Radiant Earth - SkaMo winning models' documentation

This documents the SkaMo team's models used to win both the

- XL s1+s2 and the
- s2 only

competitions.

MG Ferreira and Motoki Saitama
October 2021

Environment

The models run in a modest environment.

We used an old Macbook Air 2017 (so *macos* operating system) with a dual core Intel i5 CPU and 8G of RAM and it took roughly 1 day (24h) to complete a model.

The reason for long duration is because we made heavy use of *catboost* which is slow but which also seems to keep on improving the longer you let it run. So we used 10000(!) catboost iterations for each model.

The software we used include the latest python3, lightgbm and catboost.

At the time of writing these were as shown below.

```
python3 --version
Python 3.9.7
```

```
pip3 list                     # Only showing relevant packages
Package                       Version
catboost                      0.26.1
lightgbm                      3.2.1
numpy                         1.21.2
radiant-mlhub                 0.2.2
rasterio                      1.2.6
scikit-image                  0.18.3
scikit-learn                  0.24.2
```

Step 1 - prepare

The first step is to download the data and is detailed in the competition documentation.

If you do this you end up with the relevant data files in a directory called

`<model root>/South Africa Crop Types Competition`

Here the `<model root>` is considered the root directory of the models. If you downloaded it the same way we did, it should be `mlhub-tutorials/notebooks` .

Our code will create feature and submission directories in this model root, so you will end up with

`<model root>/features`

and

`<model root>/sub`

directories after running the code.

A good place to put the code files is in the

`<model root>/code`

directory but you need to run it from the

`<model root>/South Africa Crop Types Competition`

directory.

The output below shows what you should get from listing the files from the `<model root>` and is given here as a reference.

```
ls -l1 South\ Africa\ Crop\ Types\ Competition
```

```
ref_south_africa_crops_competition_v1_test_labels
ref_south_africa_crops_competition_v1_test_source_s1
ref_south_africa_crops_competition_v1_test_source_s2
ref_south_africa_crops_competition_v1_train_labels
ref_south_africa_crops_competition_v1_train_source_s1
ref_south_africa_crops_competition_v1_train_source_s2
```

This indicates that you have downloaded the data correctly and are ready to proceed.

git

We checked out the repository using `git` and then used the supplied notebook to download the data.

To ensure we are on the same page, some `git` details below.

```
git remote -v
```

```
origin  https://github.com/radiantearth/mlhub-tutorials (fetch)
origin  https://github.com/radiantearth/mlhub-tutorials (push)
```

```
git log
```

```
commit 0c043829f25ac3380f3960dcb87b87d3f0551195 (HEAD -> main, origin/main, origin
/HEAD)
Merge: 08695f0 c429d64
Author: Hamed Alemohammad
Date:   Mon Jul 5 12:49:49 2021 -0400

    Merge pull request #62 from radiantearth/dev

    Dev

commit c429d64c962abbd80e4a68f8eb832891be14b0fe (origin/dev)
Merge: 851e751 9d92b23
Author: Hamed Alemohammad
Date:   Mon Jul 5 12:49:13 2021 -0400

    Merge pull request #61 from radiantearth/south-africa-crops-notebooks

    minor update to SA model notebook
...
```

Step 2 - code files

We provide 3 files that you need to execute. They are

```
main4.py          # To create the feature matrices and preprocess the data
model4-3.py       # To fit the s1+s2 models and write their submission files
model4-3s2.py     # To fit the s2 only models and write their submission files
```

A good strategy is to copy them into the

`<model root>/code`

directory. The documentation below assumes you did this, but note, you can keep the code anywhere and

as long as you run it from the

`<model root>/South Africa Crop Types Competition`

you should be fine.

Step 3 - preprocess the data

The next step is to process the data.

This step is the same for both competitions and need be run only once.

This will yield a number of output files in the

`<model root>/features`

directory.

Simply run the `model4.py` file from the

`<model root>/South Africa Crop Types Competition`

directory, e.g. do something like

```
cd South\ Africa\ Crop\ Types\ Competition
python3 ../code/main4.py
```

to complete this step.

The data is big and this step will take a few hours to complete.

The output, that you should see as it processes, is shown below.

```
Building dictionaries
    Tiles in training set
    Tiles in test set
Writing tiles
Writing fields
Updating overlapped fields
...
    Now at 122381/122408 - have noted 47744062 neighbours thus far
    Now at 122391/122408 - have noted 47747028 neighbours thus far
    Now at 122401/122408 - have noted 47750159 neighbours thus far
Done - recorded 47753342 neighbours within a 0.05 radius, an average of 390.116185
2166525 neighbours per field
```

When this step is done you can proceed to the next.

Step 4 - fit the model

To fit our model, once the data is ready, simply run the `model4-3.py` file for the XL competition or the `model4-3s2.py` for the s2-only competition.

XL s1+s2 competition

Do something like

```
cd South\ Africa\ Crop\ Types\ Competition
python3 ../code/model4-3.py
```

to fit the model.

The output will look something like this

```
Reading crop labels
Reading fields
    Creating lookup
    Field ID and row mappings
    Loading known labels
Reading support data
    There are 122408 rows and 16 dates and 18 bands available
Calculating image features
Loading other features
Saving preliminary feature matrix
Fitting preliminary model
    ----[ Fold 1/10 ]-----------------------------------------------------------
        Fitting lgb
            0.9025437992469454
        Fitting cat
            0.8376056615378181
    ----[ Fold 2/10 ]-----------------------------------------------------------
        Fitting lgb
            0.9113602657080594
...
Fitting dirty model
    ----[ Fold 1/10 ]-----------------------------------------------------------
    Making feature matrix
        Satellite and other features
        Neighbourhood distributions
        Calculating field overlap information
    Features ready
        Fitting lgb
...
```

```
            0.5770006267797964
    ----[ Fold 10/10 ]----------------------------------------------------------
    Making feature matrix
        Satellite and other features
        Neighbourhood distributions
        Calculating field overlap information
    Features ready
        Fitting lgb
            0.5531502373226709
        Fitting cat
            0.5809213253489299
    Writing dirty results as submission file
        lgb
        cat
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~
Done
    Optimal mix of lgb and cat
        lgb = 1 cat = 0 loss = 0.5567254623420373 ***
        lgb = 0.95 cat = 0.050000000000000044 loss = 0.5531519951587904 ***
        lgb = 0.9 cat = 0.09999999999999998 loss = 0.5513494941136995 ***
        lgb = 0.85 cat = 0.15000000000000002 loss = 0.5503010095807532 ***
        lgb = 0.8 cat = 0.19999999999999996 loss = 0.5498062717007617 ***
        lgb = 0.75 cat = 0.25 loss = 0.5497648370548124 ***
        lgb = 0.7 cat = 0.30000000000000004 loss = 0.5501169984416008
        lgb = 0.6499999999999999 cat = 0.3500000000000001 loss = 0.550824526934383
        lgb = 0.6 cat = 0.4 loss = 0.5518623688025411
        lgb = 0.55 cat = 0.44999999999999996 loss = 0.5532145047329141
        lgb = 0.5 cat = 0.5 loss = 0.554871732833672
        lgb = 0.44999999999999996 cat = 0.55 loss = 0.5568304886999647
        lgb = 0.3999999999999999 cat = 0.6000000000000001 loss = 0.5590923198020011
        lgb = 0.35 cat = 0.65 loss = 0.5616638618394149
        lgb = 0.29999999999999993 cat = 0.7000000000000001 loss = 0.5645573050837265
        lgb = 0.25 cat = 0.75 loss = 0.5677914735917267
        lgb = 0.19999999999999996 cat = 0.8 loss = 0.5713938508549515
        lgb = 0.1499999999999999 cat = 0.8500000000000001 loss = 0.5754043459226567
        lgb = 0.09999999999999998 cat = 0.9 loss = 0.5798828806588212
        lgb = 0.04999999999999993 cat = 0.9500000000000001 loss = 0.5849277004179452
        lgb = 0.0 cat = 1.0 loss = 0.590744073804675
Dirty model results
OOF
    lgb  : 0.5567254623420373
    cat  : 0.590744073804675
    both : 0.5497648370548124
Done!
```

Note! Values here are indicative only and generated for illustration. You should get slightly different results on-screen as this runs.

On modest hardware, this will take roughly a day (24h) to complete so be prepared to wait.

We did not use colab or a GPU.

s2 only competition

This is very similar, except that the model now resides in the `model4-3s2.py` file, so, after preparing the data as outlined in the previous step, run the model as described below.

```
cd South\ Africa\ Crop\ Types\ Competition
python3 ../code/model4-3s2.py
```

This will also take roughly a day to complete.

Output files

The output file naming convention is explained below.

s2 prefix

The s2 only competition will have the same files as the XL (s1+s2 competition) but with prefix `s2-` added to the file's name.

l, c and b prefix

A model typically has three flavours

- a lightgbm version
  - these are prefixed with `l` for lightgbm
- a catboost version
  - these are prefixed wtih `c` for catboost
- some weighted combination of lightboost and catboost
  - these are prefixed with `b` for both light and cat

a and t postfix

Once a model is done, an output file is generated where the model is used to classify all the data, including the training set, and just the test set.

- if the output is written for all the data then the file name is postfixed with `a`
- if the output is only for the test set then the name is postfixed with a `t`

The `a` files are later used to calculate the weight that will be assigned to each of the catboost and

lightgbm based models to create the `b` or `both` models later on.

preliminary models

The model is built in stages, the first stage being the *preliminary* one. Models that are preliminary will have a postfix of `p`.

final models

Once the preliminary models are ready, they are augmented with neighbourhood and overlap information and become *final* models. These will have a postfix of `f`.

The final models' cross validation score is contaminated and can not be used as is.

out-of-fold models

For the *out-of-fold* models, the preliminary model is used to generate the out-of-fold classification of the fields in the neighbourhood and overlap around the field being classified. These will have a `o` as postfix.

These models are *clean* in that the cross validation score is not contaminated and should be very close to the leaderboard score.

dirty models

The so-called *dirty* models are those where actual data from the train set is used to classify models in the overlap and neighbourhood around the field.

In these models, the local cross-validation score is contaminated and can not be trusted, thus they are called *dirty*. In the end, we relied on the public leaderboard to evaluate these models.

List of submission files

The table below gives an example of some of the submission file names.

Filename	Meaning
	Preliminary models
`c4-3pa.csv`	Catboost preliminary model for all (test and train) data
`c4-3pt.csv`	Catboost preliminary model for the test set
`l4-3pa.csv`	LightGBM preliminary model for all (test and train)
`l4-3pt.csv`	LightGBM preliminary model for the test set
`b4-3pa.csv`	Combined (using *b*oth catboost and lightgbm) preliminary model for all data

`b4-3pt.csv`	Combined preliminary model for test set
	Final models
`c4-3fa.csv`	Catboost final model for all (test and train) data
`c4-3ft.csv`	Catboost final model for the test set
`l4-3fa.csv`	LightGBM final model for all (test and train)
`l4-3ft.csv`	LightGBM final model for the test set
`b4-3fa.csv`	Combined (using *b*oth catboost and lightgbm) final model for all data
`b4-3ft.csv`	Combined final model for test set
	Out-of-fold models
`c4-3oa.csv`	Catboost out-of-fold model for all (test and train) data
`c4-3ot.csv`	Catboost out-of-fold model for the test set
`l4-3oa.csv`	LightGBM out-of-fold model for all (test and train)
`l4-3ot.csv`	LightGBM out-of-fold model for the test set
`b4-3oa.csv`	Combined (using *b*oth catboost and lightgbm) out-of-fold model for all data
`b4-3ot.csv`	Combined out-of-fold model for test set
	Dirty models
`c4-3da.csv`	Catboost dirty model for all (test and train) data
`c4-3dt.csv`	Catboost dirty model for the test set
`l4-3da.csv`	LightGBM dirty model for all (test and train)
`l4-3dt.csv`	LightGBM dirty model for the test set
`b4-3da.csv`	Combined (using *b*oth catboost and lightgbm) dirty model for all data
`b4-3dt.csv`	Combined dirty model for test set *(and our final submission)*

For the s2 only competition, the same names are used but with prefix `s2-` added.

For the XL competion, we selected the following two models

- `b4-3dt.csv`
- `b4-3ot.csv`

and for the s2-only competition, we selected the s2-only versions of these, being

- `s2-b4-3dt.csv`
- `s2-b4-3ot.csv`

Validating our models

Since you asked to validate

- `b4-3dt.csv`

for the XL s1+s2 competition and

- `s2-b4-3dt.csv`

for the s2 only competition, I have switched off the *final* and *out-of-fold* models and only switched on the *dirty* models to speed things up. The *preliminary* models will also build regardless.

These two submission files should be in the `sub` directory once the programs are done.

They will be there together with a number of other files so, when you validate, be sure you use the correct files.

For reference, the files we submitted are

Competition	Submission file
XL s1 and s2	`<model root>/sub/b4-3dt.csv`
s2 only	`<model root>/sub/s2-b4-3dt.csv`

Features

I'd like to discuss the features we used to model next.

Satellite features

Cloud cover

We discarded any pixel with any cloud cover over it.

Date

We grouped the photos into half-months. All photos taken on or before the 15th of April, for example, would

be added to the `4a` bucket and all taken after the 15th, but in April, were placed into the `4b` bucket and so on for all the months. Thus we had the following 16 *dates* in our model.

- 4a
- 4b
- 5a
- 5b
- 6a
- 6b
- 7a
- 7b
- 8a
- 8b
- 9a
- 9b
- 10a
- 10b
- 11a
- 11b (16 in total)

Bands

We used all the bands and also calculated a few derived bands using well known formulae. Thus for the s1+s2 competition, we used the following.

- s2
  - B01
  - B02
  - B03
  - B04
  - B05
  - B06
  - B07
  - B08
  - B09
  - B11
  - B12
  - B8A
  - EVI = 2.5 x (B08 - B04) / (B08 + 6 x B04 - 7.5 x B01 + 1)
  - MI = (B08A - B11) / (B04 + B08A)
  - NDVI = (B08 - B04) / (B04 + B08)
  - SAVI = (B08 - B04) / { (B04 + B08 + 0.725) x 1.725 } (16 in total)

- s1
  - VH
  - VV (18 if s1 is included)

For the s2-only competition we used the same excluding the s1 bands.

Each of these bands delivered two features, being the

- average and the
- standard deviation

of the pixel values for the band in the given date bucket.

We tried different date buckets and also to use more features, e.g. also use min and max or skewness and kurtosis. We also tried to merge and blend dates with cloud cover in various ways but in the end the improvement from these refinements were so marginal that we discarded them in favour of this simple approach.

Calculating the bands

The band calculations can be found in the `calc_tile` subroutine in `main4.py` on line 1025 and onwards. Note that we calculate and write out

- the number of observations
- min
- max
- sum and
- sum of squares

and later on, in the model files, use these to calculate the average and standard deviation of each band from the number of observations, the sum and the sum of squares. We tried to also use e.g. the min and max but found that these added little to the model.

Other features

For each field, we also added the following features

- number of pixels in the field
- size of the field
- fragmentation of the field
  - 1 - the size of the 'holes' in the field relative to its total size
  - Thus a value of 1 means no fragmentation

- perimeter length of the field
  - calculated using `skimage.measure.perimeter`

- perimeter length to field size
- circular measure
    - we fit a circle to the field perimeter and return a measure of the goodness-of-fit of the model
    - for this we use `skimage.measure.CircleModel`

Calculating other features

The calculation of these features can be seen in `main4.py` at line 333 onwards.

Refinements

The model was also refined in a number of ways, as discussed below.

Neighbourhood

We create two weighted distributions of the classes of the fields around each field. The weight of each class is inversely related to the distance between the field center and that of the neighbouring field but decay differently for each distribution.

We only use nearby fields for this, that is, fields whose center is within a given radius of the center of the field for which we calculate the neighbourhood distribution.

This estimates the distribution of fields around and nearby.

This eventually contributed little to the model.

When we calculate the distribution, we either use the known classes from the train set or we estimate using the preliminary model and use the out-of-fold model predictions to calculate the distribution.

If we use actual classes it contaminates the cross validation score a little bit.

This is why there are different types of models

- preliminary
    - does not use neighbourhood information
- final
    - use training set classes for neighbourhood information
- out-of-fold
    - use out-of-fold predicted classes for neighbourhood information
- dirty
    - use training set classes for neighbourhood information

Calculating neighbourhood information

The neighbourhood calculation can be seen in `main4.py` in the `update_field_neighbours` subroutine defined on line 1186 and onwards.

Overlap

Between satellite image patches one can detect overlap between fields. The class information in the overlap regions can be used in the model and this is another refinement we utilised.

This contributed a lot to the model accuracy and can be viewed as a proxy for the benefit one could get from on-the-ground information.

The overlap algorithm is complex given that a field may overlap multiple other fields in a neighbouring tile. It contributes the following added features

- weighted distribution of classes in the overlap region
- border region size to field size
- overlap size to field size
- combined pixel size of field
- combined size of field
- combined fragmentation
- combined perimeter
- combined perimeter to size
- combined circular measure

These measures are weighted relative to the size of the different fields. Thus a bigger field that overlaps with a smaller field will have its features weigh more in the combined feature set.

If we use overlap classes from the training set it contaminates the cross validation score and so we have different types of models, as explained before and as summarised below.

Calculating overlap information

The overlap calculation is complex. It is a two-step process as you have to find tile neighbours and then wiggle the tiles until the overlap fits optimally and then calculate the overlap for the fields themselves.

The code for it can be found in `main4.py` in the subroutines `update_overlap_tiles` on line 605 onwards and `update_overlap_fields` on line 698 and onwards.

Model type summary

The table below summarises how the neighbourhood and overlap information is constructed for the different models

Type	Neighbourhood	Overlap
Preliminary	No neighbourhood	No overlap
Final	Training set	Out-of-fold preliminary model predictions
Out-of-fold	Out-of-fold preliminary model predictions	Out-of-fold preliminary model predictions
Dirty	Training set	Training set

Each model will use the neighbourhood and overlap information together with class information as outlined in the table below to construct a weighted neighbourhood and weighted overlap class distribution. This happens in the model file itself, in the

`make_feature_matrix`

subroutine which is on line 501 in `model4-3.py` for the XL s1+s2 competition and on line 504 in `model4-3s2.py` used for the s2-only version of the competition.

Other

Clean and dirty and double

The *clean* and *dirty* here refers to the validation score you get from the train set for the various models. Clean models, such as *preliminary* and *out-of-fold*, give a *clean*, comparable validation score, while the *final* and *dirty* models will not give an accurate cross validation score as it re-uses training set information that strictly speaking should not be visible when calculating the cross validation score. Thus the training set cross validation score becomes contaminated or *dirty*.

The *preliminary* models do not use neighbourhood or overlap information and is clean.

The *final* models has a bit of contamination in the neighbourhood, but, since it is blended and weighted, the contamination is relatively small. This is sometimes referred to as the *clean* model.

The *out-of-fold* models are doubly clean in that all training set information that is out-of-fold is hidden when the model is constructed whereas the *dirty* models are contaminated on both the neighbourhood and the overlap features.

For this reason, the *out-of-fold* models are also referred to as *double* models in the code.

Because of this contamination, we did not rely on our local scores and simply used the models with the best public leaderboard scores as our selected models.

Cross validation table

To illustrate this difference, the table below gives our local scores as well as the public and private leaderboard scores for the various models.

XL s1+s2

Type	Local score	Submission file	Public LB score	Private LB score
preliminary	0.8471204160	b4-3pt	0.8343956804	0.840081383607631
final/clean	0.8027660691	b4-3ft	0.8183615395	0.822271875297808
out-of-fold/double	0.8066114458	b4-3ot	0.8156807195	0.819732318315509
dirty	0.5297249250	b4-3dt	0.6379515291	0.634814703680626

Here the impact of the contamination can be seen clearly.

s2 only

Type	Local score	Submission file	Public LB score	Private LB score
preliminary	0.8741469118	s2-b4-3pt	0.8569946922	0.865112409927843
final/clean	0.8260062900	s2-b4-3ft	0.8450565823	0.849939480471237
out-of-fold/double	0.8307605659	s2-b4-3ot	0.8428550396	0.847574185795938
dirty	0.5406239697	s2-b4-3dt	0.6518394015	0.649991155980249

Cross validation folds

We used 10 cross validation folds.

Size and pixel size

We use both measures for size and for pixel size.

Pixel size

The pixel size of a field is simply a count of the number of pixels inside it.

Size

Each tile has GPS coordinates that can be translated into a size (width and height) for each pixel. These differ slightly between tiles and, despite being a very similar measure as the pixel size, if the GPS area covered by the pixels are also calculated it adds to the accuracy of the model. Thus it seems the tile aspect ratio matters.

Models

For each model, we fit a lightgbm and a catboost model. Then we use a simple 0.05 step length or 20 step grid search and the training set to determine the best combination between lightgbm and catboost and use that ratio to create a *both* model with `b` prefix.

catboost

Catboost really served us well. It was terribly slow, but seemed to improve as we throw more iterations at it. In the end we settled for 10000 iterations! This, combined with 10 folds, makes it painfully slow but even after 10000 it seems the model's accuracy is still increasing. So maybe we should have used 20000 :-)

lightgbm

lightgbm also served us well and was much, much faster than catboost, making it ideal for testing.

Improving the model

We had lots of ideas on how to make further improvements.

Some improvements we planned but did not execute.

- Date buckets
  - We tried to get more information by using different date buckets
  - We even tried to calculate rolling moving averages across photos
  - In the end this helped little but maybe there is a way to improve this
- More and other derived bands
- The dirty validation score could be cleaned up
  - This will improve the accuracy of the local cross validation score
- More shape and size features
  - These seem to contribute but can be slow to calculate
- Using pixels covered by clouds
  - This can be tricky but can add more information

- Other ideas
  - :-)