Overview

We started by getting multiple extents and tracking the details in a spreadsheet (see Extents.xlsx). We performed feature extraction followed by subsampling by class to train the model on balanced data. We each experimented with different feature extraction techniques and model priorities after identifying the most effective extents on their own. The different techniques that we each prioritized are outlined below, and in each of our summaries we will go into detail more about the outlined methods. There is overlap as well since we built off each other's findings.

Leah

- Pipeline for comparison
    - Different models
    - Different extents
    - Gridsearch/Random search
    - Feature importance
- Calculated layers
    - Base 4
    - Additional [10]
- Normalization
    - Log
    - Z score

Fynn

- Demo/Extent testing
    - Prediction maps
- Feature selection from image
    - Filtering
    - K means clustering
    - Edge detection
    - Geocoordinates
- Overlapping multiple model predictions
    - Binary model

Summaries

Leah

During this project, I did most of the base testing to see how different features and models impacted our accuracy.

I created a basic pipeline using sklearn (not later used) and functions (later used) to test how the accuracy changed for various models. On a sample size of 25000 per band using Simcoe Region, Labrador and James Bay data, I tested 5 types of models: SVC, K Nearest Neighbors, Random Forest, XGB Classifier and Gradient Boosting Classifier. Random Forest tested with the highest accuracy at 61.6% and SVC was the lowest at 39%.

Afterwards, various pipelines were done to compare different methods. Three scalers were tested to see if it improved the model in anyway. Pipelines with PCA or a feature union with PCA and Kbest were also done, however none were found to improve model accuracy. In addition, with the Random Forest pipeline, 3 types of normalization were applied to the data: Log, ZScore and outlier treatment. All three normalization tactics resulted in a drastic decrease in accuracy (ie. < 1%).

Continuing with the random forest, I created other layers with the bands. These layers and equations are listed below:

Layer	**Equation**
NDVI column	(B08-B04) / (B08+B04)
Moisture index	(B8A-B11) / (B8A+B11)
NDWI	(B03-B08) / (B03+B08)
NDSI	(B03-B11) / (B03+B11)
Normalized NIR/Blue normalized vegetation index	(df.B08 - df.B02) / (df.B08 + df.B02)
Normalized green difference vegetation index	(df.B08 - df.B03) / (df.B08 + df.B03)
Atmospheric Resistant Green	(df.B03 - df.B04) / (df.B03 + df.B04)
Green-Blue NDVI	(df.B08 - (df.B03 + df.B02)) / (df.B08 + (df.B03 + df.B02))
Red-Blue NDVI	(df.B08 - (df.B04 + df.B02)) / (df.B08 + (df.B04 + df.B02))
Green-Red NDVI	(df.B08 - (df.B03 + df.B04)) / (df.B08 + (df.B03 + df.B04))
GARI	(df.B08 - (df.B03 - (df.B02 - df.B04))) / (df.B08 - (df.B03 + (df.B02 - df.B04)))

Yellow Vegetation Index	(0.723 * df.B03) - (0.597 * df.B04) + (0.206 * df.B06) - (0.278 * df.B09)
Mid-infrared vegetation index	(df.B09 / df.B11)
GDVI	(df.B08 - df.B03)

The addition of NDVI, Moisture Index, NDWI and NDSI to the model and tested against the test data, increased the accuracy of the model by 12%. As such, ten additional layers were tested to view their feature importance. However, these additional layers did not make further improvements in the model's accuracy.

Lastly, I ran grid searches on the random forest model for optimization. A Randomized Grid Search resulted in the best parameters as: max depth = 8, min sample per leaf = 2, number of estimators = 200). However, the randomized grid search did not optimize the model as the model accuracy dropped 10%. Therefore, a regular grid search was conducted. Regular grid searches were found to continually run and no results were found. Grid searches were conducted multiple times with fewer hyperparameters and smaller sample sizes to improve the time of the grid search, but this still resulted in models running for 14+ hours.

If more time was permitted, I would have liked to try a technique to fix the angle of the bands as images showed there was a slight angle. This would have taken some time to play around with as the reference found on Github would not work on L2A products, which we were using in this case.

Band Reference for Equations
https://custom-scripts.sentinel-hub.com/custom-scripts/sentinel-2/indexdb/

Github Reference for Band Angles
https://github.com/fronzag/sentinel2_angle_bands

Fynn

I used Random forest classifier from sklearn almost exclusively (there's one exception I'll talk about later). The hyperparameters that got the best result where train and test accuracy overlapped (no overfit) was 300 trees, max depth 15 and max features 2. I started with around 20 thousand values from each of 19 classes then slowly moved up. The final models were trained on upwards of 100 thousand values for each class. Full details on which versions used what hyperparameters and how many values per class can be found in my github land cover repo in the evaluation folder in the file 'evaluation_legend.md'. Also, other than the first few versions I integrated Leah's work on the calculating layers and added at least the base 4 as features.

Important to note is that subsampling was random when pulling from different extents. Ideally when we subsampled it would pull as equal as possible from each extent, however we weren't able to make this work. Instead the subsample is random, so each input extent is contributing a random amount to the training data for each class. A good next step for this project would be to create a way to customize how much each extent is contributing or at least track what the random subsampling is doing.

There are two evaluations that I used: test accuracy and demo site accuracy. Test accuracy is derived from the prediction using the test data from sklearn's test, train split. This means it is the random pixels of many different extents, and all extents the model was trained on. It also is balanced since the split occurs after subsampling. In every single iteration with at least 100K values to train on the test accuracy remained between 60% and 65% for any feature or extent combination. Conversely, the demo site accuracy is the extent provided by Fraser. This means it's unbalanced, completely unknown, and consists of all pixels from one extent. This makes it a far better metric for how the model is performing than the test accuracy and from here on out it is mostly what I will use. With the higher performing versions I also tested them against other random extents. Something interesting was that regardless of whether the extent was one that had been subsampled for training, its accuracy would mirror the demo site accuracy for that model give or take 5%. Full details on evaluation outcomes for over 15 versions can also be found in 'evaluation_legend.md'

The first feature I experimented with was filtering. The idea was to blur values together making any value more similar to its neighbors. Since some of the patterns we are seeing form spatial 'clumps', the hope was to help an unstructured model identify a pixel as being more likely to be the same as the pixel next to it. I started with gaussian filtering since it was defined as "a weighted mean of the surrounding pixels that gives more weight to the pixel near the current pixel" (https://www.codingame.com/playgrounds/2524/basic-image-manipulation/filtering). With just the gaussian transformation model accuracy got significantly worse, however with both gaussian and base raws the model began to 'clump' values more intensely (see map version 6). For most classes this created too much intensity and worsened accuracy but its performance for cropland (an intensely clumped class) is better than any model that trained without the gaussian transformed bands (see map version 11). A sub-note of that is different possible sigma values for the gaussian filtering (ie: different possible intensities). I tried 1, 3 and 5, and 5 gave the best result. The other filtering attempt was median filtering, however it was worse (see map version 7) than the gaussian filtering so eventually I just stuck to gaussian sigma 5 when using filtering as a feature.

The next feature was clustering. Applying k means clustering to the data to create a 'cluster' column with labels. This was another attempt to increase the model's ability to clump together similar values using unsupervised learning. While I was not able to use it in a way that increased model accuracy, I still think it's an intriguing idea with a lot of avenues. It was especially interesting to see the clusters it would identify in the demo site after having been trained on many other extents. Overall, 4 clusters seemed to give the model something (see

map version 12) but didn't do as much in identifying the spatial patterns as gaussian filters did so eventually I focused my efforts elsewhere.

Edge detection was part of an effort to increase Urban accuracy. Most of the time the urban class has a spatial pattern of many lines (roads) that are extremely fine. My hope was that canny edge detection would help identify these roads and give the model a column of True/False values as to whether a pixel was edge/not edge (roads always being edges). While it was not effective helping the Urban class like I had hoped (the roads are too fine), it did help overall with accuracy and balanced accuracy for subsequent model runs. It especially helps segment off water bodies. It's worth noting that depending on what band you run the edge detection on with what level of intensity (sigma value), the results of the edge detection vary. The best results I got were from 'B8A' sigma = 3, although 'B03' sigma = 1 also picked up similar edges it tended to create more noise.

The final feature I tried was geocoordinates. Since the chance of a certain land cover appearing has a great deal to do with where it is geographically this was something I wanted to incorporate since day 1. It didn't help as much as I thought, actually making the demo accuracy 2% worse, but definitely plays a large part in making the model more robust. With the incorporation of geocoordinates there is significantly less prediction of classes that aren't in the demo extent, and I saw large increase in balanced accuracy and accuracy (5-10%) in various other random extents I tried.

The last large step I took was to overlap model predictions. The main motivation was to overlap different versions, specifically a version that had been trained with gaussian transformations and a version that had not. The most sophisticated version of this would be to combine the prediction probabilities using a weighted average of some kind. Unfortunately I wasn't able to get there, it just seemed like too much work coming into the final weeks. Instead, I made a function that takes a base model then updates it with the selected class from the predictions of a different model. It can take more than two models also, however more models the longer it takes since it has to generate entire predictions for each. Version 10 updated with class 15 (cropland) from version 13 performed better on various random extents - including the demo extent - than either model individually with a balanced accuracy of 43.27.

The next piece was to make binary models targeting specific classes. Since the multiclass model seems completely unable to pick up the Wetland pattern in the demo extent on its own (see all map versions). I created a binary model from one vs. rest for Wetland (class 14) and combined it into the model predictions as well, using it to update the wetland class. I tried both random forest and xgboost to make this binary model. Using the binary model in combination did improve wetland somewhat but overall dropped balanced accuracy. I suspect if we had a better binary model it would help more, but both binary models still struggled to pick up on the true wetland in the scene.