SkyTruth FrackFinder PA 2005-2013 Methodology

Author: Kevin Wurster - kevin@skytruth.org

A description of the methodology and workflow required to identify fracking related ponds in Pennsylvania from 2005-2013 with crowd-sourced analysis of aerial imagery.

Requirements

- QGIS >= 2.2
- Python 2.7
- GDAL/OGR 1.11.0 + Python bindings

Secondary Requirements

- skyconvert.py
- pybossa_tools
- mangleresults.py

The `pybossa_tools` can be installed by cloning the repository and running:

> ```
> $ cd pybossa_tools
> $ pip install .
> ```

Instructions on how to install `skyconvert.py` can be found in its repository.

Instructions on how to install `mangleresults.py` can be found in its repository.

Nomenclature

Each phase in the FrackFinder PA chain (except for one) is named after a frog and requires the crowd to perform a simple task.

1. Tadpole: Determine when and where well pads were active
2. MoorFrog: Identify all standing bodies of water and empty ponds within roughly 500 m of a well pad
3. DartFrog: Classify standing bodies of water or empty ponds as fracking related
4. Digitizer: Draw polygons around ponds identified to be fracking related

---

Source Data

General Description

1. Locate source data
2. Scrape and process source data

1. Locate and Scrape Source Data

The search for fracking related ponds can be significantly optimized by narrowing down the search area. A fair assumption is that these ponds are located relatively near the wells they serve, so the first step to narrowing the search area is identifying permitted wells.

For any given well, permitting and SPUD information was scraped from the Pennsylvania Department of Environmental Protection (PADEP). Additional information about when the well was actually fracked was scraped from FracFocus. These two sources are necessary to identify wells that were permitted but never drilled, which were later filtered out to reduce the number of tasks given to the crowd. The source code for the scrapers can be found in the SkyTruth scrapers repository.

Orthorectified aerial imagery from the National Agriculture Imagery Program (NAIP) was obtained for the following years: 2005, 2008, 2010, and 2013. Unfortunately the original uncompressed data was not obtainable so data was received in the standard MrSID format.

2. Process Source Data

Small groups of permit locations are often located in very close proximity since multiple wells can be drilled from a single pad. The permit locations, dates, and available imagery can be examined to identify when a well pad was active, but if every permit was examined for every year there would be a significant amount of overlap. For example, 7 wells in close proximity don't need to examined individually to determine when the actual pad was present because the pad remains as long as any well is being drilled. Instead the crowd could look at a single point representing these 7 points and still produce the same information.

Well permits were sorted into date order and clustered into groups using a nearest-neighbor algorithm where points that are within 100 m of an existing cluster are added to that cluster. A centroid and GUID were generated for each cluster in order to later reference the data. The GUID traveled with each task in the task.json['info']['siteID'] field and can be found in the databases in the `guuid` field. As additional permits were scraped they were added to existing sites but the GUID and clustered lat/long did not change.

The sites were extracted from a project called SiteInfo which creates "sites" by spatially clustering drilling permits, SPUD reports, FracFocus reports and whatever other spatial data is available. This data lives here:

```
host: ewn3.skytruth.org
db: skytruth
tables: public.appomatic.siteinfo.*
```

In order to connect a SiteID to individual well API's, the following query must be used:

```
SELECT *
from public.appomatic_siteinfo_well w
JOIN public.appomatic_siteinfo_basemodel b
ON w.site_id = b.id
```

This data has been exported and can be found in the `Source_Data` directory:

```
2005-2010/Source_Data/siteinfo.csv
```

The NAIP was uploaded to Google Maps Engine and processed with the default settings into a final map that could be accessed via an OGC compliant WMS client.

FrackFinder PA 2005-2010

Tadpole 2005-2010 Workflow

General Description

1. Generate tasks and load tasks
2. Classify
3. Export data
4. Transform data

5. Sample Data/QAQC

1. Generate and Load Tasks

Tasks were generated by finding clustered sites where at least one of the associated permits was for an unconventional well within the years 2005-2010. A sample task is shown below. The info key contains information generated by SkyTruth - everything else is generated by PyBossa. The TadPole application used static tiles extracted from the NAIP imagery so the 'url' key is a link to the scene used by this task.

```
{'app_id': 652,
 'calibration': 0,
 'created': '2013-07-23T19:12:17.678700',
 'id': 284621,
 'info': {'county': 'Bradford',
          'latitude': '41.5894',
          'longitude': '-76.3351',
          'siteID': 'b487cb9b-7da9-5685-b8ee-c6d10e548386',
          'url':
'https://s3.amazonaws.com/FrackFinder/Tadpole/Bradford/2010/2010_X-
076.3351_Y0041.5894.png',
          'year': '2010'},
 'n_answers': 10,
 'priority_0': 0.0,
 'quorum': 0,
 'state': 'completed'}
```

The input tasks for Tadpole can be found here:

```
2005-2010/Transformations_and_QAQC/Tadpole/input_tasks/Tadpole-pa_2005-
2010_Tasks.csv
```

Instead of using PyBossa, this application used CrowdCrafting.org, which accepts the above CSV as input.

2. Classify

Users were shown a scene and were asked to determine which category it fell into. The scenes were centered around a clustered set of permits so the user was instructed to use context clues to identify what was happening near the center of the scene.

- Pad: The scene contains a pad at its center
- Pad with Equipment: The scene contains a pad with visible equipment

- No Pad: The scene does not contain a pad at its center
- Unknown: The user could not discern any information from this scene

Each scene was examined by 10 different crowd volunteers.

3. Export Data

About 90% All tasks were completed by the public and the remaining 10% were completed by SkyTruth employees. The tasks and task runs were downloaded and stored in the following locations:

```
Public:
Transformations_and_QAQC/Tadpole/tasks/task.json
Transformations_and_QAQC/Tadpole/tasks/task_run.json

Internal:
TODO: Need file paths
```

A 'Task Run' represents a single user's response to a single task.

4. Transform Data

While performing the analysis related to assembling this document, the data was transformed according to the description below, but while sampling the data between applications the data was converted with `skyconvert.py` and loaded into QGIS where the sampling described in the first paragraph of `Sample Data/QAQC` took place.

The data needs to be converted to a spatial format in order to perform analyses and better understand how one feature relates to another. The `task2shp.py` utility performs all necessary conversions and task/task run aggregation. In general, the output is a an ESRI Shapefile with one feature per task and a set of attributes collected from the associated task runs. Transformations were performed with the following command:

“

```
$ ./Transformations_and_QAQC/Tadpole/bin/task2shp.py \
    Transformations_and_QAQC/Tadpole/tasks/task.json \
    Transformations_and_QAQC/Tadpole/tasks/task_run.json \
    Transformations_and_QAQC/Tadpole/transform/stats/tadpole-stats.shp
```

The resulting file contains the following fields:

```
id         ->  Task's ID as assigned by PyBossa
site_id    ->  SkyTruth assigned unique site ID
wms_url    ->  Link to image scene
county     ->  County name
year       ->  Imagery year
location   ->  Generated unique ID (lat + long + year)
n_unk_res  ->  Number of task runs where task was classified as 'unknown'
n_nop_res  ->  Number of task runs where task was classified as 'nopad'
n_eqp_res  ->  Number of task runs where task was classified as 'unknown'
n_emp_res  ->  Number of task runs where task was classified as 'empty'
n_tot_res  ->  Number of times this task was completed
crowd_sel  ->  Classification with the highest number of selections or
class1|class2|etc. for ties
qaqc       ->  Used in the manual QAQC process
p_crd_a    ->  Percent of the crowd's responses that matched the crowd_sel field
p_s_crd_a  ->  Percentages for crowd_sel ties percent1|percent2|etc. or NULL if
there was no tie
```

5. Sample Data/QAQC

The public started to slow down after they completed about 90% of all tasks. At this point the remaining tasks were exclusively completed by SkyTruth employees. Roughly 350 tasks were examined by an analyst who determined whether or not the crowd's dominant selection was correct. The data was split into five groups based on confidence level, which is the ratio of number of crowd responses matching the dominant answer to the total number of responses for that task. These groups are: 90% to 100%, 80% to 90%, 70% to 80%, 60% to 70%, 50% to 60% and < 50%. Each group of tasks was examined by a SkyTruth analyst to determine the quality of the data based on the analyst's evaluation of each individual task. The 70% to 80% bin was found to contain unreliable data based on the analyst disagreeing with the crowd's classification of roughly 25% of the tasks in this bin. All tasks with an agreement level >= 80% were passed on to the MoorFrog task generation phase and all tasks with an agreement level < 80% were re-examined internally by a SkyTruth employee. This yields a single dataset where each task has a classification that was confidently assigned by the crowd or a SkyTruth employee. A report with results from this analysis can be found in the documents for Tadpole:

```
2005-2010/Transformations_and_QAQC/Tadpole/documents/2013-07-01-14-01-03-
results_analysis.xlsx
```

An additional report titled `2013-08-09-10-08-25-results_analysis.xlsx` can also be found in the Tadpole documents but it was only used to answer some initial questions about the data and is only included for legacy purposes.

While assembling the final report you are now reading, an additional analysis was performed on the data. A random sample of 100 sites was taken from each year (2005, 2008, 2010) and manually classified by an analyst using QGIS and the same imagery displayed in the PyBossa application. There is a bug in QGIS's random sampling tool that ignores any filters applied to the input datasource so the filters must be applied and then the data must be exported to a new file, from which a random sample can be selected. Queries and exports were performed in QGIS. All queries were performed on Tadpole stats layer:

```
Transformations_and_QAQC/Tadpole/transform/stats/tadpole-stats.shp
    "year" = 2005  ->
Transformations_and_QAQC/Tadpole/sampling/queries/tadpole-query-2005.shp
    "year" = 2008  ->
Transformations_and_QAQC/Tadpole/sampling/queries/tadpole-query-2008.shp
    "year" = 2010  ->
Transformations_and_QAQC/Tadpole/sampling/queries/tadpole-query-2010.shp
```

The random samples were exported to the sampling directory and examined in place:

```
Transformations_and_QAQC/Tadpole/sampling/2005/tadpole-2005-sample-100.shp
Transformations_and_QAQC/Tadpole/sampling/2008/tadpole-2008-sample-100.shp
Transformations_and_QAQC/Tadpole/sampling/2010/tadpole-2010-sample-100.shp
```

2005 Results

2 disagreements were found, one of which was caused by the crowd preferring two classifications an equal number of times, and the other was caused by ambiguity in the imagery.

2008 Results

4 disagreements were found, which were all caused by ambiguity in the imagery.

2010 Results

6 disagreements were found, about half of which were caused by the crowd preferring two classifications an equal number of times. The other half were related to empty vs. equipment vs. nopad ambiguity caused by compressed NAIP used in the application.

MoorFrog 2005-2010 Workflow

General Description

1. Identify, generate, and load input tasks
2. Classify
3. Export data
4. Transform data
5. Sample Data

1. Identify, Generate, and Load Input Tasks

Input tasks for MoorFrog were generated from the output of Tadpole's `Sample Data/QAQC` step. The data was filtered in Excel to only include tasks classified as having a pad and a few additional attributes were added. Tadpole served individual tiled scenes but MoorFrog used WMS layers so a new WMS URL was assigned to each task. The imagery was managed at the county level, so for logistical reasons each county was exported to its own CSV, converted to JSON via `skyconvert.py` and loaded into a PyBossa application via `createTasks.py` with commands similar to these:

“

```
$ skyconvert.py {COUNTY}.csv {COUNTY}.json

$ createTasks.py -a {APP_NAME} \
    -c -v -n 10 \
    -s http://crowd.skytruth.org \
    -k {YOUR_API_KEY} \
    -t {COUNTY}.json \
```

NOTE: The `createTasks.py` command above is appropriate only for the first county loaded. All subsequent loads should replace the `-c` option with `-u` in order to update an existing application instead of creating a new one.

2. Classify

Users were shown a scene centered on a point that was confidently classified as a well pad with a red 1 km x 1 km box centered around the pad. The user was instructed to click once on the center of any body of water within the box before submitting their results. The resulting task runs contain all the clicks for any given user.

3. Export Data

All tasks were completed by the public in a single applications. The tasks and task runs were downloaded and stored in the following locations:

```
Transformations_and_QAQC/MoorFrog/tasks/task.json
Transformations_and_QAQC/MoorFrog/tasks/task_run.json
```

4. Transform Data

MoorFrog's `task2shp.py` utility produces all necessary derivative datasets and aggregations with a single command. The MoorFrog tasks have a well pad point, bounding box, and task runs containing the user's clicks. The output files include the following layers: bounding boxes, pond clicks, and well pad points. The following command produces all data:

> ```
> $ ./Transformations_and_QAQC/MoorFrog/bin/task2shp.py \
>     Transformations_and_QAQC/MoorFrog/tasks/task.json \
>     Transformations_and_QAQC/MoorFrog/tasks/task_run.json \
>     Transformations_and_QAQC/MoorFrog/transform/
> ```

Output fields: `Transformations_and_QAQC/MoorFrog/transform/MoorFrog-bbox.shp`

```
id        ->  Task's ID as assigned by PyBossa
site_id   ->  SkyTruth assigned unique site ID
location  ->  Generated primary key (lat + long + year)
wms_url   ->  Link to imagery required for this task
county    ->  County name
year      ->  Imagery year
qaqc      ->  Used in manual QAQC
```

Output fields: `Transformations_and_QAQC/MoorFrog/transform/MoorFrog-clicks.shp`

```
id       ->  Task's ID as assigned by PyBossa
task_id  ->  PyBossa assigned task_id from task_run.json (matches
task.json['task_id'])
year     ->  Imagery year
qaqc     ->  Used in manual QAQC
```

Output fields: `Transformations_and_QAQC/MoorFrog/transform/MoorFrog-wellpads.shp`

```
id        ->  Task's ID as assigned by PyBossa
site_id   ->  SkyTruth assigned unique site ID
location  ->  Generated primary key (lat + long + year)
wms_url   ->  Link to imagery required for this task
county    ->  County name
year      ->  Imagery year
qaqc      ->  Used in manual QAQC
```

5. Sample Data

The sampling explained below was performed after DartFrog was completed and is only intended to provide a better idea of the quality of the data.

When examining a MoorFrog task the user was asked to click on of all the ponds located within the bounding box. In order to determine how well the crowd performed a random sample of 100 bounding boxes per year (2005, 2008, 2010) were extracted and manually examined by an analyst who was looking for missed ponds. S

Some user's disregarded the directions and clicked on ponds outside of the bounding box. These clicks were thrown away before performing any analysis. Any pond (fracking or otherwise) within the bounding box that was not clicked on at least was once was considered omitted. The analyst created a new Shapefile containing one point per missed pond.

The QGIS bug explained and addressed in the `Tadpole Sample Data` section was accounted for when creating sample tasks:

```
Transformations_and_QAQC/MoorFrog/transform/MoorFrog-bbox.shp
    "year" = 2005  ->  Transformations_and_QAQC/MoorFrog/sampling/queries/bbox-
2005-query.shp
    "year" = 2008  ->  Transformations_and_QAQC/MoorFrog/sampling/queries/bbox-
2008-query.shp
    "year" = 2010  ->  Transformations_and_QAQC/MoorFrog/sampling/queries/bbox-
2010-query.shp
```

The random samples and missed ponds are located in the sampling directory and were examined in place:

```
Transformations_and_QAQC/MoorFrog/sampling/2005/moorfrog-2005-sample-100.shp
Transformations_and_QAQC/MoorFrog/sampling/2005/moorfrog-2005-missed-ponds.shp
Transformations_and_QAQC/MoorFrog/sampling/2008/moorfrog-2008-sample-100.shp
Transformations_and_QAQC/MoorFrog/sampling/2008/moorfrog-2008-missed-ponds.shp
Transformations_and_QAQC/MoorFrog/sampling/2010/moorfrog-2010-sample-100.shp
Transformations_and_QAQC/MoorFrog/sampling/2010/moorfrog-2010-missed-ponds.shp
```

2005 Results

2005 - 18 omitted ponds - None appear to be fracking related

2008 Results

2008 - 19 omitted ponds - None appear to be fracking related

2010 Results

2010 - 6 omitted ponds - 2 appear to be fracking related

DartFrog 2005-2010 Workflow

General Description

1. Identify, generate, and load input tasks
2. Generate and load tasks
3. Classify
4. Export data
5. Transform data
6. Sample Data

1. Identify, Generate, and Load Input Tasks

MoorFrog's is essentially a bunch of points representing where users clicked on ponds. These ponds are then classified in DartFrog to determine whether or not they are fracking related. In order to do this, the clicks must be clustered into a more sane format, otherwise the crowd would be examining each pond about 100 times instead of 10.

MoorFrog's task.json and task_run.json were combined with `mangleresults.py` which outputs a single GeoJSON file. The data is in WGS84 but clustering is best performed in a flat plane projection, so the file was loaded into QGIS where an analyst split the points into two different files, one with points falling in UTM 17N and one with points falling in UTM 18N. These files were exported in the proper UTM zone and split again into years, which yields several files:

```
points for 2005 in UTM 17N
points for 2005 in UTM 18N
points for 2008 in UTM 17N
points for 2008 in UTM 18N
points for 2010 in UTM 17N
points for 2010 in UTM 18N
```

Each file was then loaded into QGIS and processed according to the following steps:

1. Load file into QGIS
2. Apply a 20m buffer to the file using the buffer tool. Let the tool dissolve the results.
3. The buffered layer only contains a single dissolved geometry so explode it with the multipart to singleparts tool
4. Generate centroids from the singleparts file
5. Delete all attributes from the centroids layer
6. Join the exploded/singleparts buffer layer's attributes to the centroids by location - only take the nearest feature
7. Reproject the centroids to WGS84

This yields one file per year and UTM zone in WGS84.

```
centroids with attributes for 2005 in UTM 17N reprojected to WGS84
centroids with attributes for 2005 in UTM 18N reprojected to WGS84
centroids with attributes for 2008 in UTM 17N reprojected to WGS84
centroids with attributes for 2008 in UTM 18N reprojected to WGS84
centroids with attributes for 2010 in UTM 17N reprojected to WGS84
centroids with attributes for 2010 in UTM 18N reprojected to WGS84
```

Since all the files are now in WGS84, each year can be combined and exported to GeoJSON, which yields the following:

```
centroids with attributes for 2005 in WGS84
centroids with attributes for 2008 in WGS84
centroids with attributes for 2010 in WGS84
```

Each year's file was then converted to JSON with `skyconvert.py` and loaded into a PyBossa app with `createTasks`:

“

```
$ skyconvert.py {YEAR}.geojson {YEAR}.json

$ createTasks.py -a {APP_NAME} \
    -c -v -n 10 \
    -s http://crowd.skytruth.org \
    -k {YOUR_API_KEY} \
    -t {YEAR}.json \
```

NOTE: The `createTasks.py` command above is appropriate only for the first year loaded. All subsequent loads should replace the `-c` option with `-u` in order to update an existing application instead of creating a new one.

2. Classify

Users were shown a scene centered on a pond identified in MoorFrog and were asked to classify that pond based on the following criteria:

- Fracking
- Not-Fracking
- Other
- Unknown

At one point the crowd appeared to stall on the internal application so a set of about 4000 tasks were moved to an application that was only accessible by SkyTruth employees. It was later determined that these tasks were not properly moved so each task was completed at least once but may have been completed in multiple applications. There is known overlap between the public and first internal application, meaning that some tasks were completed (partially or fully) in both applications. In general it is best to interact with the data through the utilities provided. The `Compiled_Output.csv` listed below is the most reliable dataset and has the complete record of each task's history, including the final classification for each task and where it was completed.

```
2005-2010/Transformations_and_QAQC/DartFrog/transform/Compiled_Output.csv
```

While a redundancy of 10 was used for the public application but a redundancy of 3 was used for all internal applications. Since the tasks are being completed by SkyTruth's experienced image analysts a much lower redundancy can be used since the crowd is far more trustworthy. The lower redundancy also allows for the tasks to completed much quicker since they are being looked at fewer times.

A detailed explanation of each application can be found below.

Public Application

The files in `public` are directly exported from the application presented to the public. This application was never fully completed by the crowd, so when working with the tasks it is important to understand that tasks from task.json were only fully completed if there are >= 10 task runs in the task_run.json file.

First Internal Application

The files in `first-internal` were from the very first DartFrog internal application. The crowd stalled on the public application around the 50% mark so almost 4000 tasks were pulled out of the public app and moved to an internal application with a lower redundancy. PyBossa determines which task to show to a given user by looking at the redundancy. Tasks with a redundancy of 0 never need to be completed so PyBossa never gives them to a user. In order to essentially hide the 4000 tasks that were moved from the public application's users, their redundancy was to 0 by directly updating the tasks in PyBossa's database.

Final Internal Application

The files in `final-internal` are a combination of tasks from public and first-internal. An error was discovered through examining the tasks and task runs from the Public and First Internal applications that caused some tasks to never be completed. Any task that was never fully completed in these applications was loaded into a new "Final Internal" application.

Sweeper Internal Application

The `task2shp.py` utility was used to convert each task.json and its matching task_run.json file into a spatial format with a set of attributes explaining the crowd's selection and confidence level. If the crowd classified a pond as 'fracking' 8 times and 'other' 2 times, 80% of the crowd agreed. These attributes and agreement levels were used to identify which ponds needed to be re-examined one final time by SkyTruth employees. A total of 1280 ponds were placed into a sweeper application to resolve any ambiguity based on the following criteria:

For the public application, any pond with an agreement level < 80% and any pond that was confidently classified as 'unknown'

For the internal applications, any pond with an agreement level < 66%.

For all applications, any pond where the crowd's response was evenly split across two or more choices.

Missing Internal Application

After the four applications detailed above were completed, the `dartfrog-taskCompiler.py` utility was developed to stitch together a complete history for any given pond and to identify a final pond classification. Through developing this utility it was discovered that 221 tasks had never been completed in any application. These tasks were identified and completed in a final application.

3. Export Data

Each DartFrog application's tasks are stored separately in the following locations:

```
Transformations_and_QAQC/DartFrog/tasks/public/task.json
Transformations_and_QAQC/DartFrog/tasks/public/task_run.json
Transformations_and_QAQC/DartFrog/tasks/first-internal/task.json
Transformations_and_QAQC/DartFrog/tasks/first-internal/task_run.json
Transformations_and_QAQC/DartFrog/tasks/final-internal/task.json
Transformations_and_QAQC/DartFrog/tasks/final-internal/task_run.json
Transformations_and_QAQC/DartFrog/tasks/sweeper-internal/task.json
Transformations_and_QAQC/DartFrog/tasks/sweeper-internal/task_run.json
Transformations_and_QAQC/DartFrog/tasks/missing/task.json
Transformations_and_QAQC/DartFrog/tasks/missing/task_run.json
```

4. Transform Data

As with the previous applications, a `task2shp.py` utility exists to aggregate information into a single spatial file. The commands used are as follows:

“

```
  $ ./Transformations_and_QAQC/DartFrog/bin/task2shp.py \
      Transformations_and_QAQC/DartFrog/tasks/public/task.json \
      Transformations_and_QAQC/DartFrog/tasks/public/task_run.json \
      Transformations_and_QAQC/DartFrog/transform/public/stats/dartfrog-
public-stats.shp

  $ ./Transformations_and_QAQC/DartFrog/bin/task2shp.py \
      Transformations_and_QAQC/DartFrog/tasks/first-internal/task.json \
      Transformations_and_QAQC/DartFrog/tasks/first-internal/task_run.json
\
      Transformations_and_QAQC/DartFrog/transform/first-
internal/stats/dartfrog-first-internal-stats.shp

  $ ./Transformations_and_QAQC/DartFrog/bin/task2shp.py \
      Transformations_and_QAQC/DartFrog/tasks/final-internal/task.json \
      Transformations_and_QAQC/DartFrog/tasks/final-internal/task_run.json
\
      Transformations_and_QAQC/DartFrog/transform/final-
internal/stats/dartfrog-final-internal-stats.shp

  $ ./Transformations_and_QAQC/DartFrog/bin/task2shp.py \
      Transformations_and_QAQC/DartFrog/tasks/sweeper-internal/task.json \
      Transformations_and_QAQC/DartFrog/tasks/sweeper-
internal/task_run.json \
      Transformations_and_QAQC/DartFrog/transform/sweeper-
internal/stats/dartfrog-sweeper-internal-stats.shp

  $ ./Transformations_and_QAQC/DartFrog/bin/task2shp.py \
      Transformations_and_QAQC/DartFrog/tasks/missing/task.json \
      Transformations_and_QAQC/DartFrog/tasks/missing/task_run.json \
      Transformations_and_QAQC/DartFrog/transform/missing/stats/dartfrog-
missing-stats.shp
```

The output file fields are as follows:

```
id          ->  Task's ID as assigned by PyBossa
site_id     ->  SkyTruth assigned unique site ID
wms_url     ->  Link to image scene
county      ->  County name
year        ->  Imagery year
location    ->  Generated unique ID (lat + long + year)
n_unk_res   ->  Number of task runs where task was classified as 'unknown'
n_frk_res   ->  Number of task runs where task was classified as 'fracking'
n_oth_res   ->  Number of task runs where task was classified as 'other'
n_tot_res   ->  Number of times this task was completed
crowd_sel   ->  Classification with the highest number of selections or
class1|class2|etc. for ties
qaqc        ->  Used in the manual QAQC process
p_crd_a     ->  Percent of the crowd's responses that matched the crowd_sel field
p_s_crd_a   ->  Percentages for crowd_sel ties percent1|percent2|etc. or NULL if
there was no tie
```

Due to the inherent complexity at the task level, an additional `taskCompiler.py` utility is included to reconstruct a given task's history. Each row is a single task and each column contains information about that task for every application. If a task was never completed in an application, then the values for that application will be NULL. This utility outputs 3 files:

```
Compiled_Output.csv   ->  Complete task history as CSV
Compiled_Output.json  ->  Complete task history as JSON
Scrubbed_Output.csv   ->  Final set of attributes for each task as CSV
```

In order to construct these files a priority/hierarchy must be assigned to each application, meaning that if a task is completed in multiple applications the final values come from the most preferred application:

1. Missing
2. Sweeper
3. Final Internal
4. First Internal
5. Public

Additional Notes

The primary key between any given set of task.json and task_run.json files is the `id` field in the task file and the `task_id` in the task run file. This is fine when working with one application's output but creates problems when working across applications. Since each task represents a pond in space-time, a unique location key can be generated from lat + long + year.

This works without issue EXCEPT that the lat/long precision was altered on the tasks that were moved to the first internal application, so in order to get a good match all lat/long values were rounded (via Python's round() function) to the 8th decimal place when generating location keys. This worked but about 50 ponds have a location that could not be matched with anything so they were not included in any subsequent analyses. In order to get a list of the ponds, re-run the `taskCompiler.py` utility, which prints out errors every time it encounters one of these ponds.

Fields

Note that the fields are the same for each application except for a few characters pre-pended to the field name to denote which application they represent:

```
location      ->  Location key as described above - generated on the fly
wms_url       ->  Final selection - WMS URL from task.json
lat           ->  Final selection - degree of latitude from task.json
lng           ->  Final selection - degree of longitude from task.json
year          ->  Final selection - year from task.json
county        ->  Final selection - county name from task.json
comp_loc      ->  Name of application the final attributes were selected from
n_frk_res     ->  Number of times the crowd classified the pond as 'fracking'
n_oth_res     ->  Number of times the crowd classified the pond as 'other'
n_unk_res     ->  Number of times the crowd classified the pond as 'unknown'
n_tot_res     ->  Total number of times a member of the crowd examined the task
(AKA the redundancy)
crowd_sel     ->  The classification the crowd chose for the pond
p_crd_a    ->  Percent of the crowd's responses that matched the crowd_sel field
p_s_crd_a  ->  Percentages for crowd_sel ties percent1|percent2|etc. or NULL if
there was no tie
```

Each application's attributes are denoted with a few leading characters:

```
p_   ->  Public (note that p_crd_a and p_s_crd_a will start with p_p_)
fi_  ->  First Internal
fn_  ->  Final Internal
sw_  ->  Sweeper
mt_  ->  Missing Tasks
```

6. Sample Data

The sampling routine was similar to the previous applications, however there are 5 applications and 3 years to sample from. Samples were split into applications and years with 100 samples for the public application and 50 for each internal. The QGIS random sample bug was also handled similarly to the previous applications. The sampling for this phase served a slightly different purpose as the output from DartFrog would be used as the input for the Digitizer, which is where we get one polygon representing each pond, but the Digitizer also serves as the last location where a human can look at a site and determine whether or not it is actually a fracking related pond. The goal of this round of sampling was to identify exactly what should go into the digitizer (without creating too much work for the digitizing analyst) so some initial data exploration was performed to determine whether or not the crowd agreement levels could be used as a threshold for determining which ponds are loaded into the digitizer. The threshold for the internal applications was determined to be 66% and 80% for the public application. The queries and exports are as follows:

```
Transformations_and_QAQC/DartFrog/transform/public/stats/dartfrog-public-
stats.shp
    "p_crd_a" >= 80 AND "n_tot_res" >= 10 AND "year" = 2008  ->
Transformations_and_QAQC/DartFrog/sampling/query/dartfrog-public-2008.shp
    "p_crd_a" >= 80 AND "n_tot_res" >= 10 AND "year" = 2010  ->
Transformations_and_QAQC/DartFrog/sampling/query/dartfrog-public-2010.shp

Transformations_and_QAQC/DartFrog/transform/first-internal/stats/dartfrog-first-
internal-stats.shp
    "p_crd_a" >= 66 AND "year" = 2005  ->
Transformations_and_QAQC/DartFrog/sampling/query/dartfrog-first-internal-
2005.shp
    "p_crd_a" >= 66 AND "year" = 2008  ->
Transformations_and_QAQC/DartFrog/sampling/query/dartfrog-first-internal-
2008.shp
    "p_crd_a" >= 66 AND "year" = 2010  ->
Transformations_and_QAQC/DartFrog/sampling/query/dartfrog-first-internal-
2010.shp

Transformations_and_QAQC/DartFrog/transform/final-internal/stats/dartfrog-final-
internal-stats.shp
    "p_crd_a" >= 66 AND "year" = 2005  ->
Transformations_and_QAQC/DartFrog/sampling/query/dartfrog-final-internal-
2005.shp
    "p_crd_a" >= 66 AND "year" = 2008  ->
Transformations_and_QAQC/DartFrog/sampling/query/dartfrog-final-internal-
2008.shp
    "p_crd_a" >= 66 AND "year" = 2010  ->
Transformations_and_QAQC/DartFrog/sampling/query/dartfrog-final-internal-
2010.shp

Transformations_and_QAQC/DartFrog/transform/sweeper-internal/stats/dartfrog-
sweeper-internal-stats.shp
    "p_crd_a" >= 66 AND "year" = 2005  ->
Transformations_and_QAQC/DartFrog/sampling/query/dartfrog-sweeper-internal-
2005.shp
    "p_crd_a" >= 66 AND "year" = 2008  ->
Transformations_and_QAQC/DartFrog/sampling/query/dartfrog-sweeper-internal-
2008.shp
    "p_crd_a" >= 66 AND "year" = 2010  ->
Transformations_and_QAQC/DartFrog/sampling/query/dartfrog-sweeper-internal-
2010.shp
```

Additional Notes: * 2005 Public has no random sample because all 2005 tasks were completed in the First Internal and Final Internal applications. * 2010 Final Internal has no random sampling because all of the 2010 tasks were completed in the Public or First Internal applications. * The Missing Tasks were only examined once and did not need to be randomly sampled since the one response constitutes the final classification.

The manually examined files are located in the following locations:

```
Transformations_and_QAQC/DartFrog/sampling/2008/dartfrog-public-2008-sample-100.shp
Transformations_and_QAQC/DartFrog/sampling/2010/dartfrog-public-2010-sample-100.shp
Transformations_and_QAQC/DartFrog/sampling/2005/dartfrog-first-internal-2005-sample-50.shp
Transformations_and_QAQC/DartFrog/sampling/2008/dartfrog-first-internal-2008-sample-50.shp
Transformations_and_QAQC/DartFrog/sampling/2010/dartfrog-first-internal-2010-sample-50.shp
Transformations_and_QAQC/DartFrog/sampling/2005/dartfrog-final-internal-2005-sample-50.shp
Transformations_and_QAQC/DartFrog/sampling/2008/dartfrog-final-internal-2008-sample-50.shp
Transformations_and_QAQC/DartFrog/sampling/2005/dartfrog-sweeper-internal-2005-sample-50.shp
Transformations_and_QAQC/DartFrog/sampling/2008/dartfrog-sweeper-internal-2008-sample-50.shp
Transformations_and_QAQC/DartFrog/sampling/2010/dartfrog-sweeper-internal-2010-sample-50.shp
```

Public Results

Sample size: 100

2005 - NO SAMPLE (see above section) 2008 - Disagreed with 18 pond classifications - due to "fat fingering" the sample size was 99 instead of 100. 2010 - Disagreed with 10 pond classifications

First Internal Results

Sample size: 50

2005 - Disagreed with 2 pond classifications 2008 - Disagreed with 5 pond classifications 2010 - Disagreed with 4 pond classifications

Final Internal Results

Sample size: 50

2005 - Disagreed with 0 pond classifications 2008 - Disagreed with 0 pond classifications 2010 - NO SAMPLE (see above section)

Sweeper Results

Sample size: 50

2005 - Sampled all 6 sites - disagreed with 0 pond classifications 2008 - Disagreed with 6 pond classifications 2010 - Disagreed with 6 pond classifications

Missing Tasks Results

NOT SAMPLED - See additional notes from `Sample Data` section above

Digitizer 2005-2010 Workflow

In order to expedite the process all digitizing was completed internally by SkyTruth staff.

General Description

1. Identify, generate, and load input tasks
2. Digitize
3. Export data
4. Transform data
5. Resolve intersecting ponds
6. Sample Data

1. Identify, Generate, and Load Input Tasks

Input tasks for the Digitizer were generated from DartFrog's output. Based on the sampling results and number of applications for DartFrog, a set of filters were determined in order to identify which tasks should be passed on to the digitizer. For tasks completed in the public application, any task classified as fracking with agreement level >= 80% was selected, and for all other application any task classified as fracking with an agreement level >= 66% was selected. In order to create the actual input JSON file, the `Compiled_Output.csv` was loaded into QGIS and filtered to identify task ID's that should be Digitizer input tasks. The tasks corresponding to these ID's were then extracted from the `Compiled_Output.json` file, loaded into their own JSON file, and uploaded to a new PyBossa application.

```
2005-2010/Transformations_and_QAQC/DartFrog/transform/Compiled_Output.csv

Filter for tasks completed in the public application:
    "comp_loc" = 'public' AND "crowd_sel" = 'fracking' AND "p_crd_a" >= 80 AND
"n_tot_res" >= 10

Filter for tasks completed in any of the internal applications
    "comp_loc" != 'public' AND "crowd_sel" = 'fracking' AND "p_crd_a" >= 66
```

2. Digitize

The final set of 896 ponds were loaded into a single application for digitizing with a redundancy of 1 and 4 actions: draw polygons and submit, skip pond, classify as unknown, and classify as not a fracking pond The analyst was instructed to digitize and classify tasks with an obvious answer, but skip any that were questionable or required a more in depth review. The ponds marked as skipped would then be loaded into a second digitizer and be reviewed by a panel of SkyTruth employees.

At around the 1/3 mark a bug was discovered in the digitizer that prevented multiple geometries from being recorded, which is not a problem for single ponds, but for multiple ponds in close proximity that are attached to a single clustered pond point, only the first drawn polygon was recorded. The solution to this problem involved two things: the analyst was instructed to now skip any multi-ponds while a patch was developed and deployed. The patch was successful but introduced an inconsistency in the output data by storing the polygons in a slightly different way than the original application. The difference is outlined in the code blocks below.

An additional bug was discovered around the 2/3 mark that was preventing any selection from being recorded if the triangle on the submit button was clicked. This bug was quickly patched.

After the initial digitizing pass, some ponds required a second pass. Initially this second pass was only expected to include ponds the analyst skipped, but due to the two discovered bugs, the second pass contained ponds with the following characteristics: ponds that were skipped and ponds lacking a selection key. The 181 ponds marked for re-examination were loaded and examined by a panel without issue.

Only the task_run.json file from the first digitizer exhibits inconsistencies. The output from the final application uses the patched structure.

Digitizer Bug: Original Info Key

NOTE: Each list of vertexes is nested within an additional unused list 'info': {'done': {'tasks': 1},
'selection': 'done',
'shape': {'coordinates': [[[-79.77771873394, 40.129319804207],
[-79.777383349799, 40.1294248643],

[-79.777173229614, 40.12900462393],
[-79.777534878778, 40.128887441519],
[-79.77771873394, 40.129319804207]]],
'type': 'Polygon'},
'timings': {'presentTask': 231250, 'reportAnswer': 231596}}

Digitizer Bug: Patched Info Key

For sites with a single pond, the shapes key contains a list with a single element.

NOTE: Each list of vertices is nested within an additional unused list 'info': {'done': {'tasks': 1}, 'selection': 'done', 'shapes': [{'coordinates': [[[-77.916892032231, 41.871592973047], [-77.916578816183, 41.871451459773], [-77.916582589871, 41.871385420245], [-77.916725989989, 41.871228812221], [-77.917103358721, 41.871421270274], [-77.916935429635, 41.871592973047], [-77.916892032231, 41.871592973047]]], 'type': 'Polygon'}, {'coordinates': [[[-77.917186379842, 41.87189298119], [-77.916944863854, 41.871753354759], [-77.916956184916, 41.8716816547], [-77.917199587748, 41.871485422959], [-77.917450537955, 41.871694862605], [-77.917235437778, 41.871906189095], [-77.917186379842, 41.87189298119]]], 'type': 'Polygon'}], 'timings': {'presentTask': 73966, 'reportAnswer': 74292}}

3. Export Data

Due to bugs in the digitizer, the tasks were completed in two applications, `first-review` and `final-review`. The data was exported and stored in the following locations:

```
Transformations_and_QAQC/Digitizer/tasks/first-review/task.json
Transformations_and_QAQC/Digitizer/tasks/first-review/task_run.json
Transformations_and_QAQC/Digitizer/tasks/final-review/task.json
Transformations_and_QAQC/Digitizer/tasks/final-review/task_run.json
```

4. Transform Data

Similar to the previous applications, a `task2shp.py` utility is included to handle all the necessary transforms and data aggregations. The output data structure is similar to the other utilities that serves the same purpose but the features are polygons representing ponds. The input data is slightly different in that the other `task2shp.py` utilities require both the task.json and task_run.json, this one ONLY takes a modified task_run.json file. The next steps prepare that file for processing.

Before running the `task2shp.py` utility, a few pre-processing steps must be done. The first takes the task_run.json and adds all the associated task.json attributes to each task run:

“

```
  $ ~/GitHub/CrowdProjects/bin/mergeExport.py \
      Transformations_and_QAQC/Digitizer/tasks/first-review/task.json \
      Transformations_and_QAQC/Digitizer/tasks/first-review/task_run.json\
      Transformations_and_QAQC/Digitizer/transform/first-review/tasks-with-
added-fields/task_run_added_fields.json

  $ ~/GitHub/CrowdProjects/bin/mergeExport.py \
      Transformations_and_QAQC/Digitizer/tasks/final-review/task.json \
      Transformations_and_QAQC/Digitizer/tasks/final-review/task_run.json \
      Transformations_and_QAQC/Digitizer/transform/final-review/tasks-with-
added-fields/task_run_added_fields.json
```

Additionally, a classification field was added to the task.json and task_run.json to note which digitizer it passed through.

NOTE: The overwrite flag on two of the commands is to allow the utility to add the field to the file in place.

“

```
  $ ~/GitHub/CrowdProjects/bin/editJSON.py \
      -a class=first \
      Transformations_and_QAQC/Digitizer/tasks/first-review/task.json \
      Transformations_and_QAQC/Digitizer/transform/first-review/tasks-with-
added-fields/task_added_fields.json

  $ ~/GitHub/CrowdProjects/bin/editJSON.py \
      --overwrite \
      -a class=first \
      Transformations_and_QAQC/Digitizer/transform/first-review/tasks-with-
added-fields/task_run_added_fields.json \
      Transformations_and_QAQC/Digitizer/transform/first-review/tasks-with-
added-fields/task_run_added_fields.json

  $ ~/GitHub/CrowdProjects/bin/editJSON.py \
      -a class=final \
      Transformations_and_QAQC/Digitizer/tasks/final-review/task.json \
      Transformations_and_QAQC/Digitizer/transform/final-review/tasks-with-
added-fields/task_added_fields.json

  $ ~/GitHub/CrowdProjects/bin/editJSON.py \
      --overwrite \
      -a class=final \
      Transformations_and_QAQC/Digitizer/transform/final-review/tasks-with-
added-fields/task_run_added_fields.json \
      Transformations_and_QAQC/Digitizer/transform/final-review/tasks-with-
added-fields/task_run_added_fields.json
```

Tasks that were completed in both applications will be manually resolved later so the task.json and task_run.json can now be combined. The previous step added a classification field that can be used to determine which application a given task or task run came from.

“

```
  $ ~/GitHub/CrowdProjects/bin/mergeFiles.py \
      Transformations_and_QAQC/Digitizer/transform/first-review/tasks-with-
added-fields/task_run_added_fields.json \
      Transformations_and_QAQC/Digitizer/transform/final-review/tasks-with-
added-fields/task_run_added_fields.json \
      Transformations_and_QAQC/Digitizer/derivative-
data/Merged_Task_Runs.json
```

Now that the task run files have been combined the data can be handed off to the `task2shp.py` utility, which converts the JSON to pond polygons with attributes.

“

```
  $ Transformations_and_QAQC/Digitizer/bin/task2shp.py \
      --process-extra-fields \
      Transformations_and_QAQC/Digitizer/derivative-
data/Merged_Task_Runs.json \
      Transformations_and_QAQC/Digitizer/derivative-data/deliverable-ponds-
candidate.shp
```

While the input for `task2shp.py` is slightly different than the other utilities, the output fields are similar:

```
selection  ->  Unused and included by mistake
task_id    ->  PyBossa assigned task_id from task_run.json (matches
task.json['task_id'])
intersect  ->  Says if this pond intersects with another - used during the
QAQC/resolution step
comp_loc   ->  Which DartFrog application this pond was classified in
crowd_sel  ->  Classification with the highest number of selections or
class1|class2|etc. for ties
county     ->  County name
state      ->  State name
year       ->  Imagery year
location   ->  Generated unique ID (lat + long + year)
area_m     ->  Pond area in meters
delete     ->  Used during the QAQC/resolution step
```

5. Resolve Intersecting Ponds

Due to the nature of the data and the bugs discovered in the digitizer application, some ponds intersect other ponds in the file. In some cases this intersection is caused by the pond existing for multiple years, but some ponds were actually digitized twice. All of the ponds marked as intersecting with another feature were manually examined and marked for deletion, but only if it was digitized twice. Of the ponds that were digitized twice, some were digitized in both the first and final applications and some were digitized twice in a single application. For ponds that were digitized in both applications the pond digitized in the first application was marked for deletion and the pond digitized in the second was kept. For ponds that were digitized twice in a single application, an analyst examined both to determine which should be kept.

6. Sample Data

A random subsample of 100 digitized ponds were manually examined to verify both the data transformations explained above and the quality of the data. The main goal was to ensure that the attributes were properly copied from the merged task_run.json to the proper pond. Imagery was also used to verify the geometry. No issues were encountered.

FrackFinder PA 2013

Source Data

The source data for 2013 was collected and processed exactly like 2005-2010, except a more inclusive filter was applied to include any clustered site where at least one associated permit was for an unconventional well within years of 2005-2013. This means that all of the permits examined in FrackFinder 2005-2010 were examined again in FrackFinder 2013 against the 2013 imagery. This helps build a more complete history for any given site.

The task creation method and application were used from 2005-2010.

Tadpole 2013 Workflow

General Description

1. Download tasks
2. Transform data
3. Sample/QAQC

1. Download Data

The data was split between two different applications, `public` and `internal`, each containing half of the Tadpole input tasks. The exported tasks and task runs were placed in:

```
Transformations_and_QAQC/Tadpole/tasks/public/task.json
Transformations_and_QAQC/Tadpole/tasks/public/task_run.json
Transformations_and_QAQC/Tadpole/tasks/internal/task.json
Transformations_and_QAQC/Tadpole/tasks/internal/task_run.json
```

2. Transform Data

Since all tasks were only completed in one location the data can be combined, however we want to be able to know where each task came from so a classification was added in the `ST_source` field.

“

```
  $ ~/GitHub/CrowdProjects/bin/editJSON.py \
      -a ST_source=public
Transformations_and_QAQC/Tadpole/tasks/public/task.json \
      Transformations_and_QAQC/Tadpole/tasks/added_fields/public/added-
fields-task.json

  $ ~/GitHub/CrowdProjects/bin/editJSON.py \
      -a ST_source=public \
      Transformations_and_QAQC/Tadpole/tasks/public/task_run.json \
      Transformations_and_QAQC/Tadpole/tasks/added_fields/public/added-
fields-task_run.json

  $ ~/CrowdProjects/bin/editJSON.py \
      -a ST_source=internal \
      Transformations_and_QAQC/Tadpole/tasks/internal/task.json \
      Transformations_and_QAQC/Tadpole/tasks/added_fields/internal/added-
fields-task.json

  $ ~/CrowdProjects/bin/editJSON.py \
      -a ST_source=internal \
      Transformations_and_QAQC/Tadpole/tasks/internal/task_run.json \
      Transformations_and_QAQC/Tadpole/tasks/added_fields/internal/added-
fields-task_run.json
```

After adding the source field, the data was combined into a single task.json and task_run.json:

```
  $ ~/GitHub/CrowdProjects/bin/mergeFiles.py \
      Transformations_and_QAQC/Tadpole/tasks/added_fields/public/added-
fields-task.json \
      Transformations_and_QAQC/Tadpole/tasks/added_fields/internal/added-
fields-task.json \
      Transformations_and_QAQC/Tadpole/tasks/combined-task.json

  $ ~/GitHub/CrowdProjects/bin/mergeFiles.py \
      Transformations_and_QAQC/Tadpole/tasks/added_fields/public/added-
fields-task_run.json \
      Transformations_and_QAQC/Tadpole/tasks/added_fields/internal/added-
fields-task_run.json \
      Transformations_and_QAQC/Tadpole/tasks/combined-task_run.json
```

These combined files can now be converted into a single shapefile:

```
  $ ./Transformations_and_QAQC/Tadpole/bin/task2shp.py \
      Transformations_and_QAQC/Tadpole/tasks/combined-task.json \
      Transformations_and_QAQC/Tadpole/tasks/combined-task_run.json \
      Transformations_and_QAQC/Tadpole/transform/stats/tadpole-stats.shp \
      --class=%ST_source
```

Field Definitions:

```
id  ->  task.json['id']
site_id  ->  task.json['info']['siteID']
wms_url  ->  URL for appropriate imagery
wms_id  ->  Layer ID/name for NAIP imagery
wms_v  ->  OGC WMS version
county  ->  County name
state  ->  State abbreviation
year  ->  Year original permits were created
location  ->  Generated primary key based on lat + long + year
n_unk_res  ->  Number of task runs marked as 'unknown' - task_run.json['info']
['selection']
n_nop_res  ->  Number of task runs marked as 'nopad' - task_run.json['info']
['selection']
n_pad_res  ->  Number of task runs marked as 'pad' - task_run.json['info']
['selection']
n_tot_res  ->  Total number of times a task was completed in task_run.json
crowd_sel  ->  The most picked task_run.json['info']['selection'] - some are
split with '|'
qaqc  ->  String field for use when performing QAQC
p_crd_a  ->  Percentage of responses that agreed with the crowd_sel field
p_s_crd_a  ->  If two selections were favored equally, the p_crd_a for each
separated by '|'
class  ->  User defined classification - in this case it specifies public vs.
internal
```

3. Sample/QAQC

A random sample size of 5% for each application was selected after exploring the data. A total of `4241` tasks were were completed, half in the public application and half in an internal application, yielding `106` samples per application. Each task in the public application was completed 10 times and each task in the internal application was completed 3 times.

After exploring the `Transformations_and_QAQC/Tadpole/transform/stats/tadpole-stats.shp` file, a sample size of 5% was selected based on the fact that `93%` of tasks had an agreement level greater than or equal to 70%, which justifies sampling the entire population rather than a subset.

The random sampling tool in `QGIS` was used to generate a test dataset, however a bug requires a small pre-processing step. For some reason the random sampling tool totally ignores any datasource filters, so the data must first be queried and saved to a new file before performing the selection. The queries and output files are as follows:

```
"class" = 'public'   ->
Transformations_and_QAQC/Tadpole/sampling/queries/tadpole-public.shp
"class" = 'internal' ->
Transformations_and_QAQC/Tadpole/sampling/queries/tadpole-internal.shp
```

The subsequent files were loaded into `QGIS` and a sample of `5%` for each application was selected with the random sampling tool, yielding `106` features for each layer. The selections were saved in the following locations:

```
Transformations_and_QAQC/Tadpole/sampling/public/tadpole-public-5percent-
sample.shp
Transformations_and_QAQC/Tadpole/sampling/internal/tadpole-internal-5percent-
sample.shp
```

The samples were then examined one at a time by an analyst in order to determine whether they were properly classified or not

Sampling Results

The analyst disagreed with 1/106 samples in the public set and 8/106 in the internal set. The higher number of disagreements in the internal set is likely caused by the redundancy. The public examined each task 10 times but the internal tasks were only examined 3 times, which means that the public's overall selection allowed for a higher level of disagreement. If 8 out of 10 people in the `public` crowd agreed that a site should be classified as a pad, they agreed at 80%, but if 2 out of 3 people in the `internal` crowd agreed, their confidence is only 66%. The public is much less sensitive to an individual's selection. After examining all 8 disagreements, half were found to be totally ambiguous and the other half were found to be remediated pads.

The results show that both the internal and public crowd were very confident in their selections and the data could be used without any corrections. The input tasks for MoorFrog were determined to be as follows:

```
Transformations_and_QAQC/Tadpole/transform/stats/tadpole-stats.shp
    "p_crd_a >= 66 AND crowd_sel = 'pad' AND class = 'internal'"
    "p_crd_a >= 70 AND crowd_sel = 'pad' AND class = 'public'"
```

MoorFrog 2013 Workflow

In order to expedite the process all of MoorFrog was completed internally by SkyTruth staff.

General Description

1. Generate input tasks
2. Load into application
3. Classify ponds
4. QA/QC

1. Generate Input Tasks

An explanation of how the input task queries were determined can be found above in Tadpole 2013's Sample/QAQC section.

Output from `Tadpole` was processed into a set of input tasks for MoorFrog using the following commands:

```
  $ ./Transformations_and_QAQC/MoorFrog/bin/taskGenerator.py \
      Transformations_and_QAQC/Tadpole/transform/stats/tadpole-stats.shp \
      Transformations_and_QAQC/MoorFrog/input_tasks/from_internal.json \
      -query "p_crd_a >= 66 AND crowd_sel = 'pad' AND class = 'internal'"
      --add-info-class=internal

  $ ./Transformations_and_QAQC/MoorFrog/bin/taskGenerator.py \
      Transformations_and_QAQC/Tadpole/transform/stats/tadpole-stats.shp \
      Transformations_and_QAQC/MoorFrog/input_tasks/from_public.json \
      -query "p_crd_a >= 70 AND crowd_sel = 'pad' AND class = 'public'" \
      --add-info-class=public

  $ ~/GitHub/CrowdProjects/bin/mergeFiles.py \
      Transformations_and_QAQC/MoorFrog/input_tasks/from_internal.json \
      Transformations_and_QAQC/MoorFrog/input_tasks/from_public.json \

 Transformations_and_QAQC/MoorFrog/input_tasks/combined_input_tasks.json
```

2. Load Tasks

Tasks were loaded with the `createTasks.py` utility, which can be found in the pybossa_tools repository. The upload command is as follows:

“

```
  $ createTasks.py \
      -v -q 3 -n 3 -c \
      -a finder-pond \
      -s http://crowd.dev.skytruth.org \
      -k {YOUR_API_KEY} \
      -t
~/GitHub/CrowdProjects/Data/FrackFinder/PA/2013/Transformations_and_QAQC/Mo
orFrog/input_tasks/combined_input_tasks.json \
      -r "MoorFrog_2013_Fracking_Ponds_Only2"
```

3. Classify Ponds

This phase was completely done by SkyTruth employees so instead of clicking on ANY pond, users were asked to ONLY click on fracking related ponds.

4. QA/QC

The tasks and task runs were transformed using the usual application specific `task2shp.py` utility:

“

```
  $ ./2013/Transformations_and_QAQC/MoorFrog/bin/task2shp.py \
          2013/Transformations_and_QAQC/MoorFrog/output_tasks/task.json \
          2013/Transformations_and_QAQC/MoorFrog/output_tasks/task_run.json
\
          2013/Transformations_and_QAQC/MoorFrog/transform/layers/
```

The resulting dataset was examined in QGIS and it was determined that the best course of action was to cluster all clicks and pass the resulting centroids into the next application, which is described in the `Generate Input Tasks` section.

DartFrog 2013 Workflow

DartFrog was skipped for PA 2013. Instead MoorFrog users (SkyTruth staff) were asked to ONLY click on ponds that appeared to be fracking related.

Digitizer 2013 WorkFlow

In order to expedite the process all digitizing was completed internally by SkyTruth staff.

General Description

1. Generate input tasks
2. Load into application
3. Digitize
4. Export data
5. Transform data
6. Resolve intersecting ponds

1. Generate Input Tasks

The output from MoorFrog contains an inefficient number of points for digitizing purposes so they were clustered using a simple nearest neighbor algorithm and QGIS. The buffer distance was determined by trial and error, the goal being to reduce the total number of tasks the digitizer analyst was given without clustering too many ponds together into single tasks.

1. Load into QGIS: `2013/Transformations_and_QAQC/MoorFrog/transform/layers/MoorFrog-clicks.shp` and use the built in vector buffer tool:
    - Segments to approximate: 5
    - Buffer distance: 0.0001 (input file units are degrees)
    - Dissolve buffer results: True
    - Output: 2013/Transformations_and_QAQC/MoorFrog/transform/buffer-dissolve-clicks.shp
2. The previous step yields a single feature collection containing all pond geometries. Explode the collection into multiple features:

```
  $ ogr2ogr \
      -explodecollections
      2013/Transformations_and_QAQC/MoorFrog/transform/exploded-
clicks.shp \
      2013/Transformations_and_QAQC/MoorFrog/transform/buffer-
dissolve-clicks.shp
```

3. Load into QGIS: `2013/Transformations_and_QAQC/MoorFrog/transform/exploded-clicks.shp` and use the the centroid tool:
    - Output: 2013/Transformations_and_QAQC/MoorFrog/transform/pond-centroids.shp
4. Load into QGIS: `2013/Transformations_and_QAQC/MoorFrog/transform/pond-centroids.shp` and use the field calculator to add some attributes needed for task generation:
    - Field Name: X
        - Type: Real
        - Width: 10
        - Precision: 10
        - Expression: $X
    - Field Name: Y
        - Type: Real
        - Width: 10
        - Precision: 10
        - Expression: $Y
    - Field Name: longitude
        - Type: Real
        - Width: 10
        - Precision: 10
        - Expression: $X
    - Field Name: latitude
        - Type: Real
        - Width: 10
        - Precision: 10
        - Expression: $Y

The resulting shapefile is generalized and doesn't contain any attribute information. The digitizer requires the county name, which can be joined from a county boundary shapefile. Pond centroids from MoorFrog were loaded into QGIS along a Pa County vector layer.

```
2013/Transformations_and_QAQC/MoorFrog/transform/pond-centroids.shp
2013/Transformations_and_QAQC/MoorFrog/transform/PACounties/PACounty2013.shp
```

Pond centroid attributes were joined with Pa County attributes and exported:

```
2013/Transformations_and_QAQC/MoorFrog/transform/pond-centroids-attribute-join.shp
```

In QGIS, all unwanted attribute fields were removed and the layer was exported as a CSV:

```
2013/Transformations_and_QAQC/MoorFrog/transform/pond-centroids.csv
```

The CSV was converted to json format using the skyconvert tool and additional attributes were added using `finalizeInputTasks.py`

> ```
> $ skyconvert \
>     transform/pond-centroids.csv
>     ../Digitizer/input_tasks/input_no_wms.json
>
> $ 2013/Transformations_and_QAQC/Digitizer/bin/finalizeInputTasks.py \
>     input_tasks/input_no_wms.json \
>     input_tasks/input_with_wms.json
> ```

2. Load Tasks

Tasks were loaded with the `createTasks.py` utility, which can be found in the pybossa_tools repository. The installation procedure is as follows:

```
  $ ./createTasks.py \
      -n 1 -c
      -s http://crowd.skytruth.org \
      -k "{YOUR_API_KEY}" \
      -a digitizer-pond \
      -r Digitizer-pond_PA2013_internal-1 \
      -t
~/CrowdProjects/Data/FrackFinder/PA/2013/Transformations_and_QAQC/Digitizer
/input_tasks/input_with_wms.json \
```

3. Digitize

Digitizing was completed by a SkyTruth employee who was shown a scene centered around a pond that is suspected to be fracking related. If the analyst determines that the pond is indeed fracking related, they draw a polygon around the pond and move to the next task. If they find the pond is clearly not fracking related they select "Not fracking related" and move on. If they can't tell, they click "Not sure".

4. Export Data

The data was exported and stored in the following location.

```
2013/Transformations_and_QAQC/Digitizer/output_tasks/task.json
2013/Transformations_and_QAQC/Digitizer/output_tasks/task_run.json
```

5. Transform Data

The `task_run.json` file was converted to a spatial format:

“

```
  $ 2013/Transformations_and_QAQC/Digitizer/bin/task2shp.py \
      2013/Transformations_and_QAQC/Digitizer/output_tasks/task_run.json \
      2013/Transformations_and_QAQC/Digitizer/transform/digitized_ponds.shp
```

6. Resolve Intersecting Ponds

Due to the nature of the data a certain number of the digitized ponds were found to be intersecting other ponds in the layer. These intersections were resolved manually by examining each occurrence and selecting the pond with the better geometry. The resulting output was stored in a separate file:

```
2013/Transformations_and_QAQC/Digitizer/transform/resolved_intersects.shp
```