Team Epoch IV
www.teamepoch.ai

Delft University of Technology
Batch 2

Kelp Wanted: Segmenting Kelp Forests

Help researchers estimate the extent of Giant Kelp Forests by segmenting Landsat imagery.

Epoch

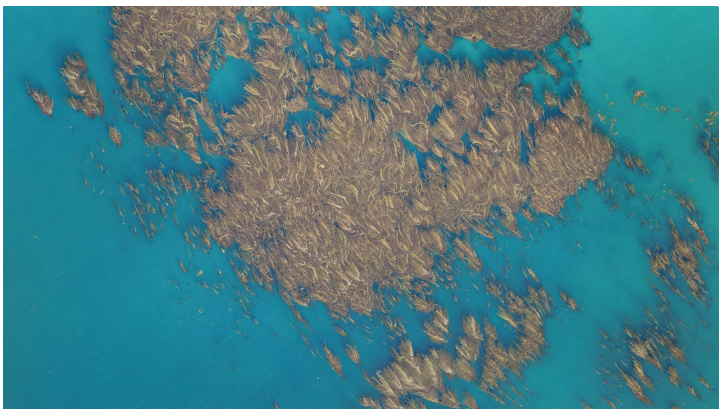
Engineering team Taqqadum:

Cahit Tolga Kopar
Emiel Witting
Hugo de Heer
Jasper van Selm
Jeffrey Lim

WOODS HOLE OCEANOGRAPHIC INSTITUTION

UMass Boston

ZOONIVERSE



February 22, 2024

1st Place Solution

Summary

This section contains a summary of our first-place solution and what we did to achieve this amazing result for us.

- Set all NaN values (-32,768) to 0.
- Feature engineering (NDVI, NDWI, ONIR, and many more varied features calculated on NDVI, such as edge detection, contrast enhancement).
- Slight Gaussian blur on targets with $\sigma = 0.5$.
- Fit XGBoost model on all features where it's predictions are used as a new feature to a DL model.
- Feature election selection step(Selection from all features and XGBoost outputs).
- Fit a StandardScaler.
- Models.
  - Architectures.
    - VGG-based UNet architectures (From Segmentation-Models-Pytorch)
    - Swin Transformer (SwinUNetR) based architecture (From Monai)
    - ConvNext architecture from Pytorch Image Models (timm)
  - Epochs: 75.
  - Batch size 16.
  - Layerwize learning rate decay (for ConvNext at 0.85[1]).
  - AdamW optimizer.
  - Cosine learning rate decay scheduler with 5 epoch warmup.
  - Weighted FocalDiceBoundary and Dice loss
- Augmentations
  - 90-degree rotations (Random choice between 0, 90, 180, 270 per image, $p = 1.0$).
  - Horizontal flip ($p = 0.5$)
  - Mosaic and Cutmix augmentations (20% chance of applying either to an image in a batch, no overlap).
- Test-time augmentations of all flip and n*90 degree combinations (8 combinations total).
- Custom ensemble selection with weighted averaging on raw model logits.
- Fitting threshold after weighted ensembling on the validation set maximizing dice score.

[1]The implementation was architecture-independent, so the counting of layers, and thus scale of the decay, might not be consistent.

What didn't work

- Weighted dice loss by multiplying by the amount of kelp within a batch.
- Removing ill-labeled images from the training dataset (5635 -> 5620).
- Stacked ensembling (using a UNet / DL architecture for ensembling that is trained on all model predictions)
- Retraining the segmentation head for more epochs after freezing all other layers.
- Prithvi model.
- TorchGeo models. (They worked, but limited us to architectures that were less performant)
- YOLO for segmentation.
- Any loss function that doesn't use DiceLoss (Such as BCE).
- Setting channel ranges that are outside valid kelp ranges[2] to 0.
- Clipping channel ranges to valid kelp ranges.
- Dithering as a postprocessing step.
- EfficientNet (overfitted very quickly).
- Applying other Test Time augmentations such as contrast / brightness.

What we haven't tried

This section contains some interesting approaches that we have not tried, but we might think they would lead to further improvements.

- Upscaling of satellite images.
- Using Deep learning techniques to fill in NaN data.
- Using classic techniques to fill in NaNs (using kNN imputation/interpolation).
- Trying different scalers (Normalization, MinMaxScaler etc..)
- Tuning the threshold for submit to match kelp distribution on training data.

Code

The code and documentation is available on GitHub. The configurations describing the ensembles and models completely can be found in the '/conf' folder. GitHub

Introduction

To start off, we would like to thank the organizers and DrivenData for organizing this competition. We as Epoch IV were hoping that we would be in the top 3 when the private scores were published and it came as a surprise seeing that we were 1st place. Not only that but seeing 9 of our submissions are actually winning submissions was even more unexpected.

[2]Valid kelp ranges: The range (min, max) of band values where there is a possibility to find kelp. For this, we looked at distribution plots on bands in which ranges of values kelp is encountered or not.

0.7187	0.7332	EWitting	id-251447 · 2w 2d ago · `3musketeers`
0.7200	0.7332	EWitting	id-251571 · 2w 1d ago · `7swin with 8 transformation self-ensembling`
0.7198	0.7331	tolgakopar	id-251721 · 1w 6d ago
0.7190	0.7330	EWitting	id-251450 · 2w 2d ago · `5powerrangers`
0.7195	0.7329	tolgakopar	id-251728 · 1w 6d ago · `7swin with double swin weight + 8 transforms + th…`
0.7196	0.7328	EWitting	id-251554 · 2w 1d ago · `7swin`
0.7237	0.7325	tolgakopar	id-252857 · 20h 38min ago · `21-02-1st`
0.7196	0.7324	EWitting	id-251569 · 2w 1d ago · `7man - 7dwarves minus rn50 plus manet`
0.7177	0.7319	EWitting	id-251378 · 2w 3d ago · `old 3-model ensemble with DL`

Figure 1.1: All our submission that get 1st place on private leaderboards

Features

To visualize our features and analyze how well our models perform, we have built a visualization dashboard with Dash in Python. During our EDA, we found out that the NIR channel has a very high importance for predicting kelp when looking at the kelp distribution ranges for each channel. From this channel, we created other new features such as NDVI which is used to quantify vegetation greenness and more. Also, from this NDVI we calculated different new features using filters such as contrast enhancement, edge detection using Sobel, sharpening, and a modal filter. These can be observed in the figure 1.2 below.

Furthermore, we fitted XGBoost on all of these features and used the predictions of the XGBoost model as a new feature for the deep learning model later on. Then we applied feature selection to select the most meaningful features and applied a standard scaler.

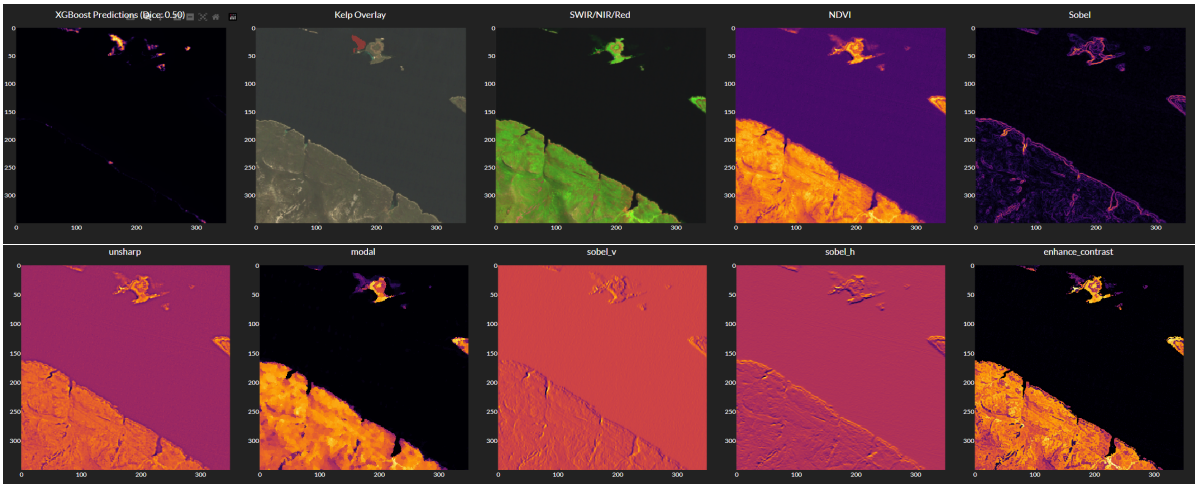


Figure 1.2: The features that we have used for our models. (The kelp overlay uses an RGB image)

Loss Functions

For the choice of the loss function, we initially went with Dice loss since it is the metric. After some research, we saw that other loss functions like Jaccard loss, BCE, and Focal loss were being used for segmentation. After doing many test runs with a Weights and Biases sweep for loss we found that dice loss was the best.

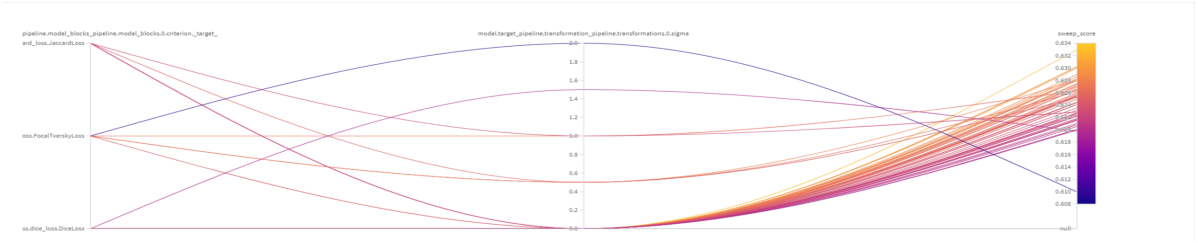


Figure 1.3: Comparison of different loss functions and the CV scores they get for a subset of 1000 images. It shows that Dice loss consistently results in the highest scores

During the last week, however, we came across a new loss function called focal dice boundary loss that was used in the winning solution of a Kaggle competition for image segmentation. When training models using dice loss with an 80%-20% split for train and validation sets our models would early stop around 60-65 epochs. What we noticed is that the validation loss with this new loss function was more stable in comparison to dice loss and it would not early stop in 75 or more epochs. This meant we could train our models for longer and this also resulted in better validation scores (The scheduler for these models were also changed to decay to 0 in 100 epochs instead of 75).

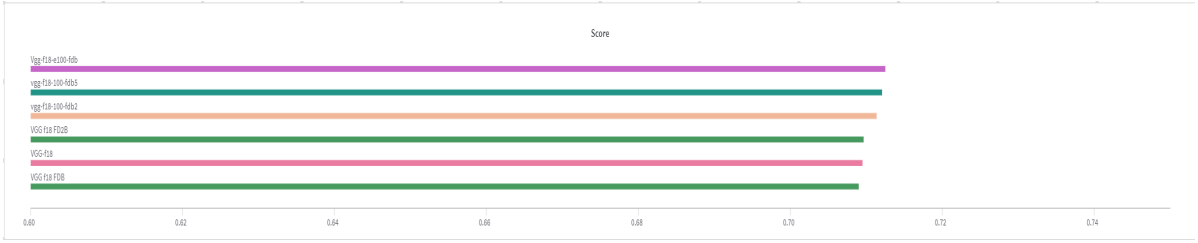


Figure 1.4: Comparison of vgg model scores for dice loss vs focal dice boundary loss

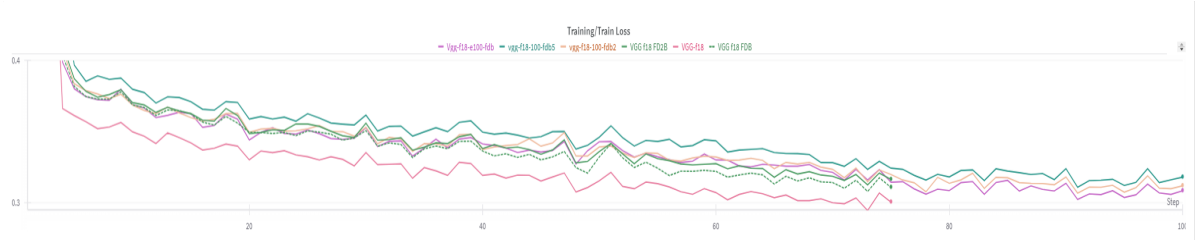


Figure 1.5: Comparison of vgg model training losses for dice loss vs focal dice boundary loss

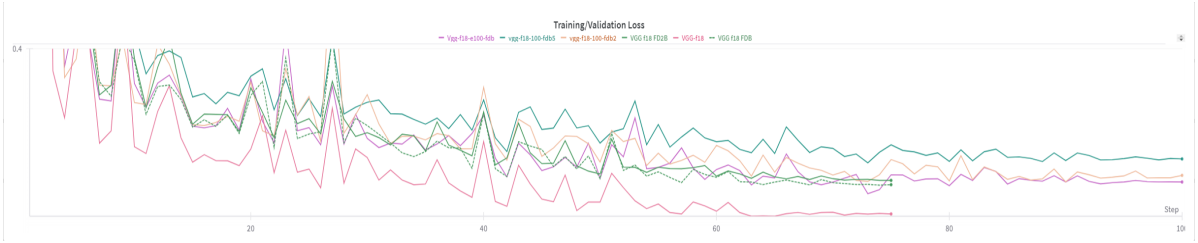


Figure 1.6: Comparison of vgg model validation losses for dice loss vs focal dice boundary loss

Threshold

Since the expected output is a binary mask the model outputs have to be converted from probabilities to binary with a threshold. We have used threshold optimization which leads to an increase in score for

individual models. However, when models are ensembled this noise in the predictions is filtered out (this is also the case for a single model with TTA), and fitting a threshold results in values in the range 0.51-0.49. We simply chose to use 0.5 for all our ensemble submissions.

TTA (Test-time Augmentations)

Since satellite images can be in any orientation and CNNs are position-dependent, ie. if the image is flipped vertically, a prediction is made and flipped back it will give a different result compared to making a prediction directly. Using all the unique combinations of flipping and rotating an image in multiples of 90 degrees and averaging the results of the 8 combinations per image gives a slight improvement in local scores. Our public LB scores for one of our ensembles also went from 0.7196 to 0.7200. After seeing this TTA was included in the rest of the submissions.

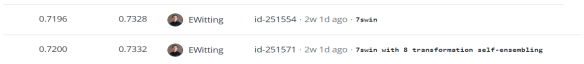


Figure 1.7: The improvement on the leaderboards scores of using TTA

HPO

For doing hyper-parameter optimization we used Weights and Biases sweeps. From the sweeps, we came to the following conclusions:

- Batch size of 16 works the best
- Vgg encoders give high scores more consistently
- Unet(plusplus) decoder works the best
- Learning rates in the scale of 1e-3 work the best

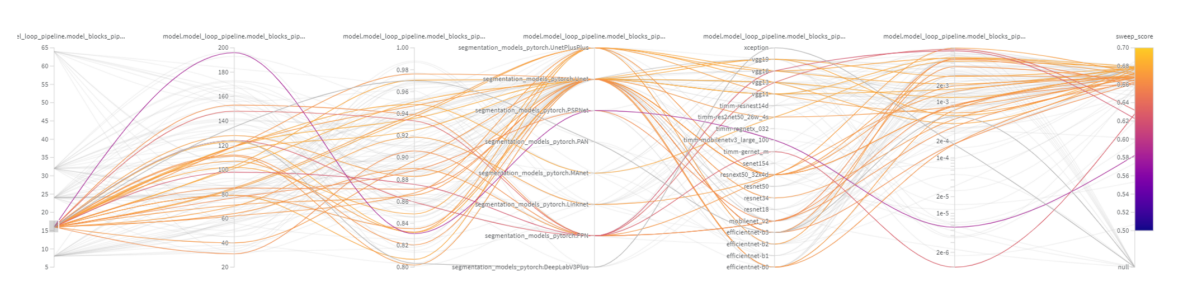


Figure 1.8: Results from the Weights and Biases sweep where runs with batch size of 16 are show. Note that these runs do not use all the features shown in figure 1.2

Ensembling

Combining models

For ensembling, we have tried multiple approaches such as majority voting, stacking ensemble and weighted average. From these methods, the weighted average of the predictions gave us the best overall scores both locally and on public LB.

Selecting models

For an ideal ensemble we combine models that are not only accurate by themselves, but make errors in different ways, so they can complement each other. Initially, we chose our ensembles by taking models with high 5-fold CV scores, and judging the variety by looking at their configuration. Variety was mostly in feature selection and encoder architecture. Later, we experimented with automated ensemble selection methods.

We used three algorithms to optimize the model selection and weights, based on model predictions on a validation set and labels.

- Genetic algorithm for optimizing Dice score from model weights. As a regularization method to prevent overfitting, the amount of possible values for the weights could be reduced. We ended up allowing the values 0,1, and 2 (before normalizing the weights across the ensemble to sum to one).
- Linear regression to optimised MSE. This works, but will result in every model being used, which not preferred for performance reasons. It can also produce negative weights, which we found to usually not be reliable when transferring to a different dataset.
- Combinatorial optimisation of MSE. Using optimizations we could very quickly evaluate the MSE for an unweighted binary selection of models. This allowed a brute-force search of all 2^N combinations.

We also generated visualizations, to make more informed manual selections. It was possible to directly show the concept that an ensemble should contain models that approach the target from different directions. This was done by comparing the correlation of the errors (prediction minus target) between models. This roughly corresponds to the angle between models. Then, we could project this correlation using PCA and show the most significant components. Figure 1.9 shows this plot for 3D. It becomes clear that similar models are clustered together, and that the automated selection method chooses the highest scoring model from each cluster, to achieve a well-rounded ensemble.

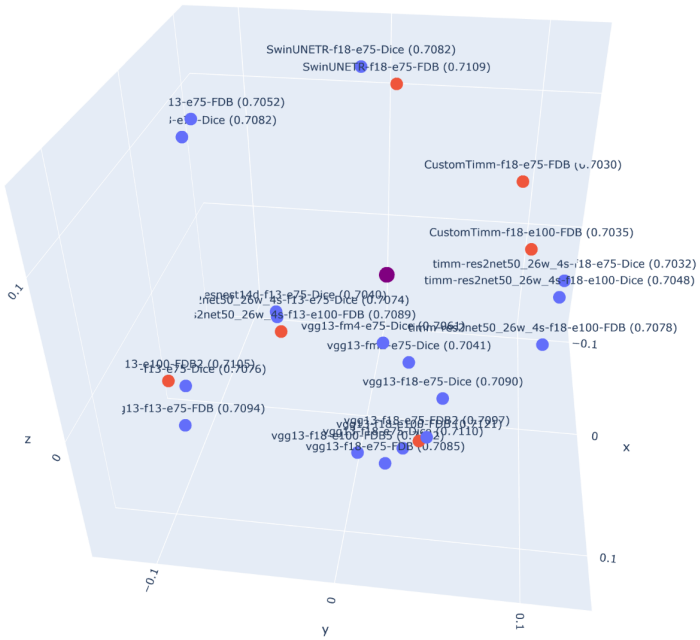


Figure 1.9: PCA of model errors, labeled by configuration and individual dice score. The red markers are models chosen by binary ensemble optimization, and the purple point (near the origin, so close to zero error), is the error of the ensemble.

To further aid in understanding model variety, we used hierarchical clustering to create a dendrogram using the model error correlation. This successfully clustered similar models, and made it clear which changes (architecture, features, etc.) make the biggest difference in model variety. See figure 1.11

These techniques were promising already, but would probably benefit a lot more from optimizing cross-validated predictions. They were applied to only a 20% validation set, and the methods might be prone to overfitting.

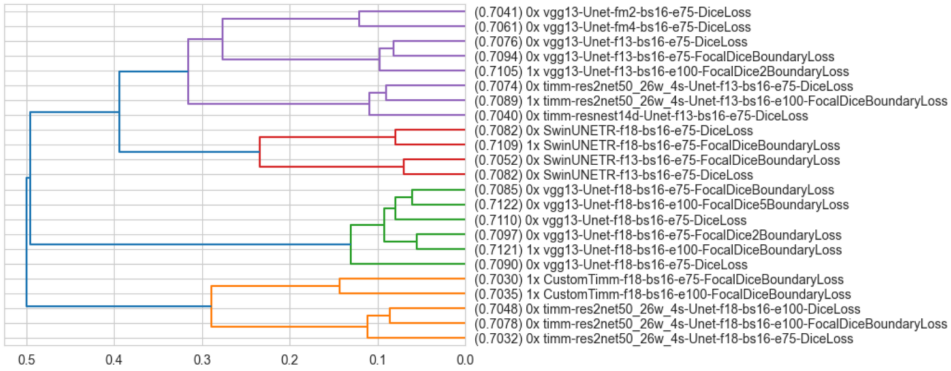


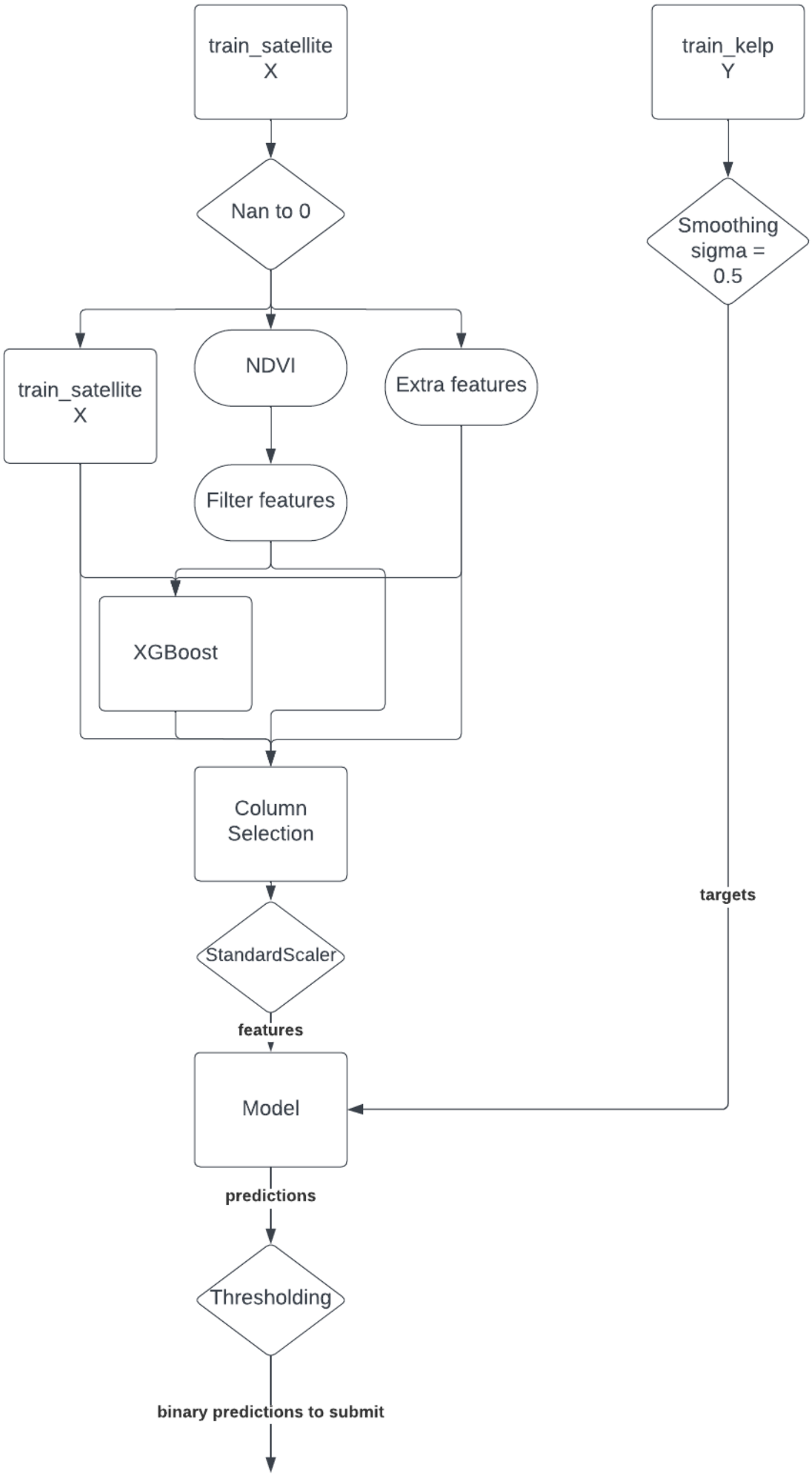
Figure 1.10: Dendrogram based on hierachical clustering using error correlation. The "1x" and "0x" signify the model weights as chosen by the ensemble optimisation process.

Hardware

The hardware that was available to us during the competition was 2 workstations with 2 A5000 GPUs and 1 workstation with 2 A6000 GPUs. All with 128GB of RAM. However, we also had access to a DGX cluster (8 H100 GPUs) for a week during the second half of the competition. This allowed us to run 5-fold CV sweeps without subsampling the dataset as shown in figure 1.8.

Pipeline

The figure below shows our overall pipeline visualized in a diagram.


train_satellite
X
train_kelp
Y
Nan to 0
Smoothing
sigma =
0.5
train_satellite
X
NDVI
Extra features
Filter features
XGBoost
Column
Selection
targets
StandardScaler
features
Model
predictions
Thresholding
binary predictions to submit