cw-eval Documentation

Release 1.0.0

David Lindenbaum and Nick Weir

Mar 12, 2019

Contents

Author CosmiQ Works

Release 1.0.0

CHAPTER 1

CosmiQ Works Evaluation API reference

1.1 Core functionality

class cw_eval.baseeval.**EvalBase**(*ground_truth_vector_file*)

Object to test IoU for predictions and ground truth polygons.

Parameters **ground_truth_vector_file** (*str*) – Path to .geojson file for ground truth.

eval_iou(*miniou=0.5, iou_field_prefix='iou_score', ground_truth_class_field='', calculate_class_scores=True, class_list=['all']*)

Evaluate IoU between the ground truth and proposals.

Parameters

- **miniou** (*float, optional*) – Minimum intersection over union score to qualify as a successful object detection event. Defaults to `0.5`.
- **iou_field_prefix** (*str, optional*) – The name of the IoU score column in `self.proposal_GDF`. Defaults to `"iou_score"`.
- **ground_truth_class_field** (*str, optional*) – The column in `self.ground_truth_GDF` that indicates the class of each polygon. Required if using `calculate_class_scores`.
- **calculate_class_scores** (*bool, optional*) – Should class-by-class scores be calculated? Defaults to `True`.
- **class_list** (*list, optional*) – List of classes to be scored. Defaults to `['all']` (score all classes).

Returns

scoring_dict_list – list of score output dicts for each image in the ground truth and evaluated image datasets. The dicts contain the following keys:

```
('class_id', 'iou_field', 'TruePos', 'FalsePos', 'FalseNeg',
'Precision', 'Recall', 'F1Score')
```

Return type list

eval_iou_spacenet_csv (*miniou=0.5*, *iou_field_prefix='iou_score'*, *imageIDField='ImageId'*, *debug=False*, *minArea=0*)

Evaluate IoU between the ground truth and proposals in CSVs.

Parameters

- **miniou** (*float , optional*) – Minimum intersection over union score to qualify as a successful object detection event. Defaults to `0.5`.
- **iou_field_prefix** (*str , optional*) – The name of the IoU score column in `self.proposal_GDF`. Defaults to `"iou_score"` .
- **imageIDField** (*str , optional*) – The name of the column corresponding to the image IDs in the ground truth data. Defaults to `"ImageId"`.
- **debug** (*bool , optional*) – Argument for verbose execution during debugging. Defaults to `False` (silent execution).
- **minArea** (*float or int , optional*) – Minimum area of a ground truth polygon to be considered during evaluation. Often set to `20` in SpaceNet competitions. Defaults to `0` (consider all ground truth polygons).

Returns

scoring_dict_list – list of score output dicts for each image in the ground truth and evaluated image datasets. The dicts contain the following keys:

```
('imageID', 'iou_field', 'TruePos', 'FalsePos', 'FalseNeg',
'Precision', 'Recall', 'F1Score')
```

Return type list

load_proposal (*proposal_vector_file, conf_field_list=['conf'], proposalCSV=False, pred_row_geo_value='PolygonWKT_Pix', conf_field_mapping=[]*)

Load in a proposal geojson or CSV.

Parameters

- **proposal_vector_file** (*str*) – Path to the file containing proposal vector objects. This can be a .geojson or a .csv.
- **conf_field_list** (*list, optional*) – List of columns corresponding to confidence value(s) in the proposal vector file. Defaults to `['conf']`.
- **proposalCSV** (*bool, optional*) – Is the proposal file a CSV? Defaults to no (`False`), in which case it's assumed to be a .geojson.
- **pred_row_geo_value** (*str, optional*) – The name of the geometry-containing column in the proposal vector file. Defaults to `'PolygonWKT_Pix'`. Note: this method assumes the geometry is in WKT format.
- **conf_field_mapping** (*dict, optional*) – `'__max_conf_class'` column value:class ID mapping dict for multiclass use. Only required in multiclass cases.

Returns

Return type `0` upon successful completion.

Notes

Loads in a .geojson or .csv-formatted file of proposal polygons for comparison to the ground truth and stores it as part of the `EvalBase` instance. This method assumes the geometry contained in the proposal file is in WKT format.

load_truth(*ground_truth_vector_file*, *truthCSV=False*, *truth_geo_value='PolygonWKT_Pix'*)
Load in the ground truth geometry data.

Parameters

- **ground_truth_vector_file** (*str*) – Path to the ground truth vector file. Must be either .geojson or .csv format.
- **truthCSV** (*bool, optional*) – Is the ground truth a CSV? Defaults to `False`, in which case it's assumed to be a .geojson.
- **truth_geo_value** (*str, optional*) – Column of the ground truth vector file that corresponds to geometry.

Returns

Return type Nothing.

Notes

Loads the ground truth vector data into the `EvalBase` instance.

cw_eval.baseeval.**eval_base**(*ground_truth_vector_file*, *csvFile=False*, *truth_geo_value='PolygonWKT_Pix'*)
Deprecated API to EvalBase.

Deprecated since version 0.3: Use *EvalBase* instead.

cw_eval.evalfunctions.**calculate_iou**(*pred_poly*, *test_data_GDF*)
Get the best intersection over union for a predicted polygon.

Parameters

- **pred_poly** (`shapely.Polygon`) – Prediction polygon to test.
- **test_data_GDF** (`geopandas.GeoDataFrame`) – GeoDataFrame of ground truth polygons to test `pred_poly` against.

Returns **iou_GDF** – A subset of `test_data_GDF` that overlaps `pred_poly` with an added column `iou_score` which indicates the intersection over union value.

Return type `geopandas.GeoDataFrame`

cw_eval.evalfunctions.**process_iou**(*pred_poly*, *test_data_GDF*, *remove_matching_element=True*)
Get the maximum intersection over union score for a predicted polygon.

Parameters

- **pred_poly** (`shapely.geometry.Polygon`) – Prediction polygon to test.
- **test_data_GDF** (`geopandas.GeoDataFrame`) – GeoDataFrame of ground truth polygons to test `pred_poly` against.
- **remove_matching_element** (*bool, optional*) – Should the maximum IoU row be dropped from `test_data_GDF`? Defaults to `True`.

Returns

Return type *This function doesn't currently return anything.*

1.2 SpaceNet Challenge eval code

cw_eval.challenge_eval.off_nadir_dataset.**eval_off_nadir**(*prop_csv*, *truth_csv*, *imageColumns={}*, *miniou=0.5*, *minArea=20*)

Evaluate an off-nadir competition proposal csv.

Uses `EvalBase` to evaluate off-nadir challenge proposals. See `imageColumns` in the source code for how collects are broken into Nadir, Off-Nadir, and Very-Off-Nadir bins.

Parameters

- **prop_csv** (*str*) – Path to the proposal polygon CSV file.
- **truth_csv** (*str*) – Path to the ground truth polygon CSV file.
- **imageColumns** (*dict, optional*) – dict of `(collect: nadir bin)` pairs used to separate collects into sets. Nadir bin values must be one of `["Nadir", "Off-Nadir", "Very-Off-Nadir"]`. See source code for collect name options.
- **miniou** (*float, optional*) – Minimum IoU score between a region proposal and ground truth to define as a successful identification. Defaults to 0.5.
- **minArea** (*float or int, optional*) – Minimum area of ground truth regions to include in scoring calculation. Defaults to `20`.
- **Returnss** –
- **-------** –
- **results_DF_Full** (*results_DF,*) –

  results_DF [`pd.DataFrame`] Summary `pd.DataFrame` of score outputs grouped by nadir angle bin, along with the overall score.

  results_DF_Full [`pd.DataFrame`] `pd.DataFrame` of scores by individual image chip across the ground truth and proposal datasets.

cw_eval.challenge_eval.off_nadir_dataset.**get_collect_id**(*imageID*)

Get the collect ID for an image name using a regex.

CHAPTER 2

CosmiQ Works Evaluation Cookbook

2.1 Evaluate a proposal for one chip against ground truth

This recipe describes how to run evaluation of a proposal (CSV or .geojson) for a single chip against a ground truth (CSV or .geojson) for the same chip.

2.1.1 CSV Eval

Steps

1. Imports
2. Load ground truth CSV
3. Load proposal CSV
4. Perform evaluation

Imports

For this test case we will only need `cw_eval` installed - Installation instructions for cw_eval

[1]:
```
# imports
import os
import cw_eval
from cw_eval.baseeval import EvalBase  # class used for evaluation
from cw_eval.data import data_dir  # get the path to the sample eval data
import pandas as pd  # just for visualizing the outputs in this recipe
```

Load ground truth CSV

We will first instantiate an `EvalBase()` object, which is the core class `cw_eval` uses for comparing predicted labls to ground truth labels. `EvalBase()` takes one argument - the path to the CSV or .geojson ground truth label object. It can alternatively accept a pre-loaded `GeoDataFrame` of ground truth label geometries.

[2]:
```
ground_truth_path = os.path.join(data_dir, 'sample_truth.csv')

eval_object = EvalBase(ground_truth_path)
eval_object
```

[2]:
```
EvalBase sample_truth.csv
```

At this point, `eval_object` has the following attributes:

- `ground_truth_fname`: the filename corresponding to the ground truth data. This is simply `'GeoDataFrame'` if a GDF was passed during instantiation.
- `ground_truth_GDF`: GeoDataFrame-formatted geometries for the ground truth polygon labels.
- `ground_truth_GDF_Edit`: A deep copy of `eval_object.ground_truth_GDF` which is edited during the process of matching ground truth label polygons to proposals.
- `ground_truth_sindex`: The RTree/libspatialindex spatial index for rapid spatial referencing.
- `proposal_GDF`: An *empty* GeoDataFrame instantiated to hold proposals later.

Load proposal CSV

Next we will load in the proposal CSV file. Note that the `proposalCSV` flag must be set to true for CSV data. If the CSV contains confidence column(s) that indicate confidence in proprosals, the name(s) of the column(s) should be passed as a list of strings with the `conf_field_list` argument; because no such column exists in this case, we will simply pass `conf_field_list=[]`. There are additional arguments available (see the method documentation) which can be used for multi-class problems; those will be covered in another recipe. The defaults suffice for single-class problems.

[3]:
```
proposals_path = os.path.join(data_dir, 'sample_preds.csv')
eval_object.load_proposal(proposals_path, proposalCSV=True, conf_field_list=[])
```

Perform evaluation

Evaluation iteratively steps through the proposal polygons in `eval_object.proposal_GDF` and determines if any of the polygons in `eval_object.ground_truth_GDF_Edit` have IoU overlap > `miniou` (see the method documentation) with that proposed polygon. If one does, that proposal polygon is scored as a true positive. The matched ground truth polygon with the highest IoU (in case multiple had IoU > `miniou`) is removed from `eval_object.ground_truth_GDF_Edit` so it cannot be matched against another proposal. If no ground truth polygon matches with IoU > `miniou`, that proposal polygon is scored as a false positive. After iterating through all proposal polygons, any remaining ground truth polygons in `eval_object.ground_truth_GDF_Edit` are scored as false negatives.

There are several additional arguments to this method related to multi-class evaluation which will be covered in a later recipe. See the method documentation for usage.

The prediction outputs a `list` of `dicts` for each class evaluated (only one `dict` in this single-class case). The `dict`(s) have the following keys:

- `'class_id'`: The class being scored in the dict, `'all'` for single-class scoring.
- `'iou_field'`: The name of the column in `eval_object.proposal_GDF` for the IoU score for this class. See the method documentation for more information.
- `'TruePos'`: The number of polygons in `eval_object.proposal_GDF` that matched a polygon in `eval_object.ground_truth_GDF_Edit`.
- `'FalsePos'`: The number of polygons in `eval_object.proposal_GDF` that had no match in `eval_object.ground_truth_GDF_Edit`.
- `'FalseNeg'`: The number of polygons in `eval_object.ground_truth_GDF_Edit` that had no match in `eval_object.proposal_GDF`.
- `'Precision'`: The precision statistic for IoU between the proposals and the ground truth polygons.
- `'Recall'`: The recall statistic for IoU between the proposals and the ground truth polygons.
- `'F1Score'`: Also known as the SpaceNet Metric, the F1 score for IoU between the proposals and the ground truth polygons.

[4]:
```
eval_object.eval_iou(calculate_class_scores=False)
```

```
151it [00:01, 110.15it/s]
```

[4]:
```
[{'class_id': 'all',
  'iou_field': 'iou_score_all',
  'TruePos': 151,
  'FalsePos': 0,
  'FalseNeg': 0,
  'Precision': 1.0,
  'Recall': 1.0,
  'F1Score': 1.0}]
```

In this case, the score is perfect because the polygons in the ground truth CSV and the proposal CSV are identical. At this point, a new proposal CSV can be loaded (for example, for a new nadir angle at the same chip location) and scoring can be repeated.

2.1.2 GeoJSON Eval

The same operation can be completed with .geojson-formatted ground truth and proposal files. See the example below, and see the detailed explanation above for a description of each step's operations.

[5]:
```
ground_truth_geojson = os.path.join(data_dir, 'gt.geojson')
proposal_geojson = os.path.join(data_dir, 'pred.geojson')

eval_object = EvalBase(ground_truth_geojson)
eval_object.load_proposal(proposal_geojson, proposalCSV=False, conf_field_list=[])
eval_object.eval_iou(calculate_class_scores=False)
```

```
28it [00:00, 85.50it/s]
```

[5]:
```
[{'class_id': 'all',
  'iou_field': 'iou_score_all',
  'TruePos': 8,
  'FalsePos': 20,
  'FalseNeg': 20,
  'Precision': 0.2857142857142857,
```

(continues on next page)

(continued from previous page)

```
  'Recall': 0.2857142857142857,
  'F1Score': 0.2857142857142857}]
```

(Note that the above comes from a different chip location and different proposal than the CSV example, hence the difference in scores)

2.2 Score an entire competition (or a whole AOI) using `cw-eval`

This recipe describes how to run evaluation of a proposal CSV for an entire competition against a ground truth CSV.

2.2.1 Things to understand before starting

When we score entire competitions, we want to ensure that competitors provide submissions for the entire area of interest (AOI), not just the subset that competitors provide scores for, in case they leave out chips that they can't predict well. Therefore, proposal files scored using this pipeline should contain predictions for every chip in the ground truth CSV. The score outputs also provide chip-by-chip results which can be used to remove non-predicted chips if needed.

When CosmiQ Works runs competitions in partnership with TopCoder, we set some cutoffs for scoring buildings:

- An IoU score of > 0.5 is required to ID a building as correctly identified.
- Ground truth buildings fewer than 20 pixels in extent are ignored. However, it is up to competitors to filter out their own small footprint predictions.

2.2.2 Imports

For this test case we will only need `cw_eval` installed - Installation instructions for cw_eval

[1]:
```
# imports
import os
import cw_eval
from cw_eval.challenge_eval.off_nadir_dataset import eval_off_nadir  # runs eval
from cw_eval.data import data_dir  # get the path to the sample eval data
import pandas as pd  # just for visualizing the outputs in this recipe
```

2.2.3 Ground truth CSV format

The following shows a sample ground truth CSV and the elements it must contain.

[2]:
```
ground_truth_path = os.path.join(data_dir, 'sample_truth_competition.csv')

pd.read_csv(ground_truth_path).head(10)
```

[2]:
```
                                             ImageId  BuildingId  \
0  Atlanta_nadir8_catid_10300100023BC100_743501_3...           0
1  Atlanta_nadir8_catid_10300100023BC100_743501_3...           1
2  Atlanta_nadir8_catid_10300100023BC100_743501_3...           2
3  Atlanta_nadir8_catid_10300100023BC100_743501_3...           3
4  Atlanta_nadir8_catid_10300100023BC100_743501_3...           4
5  Atlanta_nadir8_catid_10300100023BC100_743501_3...           5
6  Atlanta_nadir8_catid_10300100023BC100_743501_3...           6
7  Atlanta_nadir8_catid_10300100023BC100_743501_3...           7
8  Atlanta_nadir8_catid_10300100023BC100_743501_3...           8
9  Atlanta_nadir8_catid_10300100023BC100_743501_3...           9

                                       PolygonWKT_Pix  PolygonWKT_Geo
0  POLYGON ((476.88 884.61, 485.59 877.64, 490.50...               1
1  POLYGON ((459.45 858.97, 467.41 853.09, 463.37...               1
2  POLYGON ((407.34 754.17, 434.90 780.55, 420.27...               1
3  POLYGON ((311.00 760.22, 318.38 746.78, 341.02...               1
4  POLYGON ((490.49 742.67, 509.81 731.14, 534.12...               1
5  POLYGON ((319.28 723.07, 339.97 698.22, 354.29...               1
6  POLYGON ((466.49 709.69, 484.26 696.45, 502.59...               1
7  POLYGON ((433.84 673.34, 443.90 663.96, 448.70...               1
8  POLYGON ((459.24 649.03, 467.38 641.90, 472.84...               1
9  POLYGON ((403.55 643.50, 416.98 630.51, 440.36...               1
```

Important points about the CSV format:

- The column denoting the chip ID for a given geospatial location must be titled `ImageId`.
- The column containing geometries must be in WKT format and should be titled `PolygonWKT_Pix`.
- The `BuildingId` column provides a numeric identifier sequentially numbering each building *within each chip*. Order doesn't matter.
- **For chips with no buildings**, a single row should be provided with `BuildingID=-1` and `PolygonWKT_Pix="POLYGON EMPTY"`.

2.2.4 Proposal CSV format

[3]:
```
proposals_path = os.path.join(data_dir, 'sample_preds_competition.csv')
pd.read_csv(proposals_path).head(10)
```

[3]:
```
                                             ImageId  BuildingId  \
0  Atlanta_nadir8_catid_10300100023BC100_743501_3...           0
1  Atlanta_nadir8_catid_10300100023BC100_743501_3...           1
2  Atlanta_nadir8_catid_10300100023BC100_743501_3...           0
3  Atlanta_nadir8_catid_10300100023BC100_743501_3...           1
4  Atlanta_nadir8_catid_10300100023BC100_743501_3...           2
5  Atlanta_nadir8_catid_10300100023BC100_743501_3...           0
6  Atlanta_nadir8_catid_10300100023BC100_743501_3...           1
7  Atlanta_nadir8_catid_10300100023BC100_743501_3...           2
8  Atlanta_nadir8_catid_10300100023BC100_743501_3...           3
9  Atlanta_nadir8_catid_10300100023BC100_743501_3...           4

                                       PolygonWKT_Pix  Confidence
0  POLYGON ((0.00 712.83, 158.37 710.28, 160.59 6...           1
1  POLYGON ((665.82 0.00, 676.56 1.50, 591.36 603...           1
2  POLYGON ((182.62 324.15, 194.25 323.52, 197.97...           1
```

(continues on next page)

(continued from previous page)

```
3  POLYGON ((92.99 96.94, 117.20 99.64, 114.72 12...           1
4  POLYGON ((0.82 29.96, 3.48 40.71, 2.80 51.00, ...           1
5  POLYGON ((476.88 884.61, 485.59 877.64, 490.50...           1
6  POLYGON ((459.45 858.97, 467.41 853.09, 463.37...           1
7  POLYGON ((407.34 754.17, 434.90 780.55, 420.27...           1
8  POLYGON ((311.00 760.22, 318.38 746.78, 341.02...           1
9  POLYGON ((490.49 742.67, 509.81 731.14, 534.12...           1
```

The only difference between the ground truth CSV format and the prediction CSV format is the `Confidence` column, which can be used to provide prediction confidence for a polygon. Alternatively, it can be set to 1 for all polygons to indicate equal confidence.

2.2.5 Running eval on the Off-Nadir challenge: Python API

`cw-eval` currently contains code for scoring proposals from the Off-Nadir Building Detection challenge. There are two ways to run scoring: using the Python API or using the CLI (see later in this recipe). The below provides an example using the Python API.

If you provide proposals and ground truth formatted as described earlier, no additional arguments are required unless you would like to alter the default scoring settings. If so, see the API docs linked above.

The scoring function provides two outputs:

- `results_DF`, a summary Pandas DataFrame with scores for the entire AOI split into the nadir/off-nadir/very off-nadir bins
- `results_DF_Full`, a DataFrame with chip-by-chip score outputs for detailed analysis. For large AOIs this function takes a fair amount of time to run.

[4]:
```
results_DF, results_DF_Full = eval_off_nadir(proposals_path, ground_truth_path)
```

```
100%|| 33/33 [00:14<00:00,  2.11it/s]
```

[5]:
```
results_DF
```

[5]:
```
                F1Score  FalseNeg  FalsePos  Precision  Recall  TruePos
nadir-category
Nadir               1.0         0         0        1.0     1.0     2319
```

(This ground truth dataset only contained nadir imagery, hence the absence of the other bins)

[6]:
```
results_DF_Full.head(10)
```

[6]:
```
   F1Score  FalseNeg  FalsePos  Precision  Recall  TruePos  \
0      1.0         0         0        1.0     1.0       96
1      1.0         0         0        1.0     1.0        3
2      1.0         0         0        1.0     1.0       43
3      1.0         0         0        1.0     1.0       67
4      1.0         0         0        1.0     1.0        3
5      1.0         0         0        1.0     1.0       91
6      1.0         0         0        1.0     1.0       80
7      1.0         0         0        1.0     1.0       96
8      1.0         0         0        1.0     1.0      112
9      1.0         0         0        1.0     1.0       78
```

(continues on next page)

(continued from previous page)

```
                                                imageID  iou_field nadir-category
0  Atlanta_nadir8_catid_10300100023BC100_743501_3...  iou_score           Nadir
1  Atlanta_nadir8_catid_10300100023BC100_743501_3...  iou_score           Nadir
2  Atlanta_nadir8_catid_10300100023BC100_743501_3...  iou_score           Nadir
3  Atlanta_nadir8_catid_10300100023BC100_743501_3...  iou_score           Nadir
4  Atlanta_nadir8_catid_10300100023BC100_743501_3...  iou_score           Nadir
5  Atlanta_nadir8_catid_10300100023BC100_743501_3...  iou_score           Nadir
6  Atlanta_nadir8_catid_10300100023BC100_743501_3...  iou_score           Nadir
7  Atlanta_nadir8_catid_10300100023BC100_743501_3...  iou_score           Nadir
8  Atlanta_nadir8_catid_10300100023BC100_743501_3...  iou_score           Nadir
9  Atlanta_nadir8_catid_10300100023BC100_743501_3...  iou_score           Nadir
```

2.3 Running eval on the Off-Nadir Challenge using the CLI

The `cw-eval` CLI allows competition scoring without even needing to open a Python shell. Its usage is as follows:

```
$ spacenet_eval --proposal_csv [proposal_csv_path] --truth_csv [truth_csv_path] --
↪output_file [output_csv_path]
```

Argument details:

- `--proposal_csv`, `-p`: Path to the proposal CSV. Required argument. See the API usage details above for CSV specifications.
- `--truth_csv`, `-t`: Path to the ground truth CSV. Required argument. See the API usage details above for CSV specifications.
- `--output_file`, `-o`: Path to save the output CSVs to. This script will produce two CSV outputs: `[output_file].csv`, which is the summary DataFrame described above, and `[output_file]_full.csv`, which contains the chip-by-chip scoring results.

Not implemented yet: The CLI also provides a `--challenge` command, which is not yet implemented, but will be available in future versions to enable scoring of other SpaceNet challenges.

Example:

[7]:
```
%%bash -s "$proposals_path" "$ground_truth_path" # ignore this line - magic line to␣
↪run bash shell command
spacenet_eval --proposal_csv $1 --truth_csv $2 --output_file results  # argument␣
↪values taken from magic line above
```

```
                F1Score  FalseNeg  FalsePos  Precision  Recall  TruePos
nadir-category
Nadir               1.0         0         0        1.0     1.0     2319
Writing summary results to result.csv
Writing full results to result_full.csv
```

```
100%|| 33/33 [00:17<00:00,  1.16it/s]
```

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Python Module Index

C

Index