ML_Final Project

December 9, 2021

[ ]:
```
## Install dependencies as needed
pip install pandas
pip install plotly
pip install pyproj
```

1 ML 497 Final Project

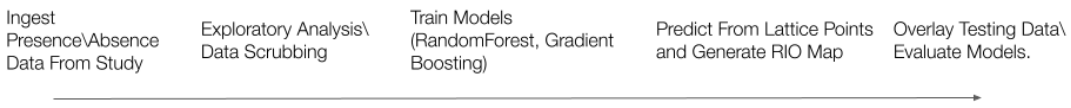
1.0.1 Modeling Icelandic Ptarmagin Population

1.0.2 Abstract

The goal of this project is to use a machine learning model to predict the presence of the icelanding rock ptarmagin. To do so GIS data will be used in conjungtion with several machine learning models to generate a relative index of occourance map. This map will tell us where we can expect the Ptarmagin habitat to be, which will in turn allow us better insights on how might preserve the habitat and animal species.

1.0.3 Methods

For this model we will be generating an ensemble model, not only do they generally perform better than single models(Delgado Paper) but it will also provide ample opportunity to showcase what we have learned throughout the semester. Our model will be constructed by averaging the RIO maps of Random Forest and Gradient Boosting. This model will also provide an interesting benchmark into how these different models perform with ecological data. The following graphic provides general workflow details for our project.



1.0.4 Ingesting Data

[41]:
```
##Data was obtained in csv format. We will be ingesting it in the form of a␣
 ↪pandas dataframe.
import pandas as pd
import numpy as np

Train = pd.read_csv('ptarmigangbif_trainFH2OVRLcPRESBACK.csv')
Train = Train.drop(['id', 'FID'],axis=1)
```

```
Presence = pd.read_csv('ptarmigan_transect_test.csv')
Lattice = pd.read_csv('regpoints1000FH2seloOVRLbXYc.csv')
Lattice = Lattice.drop(['id', 'FID', 'veg250prj'],axis=1)
```

[42]:
```
Train.head()
```

[42]:

	NDVImax250	JJA_tavg25	JJA_ppt_av	JJA_mean_w	distance_w	dist_pastu	\
0	NaN	NaN	NaN	NaN	NaN	NaN	
1	5955.970215	7.80000	47.666699	4.66667	250.0	4802.339844	
2	NaN	NaN	NaN	NaN	NaN	NaN	
3	6054.459961	7.89794	47.293701	4.73730	0.0	4609.770020	
4	6111.259766	7.90000	47.000000	4.70000	250.0	4924.430176	

	dem250	remoteness	soil250	veg250	slope250	PresBack
0	NaN	NaN	NaN	NaN	NaN	0
1	1.0	0.0	2.0	4.0	0.00000	0
2	NaN	NaN	NaN	NaN	NaN	0
3	8.0	0.0	2.0	4.0	2.91688	0
4	10.0	0.0	2.0	4.0	2.47191	0

[43]:
```
Presence.head()
```

[43]:

	RV	x	y
0	presence	2594052.0	326648.999999
1	presence	2607413.0	335016.999999
2	presence	2780171.0	431362.999999
3	presence	2780171.0	431362.999999
4	presence	2773427.0	410624.999999

[44]:
```
Lattice.head()
```

[44]:

	NDVImax250	JJA_tavg25	JJA_ppt_av	JJA_mean_w	distance_w	dist_pastu	\
0	NaN	NaN	NaN	NaN	NaN	NaN	
1	5955.970215	7.80000	47.666699	4.66667	250.0	4802.339844	
2	NaN	NaN	NaN	NaN	NaN	NaN	
3	6054.459961	7.89794	47.293701	4.73730	0.0	4609.770020	
4	6111.259766	7.90000	47.000000	4.70000	250.0	4924.430176	

	dem250	remoteness	soil250	veg250	slope250	POINT_X	POINT_Y
0	NaN	NaN	NaN	NaN	NaN	2744254.077	474008.5897
1	1.0	0.0	2.0	4.0	0.00000	2832254.077	474008.5897
2	NaN	NaN	NaN	NaN	NaN	2744254.077	473008.5897
3	8.0	0.0	2.0	4.0	2.91688	2825254.077	473008.5897
4	10.0	0.0	2.0	4.0	2.47191	2826254.077	473008.5897

1.0.5 Exporatory Analysis Data Scrubbing

Here we can see that the training set had a totoal of 102,733 data points, with several of them containing missing data just by looking at how the NDVImax250 column has several thousand less entries. In general the decision tree methods we will use to model the data work just fine with missing data, by either imputing a mean or by weighing in the cost function. MaxEnt also by default removes all rows with missing values.

[45]:
```
Train.describe()
```

[45]:

	NDVImax250	JJA_tavg25	JJA_ppt_av	JJA_mean_w	\
count	94892.000000	104337.000000	104337.000000	104337.000000	
mean	4563.137758	7.315812	78.414729	5.546364	
std	2690.554207	2.211253	26.971311	1.429474	
min	-1963.520020	-1.366670	35.727200	2.347660	
25%	2098.502503	6.594300	56.162498	4.654430	
50%	5345.149902	7.720850	72.681602	5.208600	
75%	6868.447632	8.732580	94.003899	5.933330	
max	9172.780273	11.616300	163.636002	15.423200	

	distance_w	dist_pastu	dem250	remoteness	\
count	105041.000000	105041.000000	105041.000000	105040.000000	
mean	2762.631829	15457.899025	497.443143	0.330541	
std	3842.588472	17887.872549	372.611507	0.470410	
min	0.000000	0.000000	-1.000000	0.000000	
25%	559.017029	2500.000000	189.160004	0.000000	
50%	1500.000000	7766.109863	464.553009	0.000000	
75%	3288.239990	22360.699220	700.000000	1.000000	
max	31250.000000	79649.203130	2023.380005	1.000000	

	soil250	veg250	slope250	PresBack
count	104905.000000	105041.000000	105041.000000	105780.000000
mean	6.766188	7.968308	6.750456	0.028805
std	4.699340	3.784224	8.974685	0.167259
min	1.000000	1.000000	0.000000	0.000000
25%	2.000000	4.000000	0.385425	0.000000
50%	6.000000	10.000000	3.153710	0.000000
75%	11.000000	11.000000	8.922500	0.000000
max	16.000000	13.000000	71.227997	1.000000

[46]:
```
Train = Train.dropna()
Train.head()
```

[46]:

	NDVImax250	JJA_tavg25	JJA_ppt_av	JJA_mean_w	distance_w	dist_pastu	\
1	5955.970215	7.80000	47.666699	4.66667	250.0	4802.339844	
3	6054.459961	7.89794	47.293701	4.73730	0.0	4609.770020	
4	6111.259766	7.90000	47.000000	4.70000	250.0	4924.430176	
5	5942.950195	7.90000	47.000000	4.70000	0.0	5408.330078	

```
6  6099.029785     7.90000   47.648201     4.70000       250.0  6020.799805

   dem250  remoteness  soil250  veg250  slope250  PresBack
1     1.0         0.0      2.0     4.0   0.00000         0
3     8.0         0.0      2.0     4.0   2.91688         0
4    10.0         0.0      2.0     4.0   2.47191         0
5     6.0         0.0      2.0     4.0   0.00000         0
6     3.0         0.0      2.0     4.0   0.00000         0
```

Here we can see that our test data has no missing values. Always good to double check!

[47]:
```
Presence.isna().sum()
```

[47]:
```
RV    0
x     0
y     0
dtype: int64
```

Looking at the prediction data for our frame, we can see that it has some missing data. This will show up in our RIO plots, however it is not a big deal. We could simply discard those plots, and simply mark them in our RIO plot. We could also simply impute the mean of the values near the missing values. We could also run them through only the Random Forrest and Gradiant Boosting models which are robust to missing data and exclude the MaxEnt predictions.

[48]:
```
Lattice.describe()
Lattice = Lattice.dropna()
Lattice
```

[48]:
```
          NDVImax250  JJA_tavg25  JJA_ppt_av  JJA_mean_w   distance_w  \
1       5955.970215     7.80000   47.666699     4.66667   250.000000
3       6054.459961     7.89794   47.293701     4.73730     0.000000
4       6111.259766     7.90000   47.000000     4.70000   250.000000
5       5942.950195     7.90000   47.000000     4.70000     0.000000
6       6099.029785     7.90000   47.648201     4.70000   250.000000
…               …           …           …           …           …
102727   408.195007    10.83330  126.084000     5.40000   500.000000
102728   616.833984    10.83330  127.661003     5.40000   500.000000
102729   330.421997    10.91390  127.055000     5.40000  1346.290039
102730   436.752991    10.96060  125.667000     5.37273   500.000000
102731   399.510986    11.03330  125.072998     5.30000   500.000000

          dist_pastu  dem250  remoteness  soil250  veg250  slope250  \
1       4802.339844     1.0         0.0      2.0     4.0  0.000000
3       4609.770020     8.0         0.0      2.0     4.0  2.916880
4       4924.430176    10.0         0.0      2.0     4.0  2.471910
5       5408.330078     6.0         0.0      2.0     4.0  0.000000
6       6020.799805     3.0         0.0      2.0     4.0  0.000000
…               …       …           …        …       …        …
```

```
102727  5857.689941    10.0         0.0     14.0    11.0  0.005607
102728  6656.759766     7.0         0.0     14.0    11.0  2.743580
102729  7504.169922     5.0         0.0     14.0    11.0  0.000000
102730  8385.250000     4.0         1.0     14.0    11.0  0.000000
102731  9290.450195     2.0         1.0     14.0    11.0  0.000000

            POINT_X      POINT_Y
1       2832254.077  474008.5897
3       2825254.077  473008.5897
4       2826254.077  473008.5897
5       2827254.077  473008.5897
6       2828254.077  473008.5897
…               …            …
102727  2713254.077  122008.5897
102728  2714254.077  122008.5897
102729  2715254.077  122008.5897
102730  2716254.077  122008.5897
102731  2717254.077  122008.5897

[91211 rows x 13 columns]
```

1.0.6 Converting Coordinates to Latitude and Longitude

[49]:
```
## Import PyProj library to convert to LAT and LON
from pyproj import Transformer
transformer = Transformer.from_crs('epsg:8088', 'epsg:4326')
PresenceLat, PresenceLon = transformer.transform(Presence['x'],Presence['y'])
PredictLat, PredictLon = transformer.
 ↪transform(Lattice['POINT_X'],Lattice['POINT_Y'])
```

[50]:
```
## Save Converted Coordinates in Respective Dataframes
Presence['Latitude'] = PresenceLat
Presence['Longitude'] = PresenceLon
Lattice['Latitude'] = PredictLat
Lattice['Longitude'] = PredictLon
```

1.1 Unused Test Evaluation

1.1.1 Generating Closest Test Prediction

We want to be able to evaluate the performance of our model with the testing data. To do so we need to generate the input data for each of our entry data, so for each point in our testing data we will find the closest corresponding point in our prediction lattice. Once we genratete our predictions from the model we will compare them to our testing data.

[13]:
```
PresenceInputDataIndex = []
for i in range(len(Presence)):
    xPres = Presence['x'][i]
```

```
    yPres = Presence['y'][i]
    xPredict = Lattice['POINT_X']
    yPredict = Lattice['POINT_Y']
    distance = (((xPredict - xPres)**2) + ((yPredict - yPres)**2)) ** .5
    PresenceInputDataIndex.append(distance[distance == min(distance)].index[0])
```

[114]:
```
TestData = Lattice.drop(['POINT_X','POINT_Y', 'Latitude', 'Longitude'], axis=1).
↪loc[PresenceInputDataIndex]
TestData['Latitude'] = Presence['Latitude'].tolist()
TestData['Longitude'] = Presence['Longitude'].tolist()
TestData['PresBack'] = 1
```

1.2 Training Models

Random Forest The RandomForest model was created in SPM using the random forest engine. Mostly default settings were used, with the number of trees set to 500 instead of 200. The model achived a ROC score of 0.97302 on out of bag samples. Under a balanced threshold the model achived an accuracy of 91.88% on OOB samples. Variable Importance showed that DEM250, JJA_TAVG25, JJA_PPT_AV, NDVIMAX250, and DIST_PASTU were the top five predictors.

TreeNet (Gradient Boost) The Gradiant Boost model was created in SPM using the TreeNet engine. Again mostly default setting were used, with the number of trees set to 500, the target class weights were set to balanced, and testing was done with v-fold cross validation. The model achieved a ROC score of 0.97233 on the v-fold data. Under a balanced threshold the model achived an accuracy of 91.75% on the v-fold data. Variable importance showed that DEM250, JJA_PPT_AV, JJA_TAVG25, SOIL250, and JJA_MEAN_W were the top five predictors

1.2.1 Exporting Training and Prediction Data for SPM Models

[27]:
```
Train.to_csv('Train2SPM.csv', index = False)
Lattice.to_csv('Lattice2SPM.csv', index = False)
TestData.to_csv('TestData2SPM.csv', index = False)
```

1.2.2 Importing Lattice Predictions

[76]:
```
## Score CSV was exported from SPM.
LatticePredictionGBM = pd.read_csv('LatticeGBM_score.csv')
LatticePredictionRFM = pd.read_csv('LatticeRFM_score.csv')

## Adding Lattice Coordinated to SPM prediction.
LatticePredictionGBM['Longitude'] = Lattice['Longitude'].tolist()
LatticePredictionGBM['Latitude'] = Lattice['Latitude'].tolist()
LatticePredictionRFM['Longitude'] = Lattice['Longitude'].tolist()
LatticePredictionRFM['Latitude'] = Lattice['Latitude'].tolist()

## Reformat Dataframe for Plotting
```

```
LatticePredictionGBM = LatticePredictionGBM.loc[:,['PREDICTION','PROB_1',␣
↪'Latitude', 'Longitude']]
LatticePredictionRFM = LatticePredictionRFM.loc[:,['RESPONSE','PROB_1',␣
↪'Latitude', 'Longitude']]

## Generate Dataframe for Ensemble model
LatticePredictionEnsRIO = pd.DataFrame()
LatticePredictionEnsRIO['PROB_1'] = (LatticePredictionGBM['PROB_1'] +␣
↪LatticePredictionRFM['PROB_1'])/2
LatticePredictionEnsRIO['Longitude'] = Lattice['Longitude'].tolist()
LatticePredictionEnsRIO['Latitude'] = Lattice['Latitude'].tolist()
```

1.2.3 Plotting Data

[61]:
```
import plotly.graph_objects as go
import os

if not os.path.exists("images"):
    os.mkdir("images")
```

[95]:
```
fig = go.Figure(data=go.Scattergeo(
        lat = Presence['Latitude'],
        lon = Presence['Longitude'],
        marker = dict(
            size = 3.5,
            ),
        )
)

fig.update_layout(
    width=3000,
    height=1500,
    margin = dict(
            autoexpand=True,
            l=0, #left margin
            r=0, #right margin
            b=0, #bottom margin
            t=50  #top margin
          ),

    geo = dict(
            resolution = 50,
            lonaxis_range= [-31,-6],
            lataxis_range= [63, 67],
        )
    )
fig.update_geos(projection_type="mercator")
```

```
fig.write_image("images/fig.png", width=3000, height=1500)
```

[92]:
```
figRFMrio = go.Figure(data=go.Scattergeo(
        lat = LatticePredictionRFM['Latitude'],
        lon = LatticePredictionRFM['Longitude'],
        marker_color = LatticePredictionRFM['PROB_1'],
        marker = dict(
            size = 3.5,
        ),
    )
)

figRFMrio.update_layout(
    width=3000,
    height=1500,
    margin = dict(
            autoexpand=True,
            l=0, #left margin
            r=0, #right margin
            b=0, #bottom margin
            t=50  #top margin
          ),

    geo = dict(
            resolution = 50,
            lonaxis_range= [-31,-6],
            lataxis_range= [63, 67],
        )
    )
figRFMrio.update_geos(projection_type="mercator")
figRFMrio.write_image("images/figRFMrio.png", width=3000, height=1500)
```

[93]:
```
figGBMrio = go.Figure(data=go.Scattergeo(
        lat = LatticePredictionGBM['Latitude'],
        lon = LatticePredictionGBM['Longitude'],
        marker_color = LatticePredictionGBM['PROB_1'],
        marker = dict(
            size = 3.5,
        ),
    )
)

figGBMrio.update_layout(
    width=3000,
    height=1500,
    margin = dict(
            autoexpand=True,
```

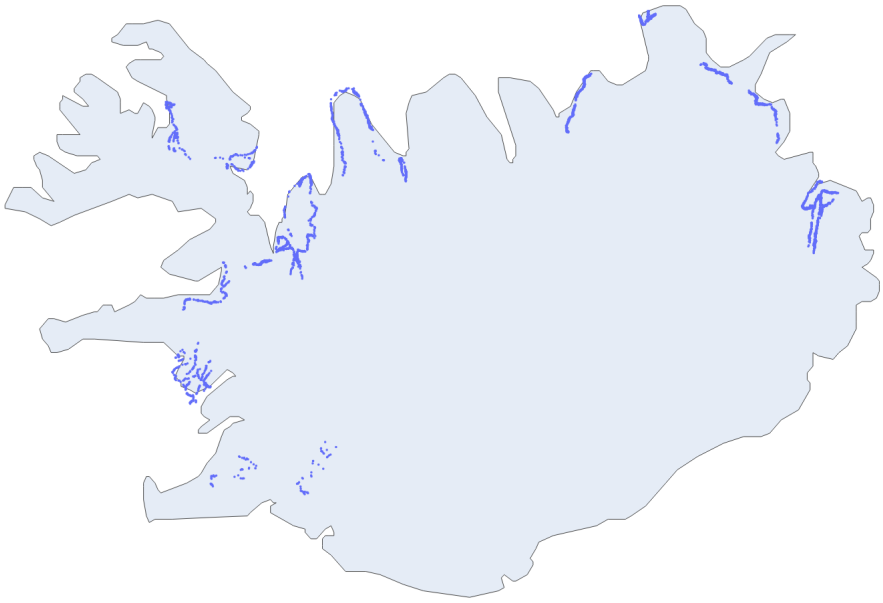
```
            l=0, #left margin
            r=0, #right margin
            b=0, #bottom margin
            t=50  #top margin
          ),

    geo = dict(
            resolution = 50,
            lonaxis_range= [-31,-6],
            lataxis_range= [63, 67],
        )
    )
figGBMrio.update_geos(projection_type="mercator")
figGBMrio.write_image("images/figGBMrio.png", width=3000, height=1500)
```

[94]:
```
figENSrio = go.Figure(data=go.Scattergeo(
        lat = LatticePredictionEnsRIO['Latitude'],
        lon = LatticePredictionEnsRIO['Longitude'],
        marker_color = LatticePredictionEnsRIO['PROB_1'],
        marker = dict(
           size = 3.5,
        ),
    )
)

figENSrio.update_layout(
    width=3000,
    height=1500,
    margin = dict(
            autoexpand=True,
            l=0, #left margin
            r=0, #right margin
            b=0, #bottom margin
            t=50  #top margin
          ),

    geo = dict(
            resolution = 50,
            lonaxis_range= [-31,-6],
            lataxis_range= [63, 67],
        )
    )
figENSrio.update_geos(projection_type="mercator")
figENSrio.write_image("images/figENSrio.png", width=3000, height=1500)
```

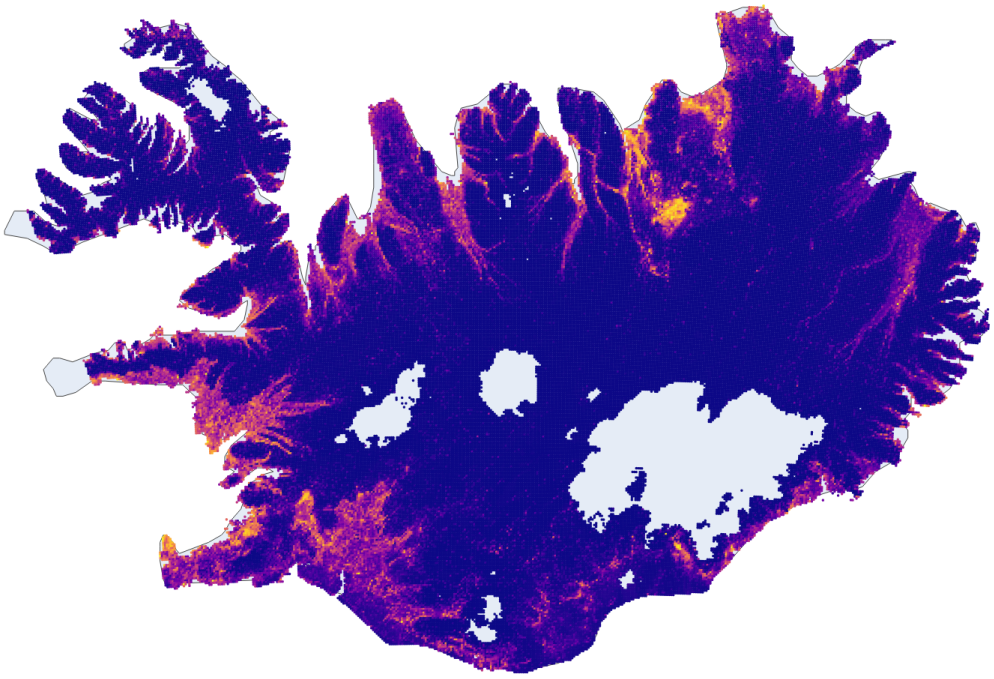
[113]:
```
Trace1=go.Scattergeo(
        lat = LatticePredictionEnsRIO['Latitude'],
```

```
        lon = LatticePredictionEnsRIO['Longitude'],
        marker_color = LatticePredictionEnsRIO['PROB_1'],
        marker = dict(
           size = 3.5,
           opacity = .85,

        ),
    )

Trace2=go.Scattergeo(
        lat = Presence['Latitude'],
        lon = Presence['Longitude'],
        marker = dict(
            size = 3.5,
            color = 'cyan'
        ),
    )


data = [Trace1, Trace2]
figOVERLAY = go.Figure(data)


figOVERLAY.update_layout(
    width=3000,
    height=1500,
    margin = dict(
            autoexpand=True,
            l=0, #left margin
            r=0, #right margin
            b=0, #bottom margin
            t=50  #top margin
          ),

    geo = dict(
            resolution = 50,
            lonaxis_range= [-31,-6],
            lataxis_range= [63, 67],
        )
    )
figOVERLAY.update_geos(projection_type="mercator")
figOVERLAY.write_image("images/figOVERLAY.png", width=3000, height=1500)
```

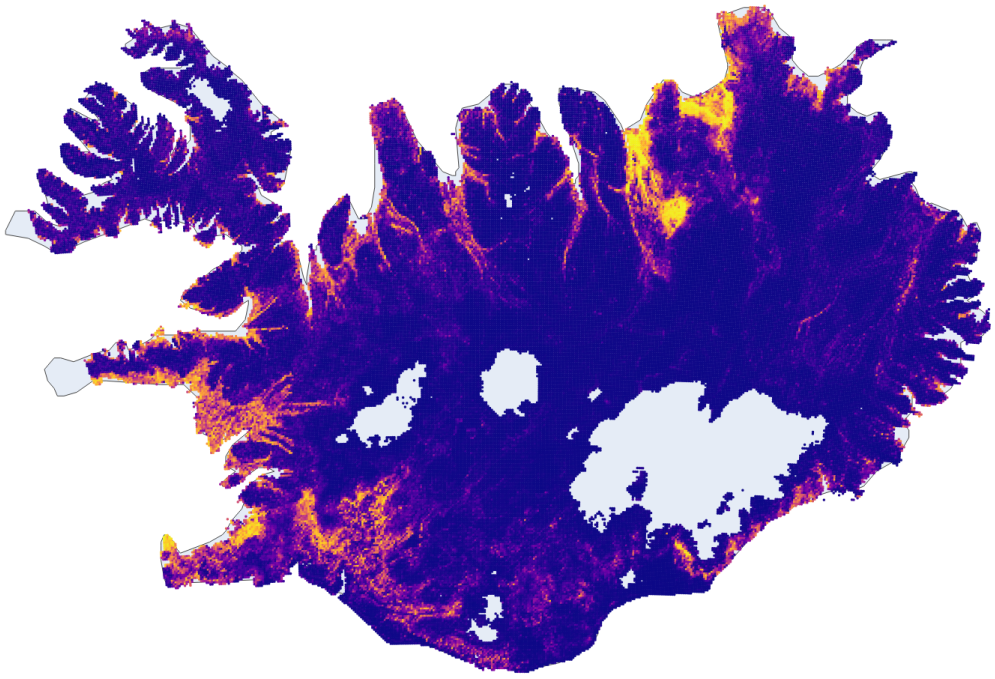
1.2.4 Presence Data



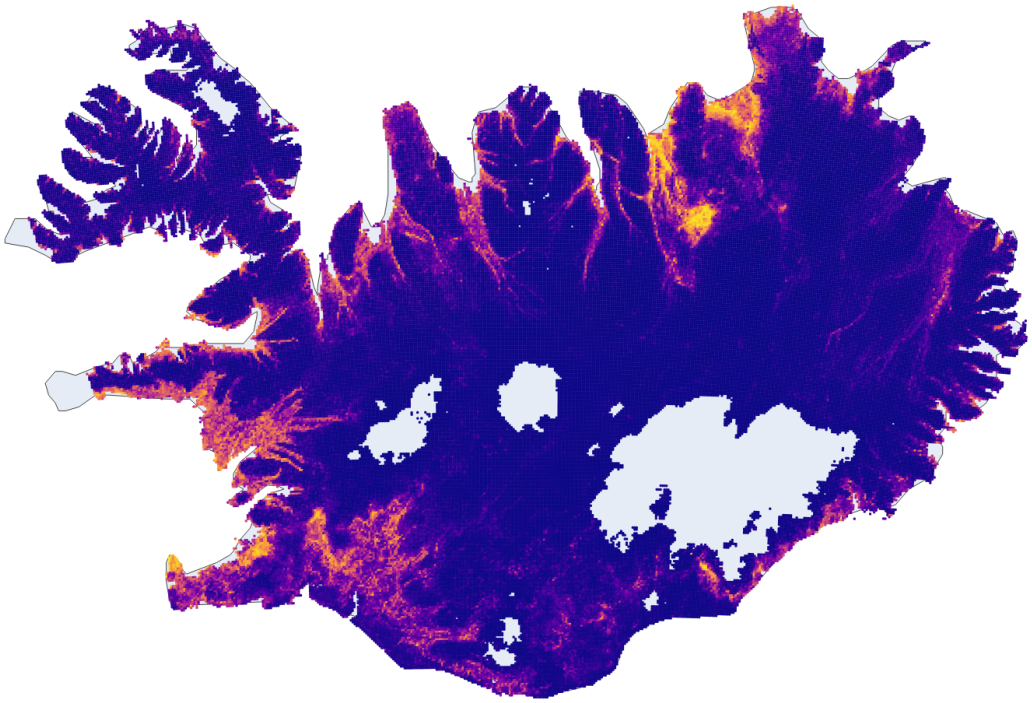
1.2.5 Random Forest RIO Map



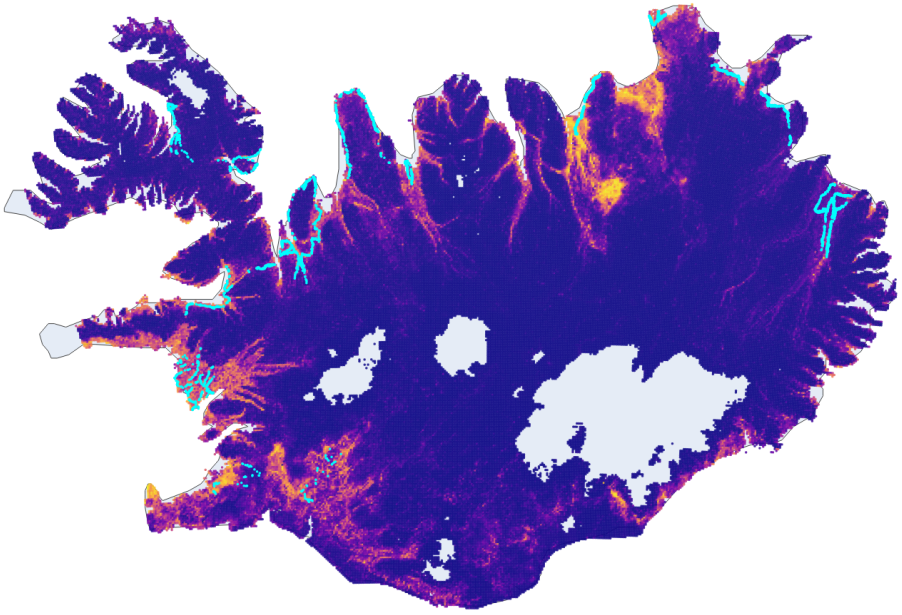
1.2.6 Gradient Boost RIO Map



1.2.7 Ensemble RIO Map



1.2.8 Ensemble RIO Map with Presence Overlay



1.3 Conclusion

Looking at the RIO maps it seems as though we generally have good coverage over all the presence data. In an effort to quantify the quality of our model's prediction I attempted to use the closest lattice point to the presence data. This resulted in incredibly poor predictive results with the model only accuratly predicting presence in less than 1 percent of the data. Clearly a better estimate would come from using some sort of Kriging or k means estimator, for the predictor data at the presence points. I found that Plotly was an incredibly frustrating tool for plotting this type of data. Next time I will likely use something like pyGMT or qGIS create the maps.