nb, pixel\_nb))

lap.setdiag(-np.ravel(lap.sum(axis=0)))

return lap.tocsr()

и функция:

def \_build\_linear\_system(data, spacing, labels, nlabels, mask,

beta, multichannel):

if mask is None:

labels = labels.ravel()

else:

labels = labels[mask]

indices = np.arange(labels.size)

seeds\_mask = labels > 0

unlabeled\_indices = indices[~seeds\_mask]

seeds\_indices = indices[seeds\_mask]

lap\_sparse = \_build\_laplacian(data, spacing, mask=mask,

beta=beta, multichannel=multichannel)

rows = lap\_sparse[unlabeled\_indices, :]

lap\_sparse = rows[:, unlabeled\_indices]

B = -rows[:, seeds\_indices]

seeds = labels[seeds\_mask]

seeds\_mask = sparse.csc\_matrix(np.hstack(

[np.atleast\_2d(seeds == lab).T for lab in range(1, nlabels + 1)]))

rhs = B.dot(seeds\_mask)

return lap\_sparse, rhs

Переменная pixel\_nb (помечено красным) определяет размер матрицы лапласиана, выдаваемой на выходе функции \_build\_laplacian. Далее в функции \_build\_linear\_system в строке, помеченной красным мы производим операцию выборки из массива lap\_sparse по индексу unlabeled\_indices, где из-за несоответствия (при определенных условиях) размеров индекса unlabeled\_indices и массива lap\_sparse получается ошибка IndexError, которая затем обрабатывается в \_\_getitem\_\_ массивов scipy.

Таким образом, ошибка состоит в неправильном вычислении значения pixel\_nb в функции \_build\_laplacian, что приводит к тому, что индекс unlabeled\_indices содержит значения, большие длины массива, при этом код не содержит дефектов необработки крайних ситуаций.

ПРИМЕР 3. Рассмотрим пример ошибки, связанной с обработкой файлов. У нас есть две функции:

def send(self, content=None, \*, wait=False, username=None, avatar\_url=None, tts=False,

file=None, files=None, embed=None, embeds=None):

"""|maybecoro|

Sends a message using the webhook.

If the webhook is constructed with a :class:`RequestsWebhookAdapter` then this is

not a coroutine.

The content must be a type that can convert to a string through ``str(content)``.

To upload a single file, the ``file`` parameter should be used with a

single :class:`File` object.

If the ``embed`` parameter is provided, it must be of type :class:`Embed` and

it must be a rich embed type. You cannot mix the ``embed`` parameter with the

``embeds`` parameter, which must be a :class:`list` of :class:`Embed` objects to send.

Parameters

------------

content

The content of the message to send.

wait: bool

Whether the server should wait before sending a response. This essentially

means that the return type of this function changes from ``None`` to

a :class:`Message` if set to ``True``.

username: str

The username to send with this message. If no username is provided

then the default username for the webhook is used.

avatar\_url: str

The avatar URL to send with this message. If no avatar URL is provided

then the default avatar for the webhook is used.

tts: bool

Indicates if the message should be sent using text-to-speech.

file: :class:`File`

The file to upload. This cannot be mixed with ``files`` parameter.

files: List[:class:`File`]

A list of files to send with the content. This cannot be mixed with the

``file`` parameter.

embed: :class:`Embed`

The rich embed for the content to send. This cannot be mixed with

``embeds`` parameter.

embeds: List[:class:`Embed`]

A list of embeds to send with the content. Maximum of 10. This cannot

be mixed with the ``embed`` parameter.

Raises

--------

HTTPException

Sending the message failed.

NotFound

This webhook was not found.

Forbidden

The authorization token for the webhook is incorrect.

InvalidArgument

You specified both ``embed`` and ``embeds`` or the length of

``embeds`` was invalid.

Returns

---------

Optional[:class:`Message`]

The message that was sent.

"""

payload = {}

if files is not None and file is not None:

raise InvalidArgument('Cannot mix file and files keyword arguments.')

if embeds is not None and embed is not None:

raise InvalidArgument('Cannot mix embed and embeds keyword arguments.')

if embeds is not None:

if len(embeds) > 10:

raise InvalidArgument('embeds has a maximum of 10 elements.')

payload['embeds'] = [e.to\_dict() for e in embeds]

if embed is not None:

payload['embeds'] = [embed.to\_dict()]

if content is not None:

payload['content'] = str(content)

payload['tts'] = tts

if avatar\_url:

payload['avatar\_url'] = avatar\_url

if username:

payload['username'] = username

if file is not None:

try:

to\_pass = (file.filename, file.open\_file(), 'application/octet-stream')

return self.\_adapter.execute\_webhook(wait=wait, file=to\_pass, payload=payload)

finally:

file.close()

elif files is not None:

try:

to\_pass = [(file.filename, file.open\_file(), 'application/octet-stream')

for file in files]

return self.\_adapter.execute\_webhook(wait=wait, files=to\_pass, payload=payload)

finally:

for file in files:

file.close()

else:

return self.\_adapter.execute\_webhook(wait=wait, payload=payload)

и вторая

def execute\_webhook(self, \*, payload, wait=False, file=None, files=None):

if file is not None:

multipart = {

'file': file,

'payload\_json': utils.to\_json(payload)

}

data = None

elif files is not None:

multipart = {

'payload\_json': utils.to\_json(payload)

}

for i, file in enumerate(files, start=1):

multipart['file%i' % i] = file

data = None

else:

data = payload

multipart = None

url = '%s?wait=%d' % (self.\_request\_url, wait)

maybe\_coro = self.request('POST', url, multipart=multipart, payload=data)

return self.handle\_execution\_response(maybe\_coro, wait=wait)

При асинхронной отправке файлов с локального диска с помощью веб-хуков (специальных функций, которые активируют определенные HTTP-запросы по заданному адресу в ответ на какое-то событие на веб-странице, скажем, обрабатывая событие отправки файла)

await webhook.send(file=discord.File("file.txt"))

выдается ошибка о том, что файл закрыт:

ValueError: I/O operation on closed file

В секции try (помечена красным) происходит собственно отправка файла и ожидается ответ от сервера, при этом в разделе finally перед этим происходит закрытие отправляемого файла, что приводит к ошибке.

Таким образом, ошибка состоит в том, что не синхронизированы завершение отправки файла и закрытие этого файла в рамках асинхронного запуска (Incorrect Behavior Order). Дефект, сопровождающий данную ошибку, состоит в том, что обработка события отправки файла и закрытие этого файла фактически происходят в разных функциях и не обрабатывается ситуация с уже закрытой файловой переменной file ((Incomplete Internal State Distinction).

(см. <https://github.com/Rapptz/discord.py/commit/311788c7cbac19ee62c6e38a1214ec32259d9d29>)

Подытожим. Мы различаем ошибки и дефекты. Реальные ошибки могут быть самые разные. Мы пытаемся сами придумать для каждой реальной ошибки ее синтетический аналог в виде дефекта, который можно обнаружить статически, т.е. не запуская код на выполнение. Многие типы дефектов и ошибок содержатся в классификации, описанной выше.