Final_Project

April 19, 2022

0.1 OCEN 460 Final Project: Coral Prediction

0.1.1 Team: _/Sample_Text/

0.1.2 Members: James Frizzell and Nate Baker

0.1.3 Aggie Honor Code: By submitting this report, I agree to the following:

Aggies do not lie, cheat, or steal, or tolerate those who do. I have not given or received any unauthorized aid on this assignment.

0.1.4 1. Introduction and Executive Summary

Deep sea coral can be found throughout the world's oceans, from depths of 150 to 10,000 feet. Coral is a living organism that can grow to be dozens of feet tall. Furthermore, it acts as a key habitat to many smaller marine animals, like shrimp, fish, and crustaceans. Perhaps most importantly, deep sea coral can act as an indicator for general ocean health, as it is sensitive to changes in salinity, temperature, and ocean acidity. If the coral is no longer growing or is actively dying, other ocean species may also be affected, causing a massive ripple through ocean ecosystems and to the global scale.

The purpose of this project is to use existing data on the presence of deep sea coral to predict whether said coral can grow, specific given oceanographic conditions. The latitude, longitude, depth, temperature, salinity, and dissolved oxygen levels are used to predict a binary value, where 1 means that coral can grow and 0 means that coral cannot grow. This value will then be multiplied by a factor of 100 to represent a percentage.

Two datasets are used to provide data. The World Ocean Atlas (WOA) provides depth, salinity, dissolved oxygen, and temperature at recorded latitude and longitude values. The temperature values are provided as a function of measured depth. The Deep Sea Coral (DSC) data was found kaggle.com and originates from the National Oceanic and Atmospheric Administration (NOAA). The DSC dataset provides latiude, longitude, and depth values for where coral was recorded to be present.

Following data acquisition and background domain research, the two datasets were geospatially joined to match the correspondent latitude and longitude values.

The columns of this dataset are: Latitude | Longitude | Depth (m) | Temperature (c) | Salinity (ppt) | Dissolved Oxygen (umol/kg).

Once a single dataset was compiled, the correlation coefficients of the input variables were determined. Following proper data science practices reagarding model fit, the database was split into a training and a test dataset at a 80:20 ratio. To find the optimal model architecture for the data,

several hundred architectures were generated with varying numbers of neurons and hidden layers. Each was fit and the following accuracy metrics were evaluated: AIC, R Squared (R2), Mean Average Error (MAE), and Root Mean Squared Error (RMSE). The best model was then selected based on the least AIC values and the greatest R2 value. This geometry was then further evaluated with different activation functions, dropout layers, and variable training epochs. Finally, the working network was bundled into a package, allowing users to input ocean conditions and outputting a likelihood of coral growth.

This project is available to follow on Github at the following link: https://github.com/jafrizzell/coral-prediction.git

[1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import pathlib
import os

%matplotlib inline
#Describe Datasets and project idea
```

0.1.5 2. Data Intake

The deep sea coral dataset reports latitude and longitude of known coral growth locations with the depth at which the coral is growing. The World Ocean Atlas reports depth measurements in increments of 5 meters for depths of 0 to 100 meters, 10 meters for 100 to 500 meters, 50 meters for 500 to 2000 meters, and in 100 meters for greater than 2000 meters. The following code was used and adjusted to round the Deep Sea Coral dataset to match this convention.

[2]:
```
path = 'C:/Users/jafri/Documents/GitHub/coral-prediction/processed_data/
 ↪deep_sea_corals_rounded.csv'

raw = pd.read_csv(path)

def round_depth(x, base):
    return int(base * round(float(x)/base))


raw['depth'] = raw['depth'].apply(lambda x: round_depth(x, base=5))

raw = raw[raw.depth >= 0]
print(len(raw))
raw.to_csv('C:/Users/jafri/Documents/GitHub/coral-prediction/processed_data/
 ↪deep_sea_corals_rounded_depthcorr.csv')
```

0.1.6 3. Comparison and Alignment of Both Datasets

The following code aligns latitude and longitude values from the Deep Sea Coral dataset with the lat/long values from the WOA dataset with a tolerance of 0.5 degrees. Second_param file can be changed to indicate the oceanographic variable of interest. WOA data is right-joined to DSC data for further preprocessing.

This code yields a .csv file that contains the DSC data and the WOA data. The WOA data is depth-stratified.

[3]:
```
import geopandas

coral = 'D:/TAMU Work/TAMU 2022 SPRING/OCEN 460/depthtempsal_short2.csv'
second_param = 'D:/TAMU Work/TAMU 2022 SPRING/OCEN 460/woa18_all_O00mn01.csv'
 ↪# "O00mn01" indicates O2 data

raw_coral = pd.read_csv(coral)
raw_coral = geopandas.GeoDataFrame(raw_coral, geometry=geopandas.
 ↪points_from_xy(raw_coral.longitude, raw_coral.latitude))
raw_coral.depth = raw_coral.depth.astype(float)
raw_coral.latitude = raw_coral.latitude.astype(float)
raw_coral.longitude = raw_coral.longitude.astype(float)

raw_param = pd.read_csv(second_param)
raw_param = raw_param.astype(float)
raw_param = geopandas.GeoDataFrame(raw_param, geometry=geopandas.
 ↪points_from_xy(raw_param.longitude, raw_param.latitude))


depth_sal = raw_coral.sjoin_nearest(raw_param, max_distance=0.5)

depth_sal.to_csv('D:/TAMU Work/TAMU 2022 SPRING/OCEN 460/depthtempsaloxy.csv',
 ↪index=False)
```

To resolve the stratified nature of the WOA data, the following code is used to select the corresponding WOA column for the DSC depth of interest.

[4]:
```
path = 'D:/TAMU Work/TAMU 2022 SPRING/OCEN 460/depthtempsaloxy.csv'

raw = pd.read_csv(path)
raw = raw[raw['depth'] <= 5500]
skipped = 0
for i in range(len(raw)):
    try:
        depth = str(raw['depth'][i])
        raw['oxygen'][i] = raw[depth][i]
    except KeyError:
        skipped+=1
        pass

print("skipped:", skipped)

raw.to_csv('D:/TAMU Work/TAMU 2022 SPRING/OCEN 460/depthtempsaloxy_short.csv',
 ↪index=False)
```

0.1.7 4. Creation of a Control Dataset

The following code determines the maximum depth for each lat/long pair in the WOA dataset. These datapoints were then used to create a control dataset describing where coral is not present, in order to compare to the DSC dataset. Code displayed in sections 2 and 3 were used to add the temperature, salinity, and oxygen variables to the control dataset.

[5]:
```
path = 'D:/TAMU Work/TAMU 2022 SPRING/OCEN 460/woa18_decav_t00mn04.csv'

raw = pd.read_csv(path)
depth = []

for i in range(len(raw)):
    for j in range(103):
        curr = raw.iloc[i, -1-j]
        plus = raw.iloc[i, -2-j
        if j == 0 and np.isfinite(curr):
            depth.append(raw.columns[-1])
            break
        elif np.isnan(curr) and np.isfinite(plus):
            depth.append(raw.columns[-2-j])
            break
        elif j == 102:
            depth.append(raw.columns[-1])
print(len(depth))
print(len(raw.latitude))
out = pd.DataFrame({'latitude': raw.latitude,
                    'longitude': raw.longitude,
                    'depth': depth})

out.to_csv('D:/TAMU Work/TAMU 2022 SPRING/OCEN 460/depths.csv', index=False)
```

Ultimately, the final dataset had the following metadata: Latitude | Longitude | Depth (m) | Temperature (c) | Salinity (ppt) | Dissolved Oxygen (umol/kg) .

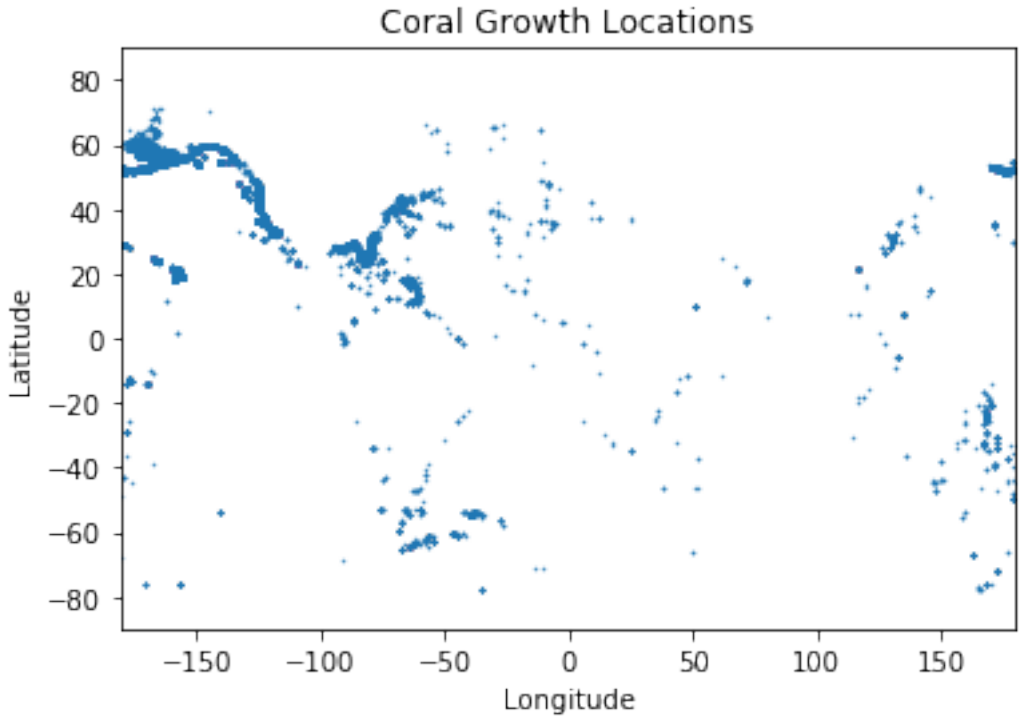
0.1.8 5. Dataset Visualization

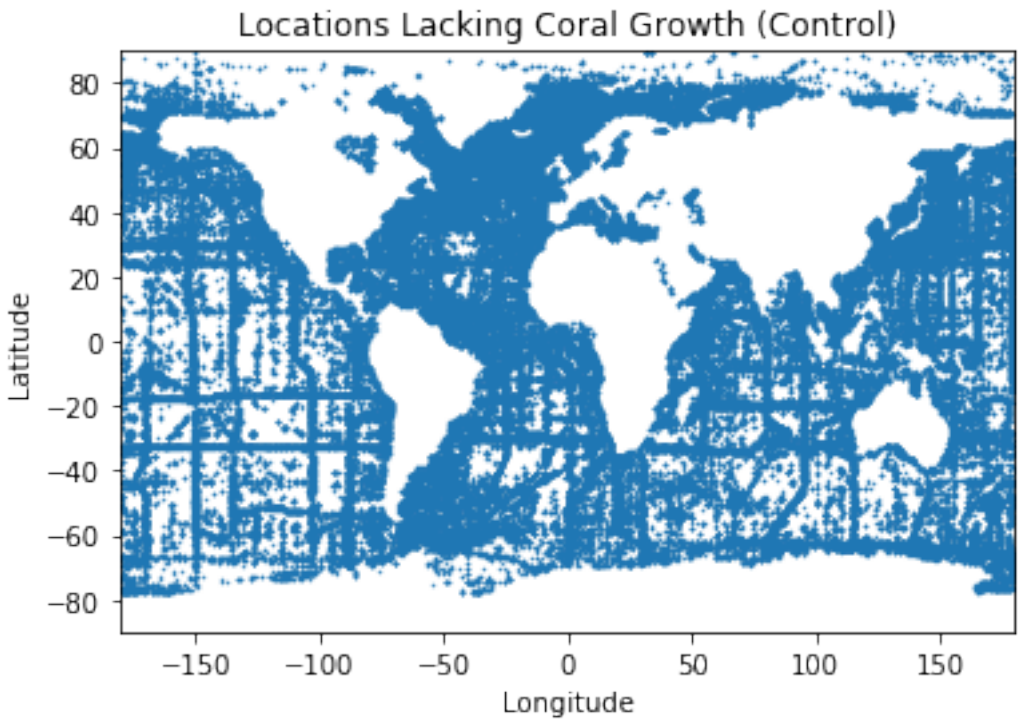
[6]:
```
#Reprocessed Data for Visualization
path = str(pathlib.Path(os.getcwd())) +␣
 ↪'\processed_data\combined_data_truncated.csv'
raw = pd.read_csv(path)
print(raw.describe())

#Visualization
coral_present_bool = raw[raw.coral_present == 1]
plt.scatter(coral_present_bool['longitude'], coral_present_bool["latitude"], s=␣
 ↪0.2)
plt.title("Coral Growth Locations")
```

```
plt.xlabel("Longitude")
plt.ylabel("Latitude")
plt.xlim([-180,180])
plt.ylim([-90,90])
plt.show()

coral_missing_bool = raw[raw.coral_present == 0]
plt.scatter(coral_missing_bool['longitude'], coral_missing_bool["latitude"], s=␣
↪0.2)
plt.title("Locations Lacking Coral Growth (Control)")
plt.xlabel("Longitude")
plt.ylabel("Latitude")
plt.xlim([-180,180])
plt.ylim([-90,90])
plt.show()

print("Number of Coral Growth Datapoints:", len(coral_present_bool))
print("Number of Datapoints with no Coral Growth", len(coral_missing_bool))

plt.scatter(raw.longitude, raw.latitude, s=0.2, c=raw.depth)
plt.title("Cumulative Dataset, Colored By Depth")
plt.xlabel("Longitude")
plt.ylabel("Latitude")
plt.xlim([-180,180])
plt.ylim([-90,90])
plt.colorbar()
plt.show()
plt.scatter(raw.longitude, raw.latitude, s=0.2, c=raw.temperature)
plt.title("Cumulative Dataset, Colored By Temperature")
plt.xlabel("Longitude")
plt.ylabel("Latitude")
plt.xlim([-180,180])
plt.ylim([-90,90])
plt.colorbar()
plt.show()
plt.scatter(raw.longitude, raw.latitude, s=0.2, c=raw.salinity)
plt.title("Cumulative Dataset, Colored By Salinity")
plt.xlabel("Longitude")
plt.ylabel("Latitude")
plt.xlim([-180,180])
plt.ylim([-90,90])
plt.colorbar()
plt.show()
plt.scatter(raw.longitude, raw.latitude, s=0.2, c=raw.oxygen)
plt.title("Cumulative Dataset, Colored By Oxygen")
plt.xlabel("Longitude")
plt.ylabel("Latitude")
```

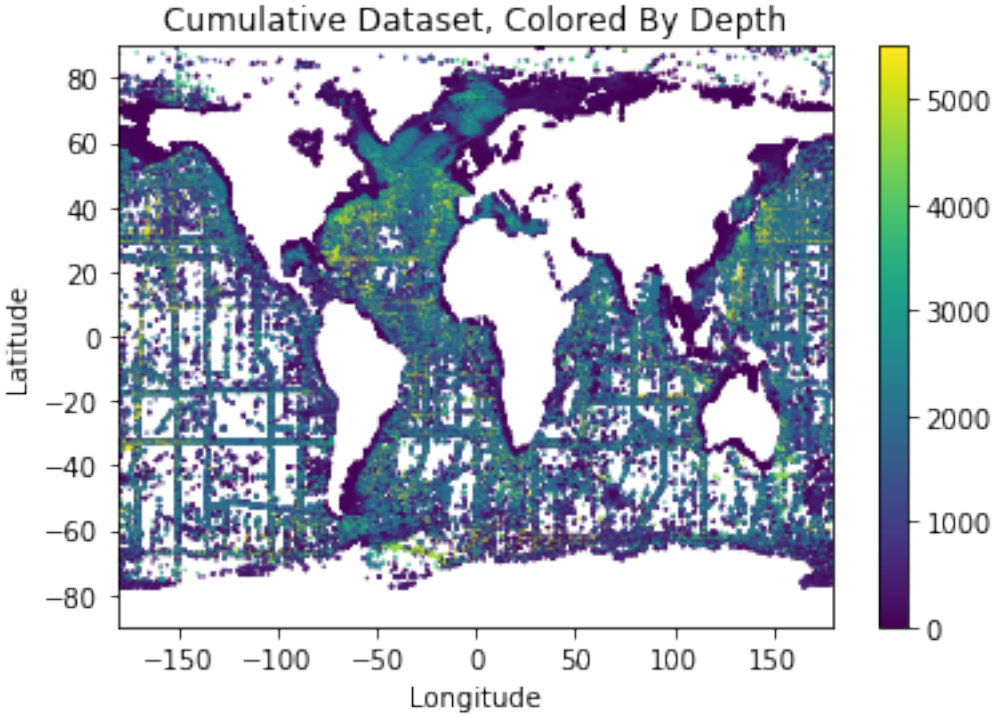
```
plt.xlim([-180,180])
plt.ylim([-90,90])
plt.colorbar()
plt.show()
```

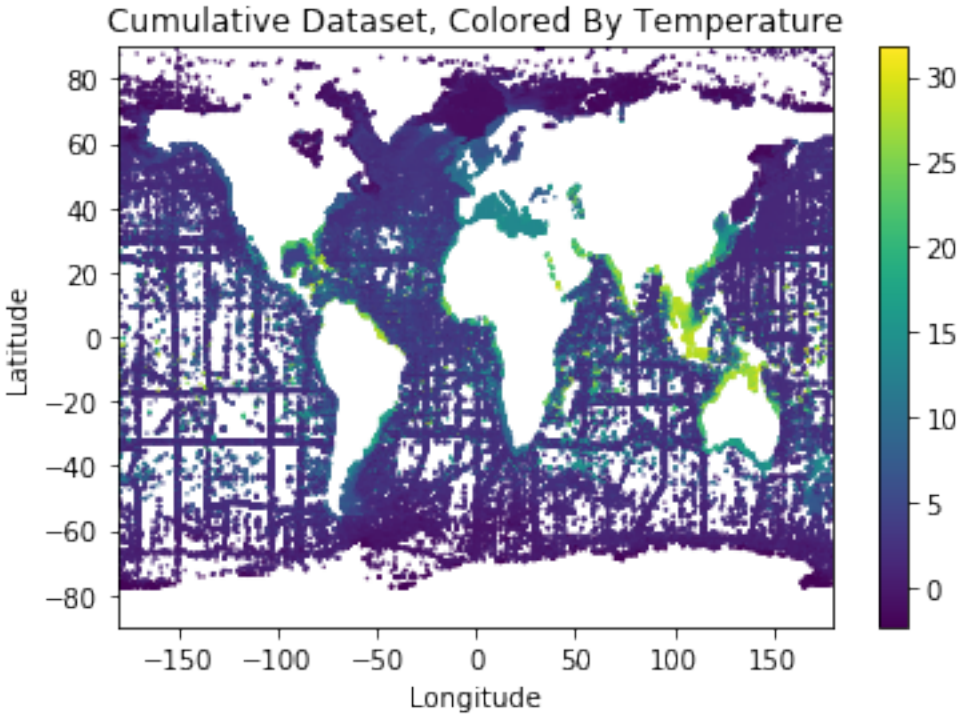
```
       coral_present       latitude      longitude          depth  \
count  341773.000000  341773.000000  341773.000000  341773.000000
mean        0.582340      23.237462     -68.404081    1066.013070
std         0.493174      33.783962      96.027313     989.540361
min         0.000000     -77.875000    -179.989750       0.000000
25%         0.000000      16.625000    -124.339620     235.000000
50%         1.000000      35.641580    -119.498760     850.000000
75%         1.000000      40.811190     -25.625000    1750.000000
max         1.000000      89.875000     179.989750    5500.000000

              round_d    temperature       salinity         oxygen
count  341773.000000  341773.000000  341773.000000  341773.000000
mean     1067.012915       5.191193      34.392171      42.294298
std       989.771161       4.584556       1.343642      29.076344
min         0.000000      -2.271000       0.000000       0.199000
25%       225.000000       2.452000      34.228000      12.890000
50%       850.000000       3.878000      34.520000      41.558000
75%      1750.000000       7.371000      34.699000      65.495000
max      5500.000000      31.751000      41.310000     132.182000
```

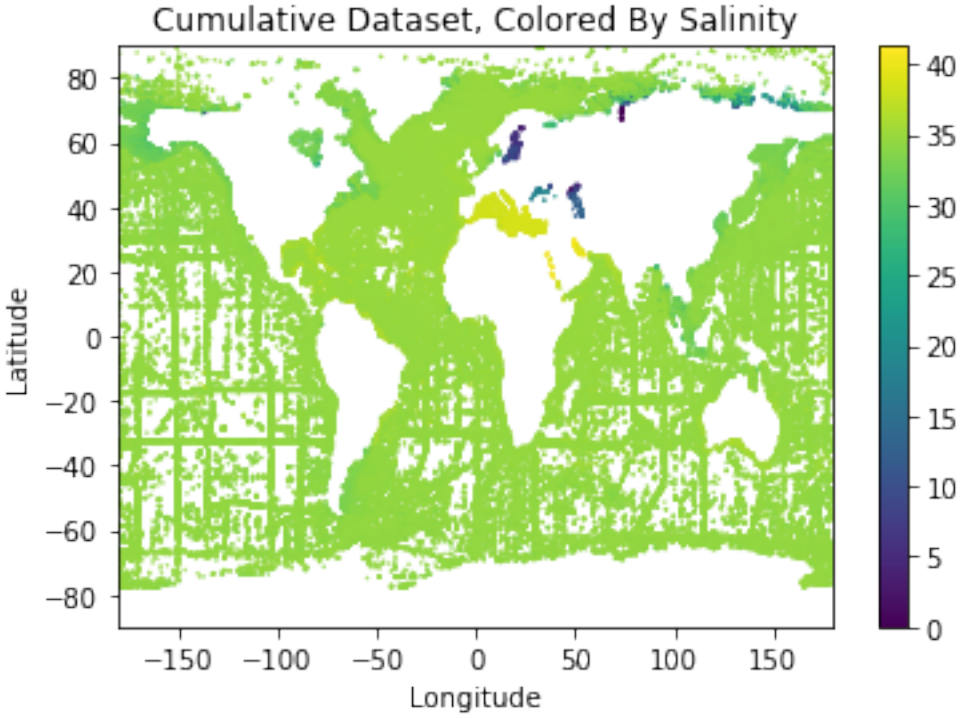


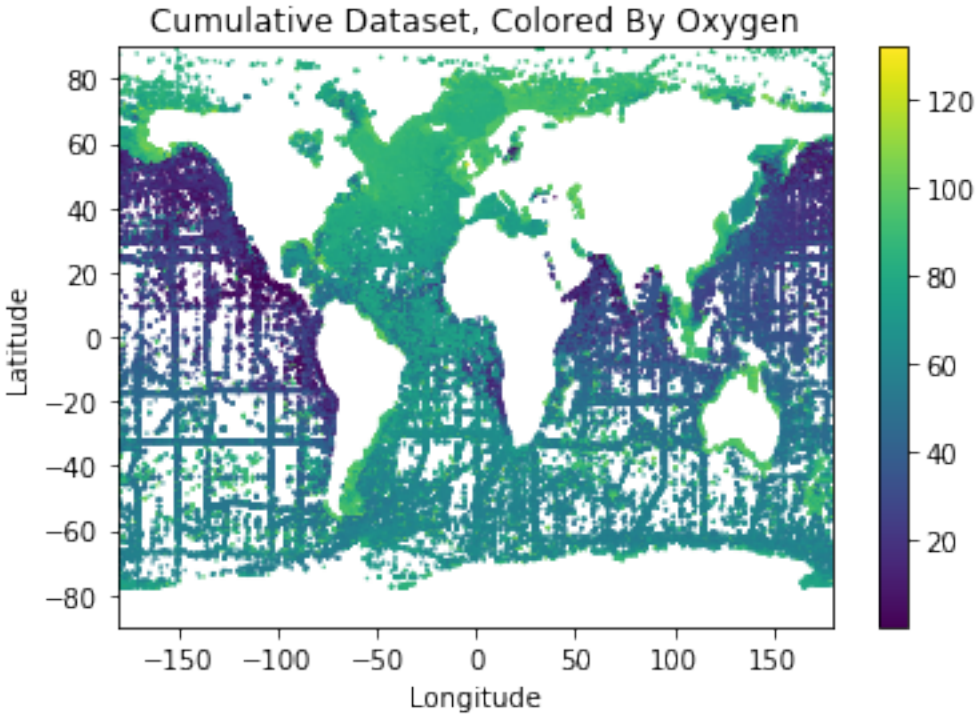



```
Number of Coral Growth Datapoints: 199028
Number of Datapoints with no Coral Growth 142745
```




Cumulative Dataset, Colored By Temperature
Latitude
Longitude
80
60
40
20
0
−20
−40
−60
−80
−150
−100
−50
0
50
100
150
30
25
20
15
10
5
0


Cumulative Dataset, Colored By Salinity
Latitude
Longitude
80
60
40
20
0
−20
−40
−60
−80
−150
−100
−50
0
50
100
150
40
35
30
25
20
15
10
5
0



0.1.9 6.1 Pattern Extraction & Why Algorithm Chosen (+ Parameter Tuning)

[7]:
```
import os
import tensorflow as tf
import pandas as pd
import numpy as np
from tensorflow import keras
from tensorflow.keras import layers
import matplotlib.pyplot as plt
import RegscorePy
from math import sqrt, floor
import pathlib
from itertools import product
from scipy.stats import pearsonr
import time
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
%matplotlib inline
```

Imports the necessary packages. Some uncommon ones are RegscorePy, which is a custom open-source library used to calculate the Akaike Information Criterion (AIC) of tensorflow models, and itertools.product which is used to generate all combinations of elements in a list.

0.1.10 6.2 Load Data In

The raw data is loaded in from a csv file and processed into a training and testing dataset.

[8]:
```
cwd = pathlib.Path(os.getcwd())
path = str(cwd.parent) + '/coral-prediction/processed_data/
 ↪combined_data_truncated.csv'

raw = pd.read_csv(path)


raw = raw.sample(frac=0.2, random_state=0)

print(raw.describe())

raw.pop('species')
raw.pop('round_d')

train = raw.sample(frac=0.8, random_state=0)
test = raw.drop(train.index)

train_features = train.copy()
test_features = test.copy()

train_labels = train_features['coral_present']
test_labels = test_features['coral_present']
train_features.pop('coral_present')
test_features.pop('coral_present')
```

```
       coral_present      latitude     longitude         depth       round_d  \
count   68355.000000  68355.000000  68355.000000  68355.000000  68355.000000
mean        0.584273     23.283085    -68.820676   1063.477800   1064.483798
std         0.492850     33.799299     95.889386    985.059453    985.321543
min         0.000000    -77.875000   -179.966080      0.000000      0.000000
25%         0.000000     16.875000   -124.591595    235.000000    225.000000
50%         1.000000     35.652870   -119.501450    855.000000    850.000000
75%         1.000000     40.811420    -26.375000   1735.000000   1750.000000
max         1.000000     89.625000    179.973800   5500.000000   5500.000000

        temperature      salinity        oxygen
count  68355.000000  68355.000000  68355.000000
mean       5.175744     34.384281     42.198043
std        4.551002      1.371827     29.077581
min       -2.248000      0.022000      0.434000
25%        2.457000     34.228000     12.890000
50%        3.878000     34.520000     41.558000
75%        7.371000     34.696000     65.169500
max       31.751000     41.310000    120.705000
```

```
[8]: 304228    0
     213311    0
     172619    1
     141430    1
     133490    1
     250266    0
     6251      1
     199933    0
     329023    0
     16799     1
     59488     1
     133226    1
     295199    0
     280443    0
     240202    0
     2107      1
     264787    0
     52521     1
     313974    0
     227559    0
     271879    0
     93042     1
     249838    0
     222392    0
     267409    0
     181223    1
     95179     1
     135531    1
     326845    0
     310983    0
               ..
     305227    0
     155537    1
     95610     1
     264519    0
     96039     1
     24083     1
     186999    1
     64262     1
     195875    1
     140431    1
     154380    1
     298923    0
     177985    1
     266179    0
     333985    0
     102172    1
```

```
177471    1
142577    1
16905     1
123929    1
161761    1
202345    0
132827    1
41560     1
262989    0
267861    0
209754    0
192209    1
36135     1
278318    0
Name: coral_present, Length: 13671, dtype: int64
```

0.1.11 7. Feature Evaluation - Correlation Coefficients

The correlation coefficients and p-values for each feature are reported. Since all p-values are <0.05, each feature selected is relevant to the model and will be kept.

It is interesting that the longitude is strongly negatively correlated to the presence of coral. This implies that at more eastern locations (say 90E - 180E) it is less likely that coral will grow. This may be a residual of other conditions, such as the much deeper waters in the eastern Pacific Ocean - since the depth is also negatively correlated.

[9]:
```
from scipy.stats import pearsonr
print(train.corr()['coral_present'])
print(pearsonr(train_features['latitude'], train_labels))
print(pearsonr(train_features['longitude'], train_labels))
print(pearsonr(train_features['depth'], train_labels))
print(pearsonr(train_features['temperature'], train_labels))
print(pearsonr(train_features['salinity'], train_labels))
print(pearsonr(train_features['oxygen'], train_labels))
```

```
coral_present    1.000000
latitude         0.483094
longitude       -0.623733
depth           -0.381216
temperature      0.244996
salinity        -0.075697
oxygen          -0.484223
Name: coral_present, dtype: float64
(0.4830940064618191, 0.0)
(-0.6237334337917884, 0.0)
(-0.3812161882542593, 0.0)
(0.24499644116438987, 0.0)
(-0.07569703383007981, 2.618925049594667e-70)
(-0.4842228643654894, 0.0)
```

0.1.12 8.1 Definition of Functions

The following functions were used during the training and evaluation of the model.

fit_and_evaluate() accepts a model architecture and fits the training data to it, and the reports the accuracy metrics based on the test dataset.

The hyperparameters selected for the training are: 20% validation split and 50 epochs training duration. These were selected by testing different configurations and choosing the hyperparameters that resulted in the most accurate model.

plot_loss() accepts the model training residuals and plots them over the training duration (number of epochs) the loss and validation loss are both shown.

add_layer() is a part of the parametric model study, which can add hidden layers to a tensorflow model by passing parameters and hyperparameters. This functionality will hopefully be bundled into a package at some point so that users can pip install the ability to do a parametric study.

build_and_compile_model() accepts the model architecture from the user and creates the tensorflow model. It then fits the model and performs the accuracy evaluations.

[10]:
```
def fit_and_evaluate(architecture):
    dnn_model = build_and_compile_model(architecture)

    history = dnn_model.fit(train_features, train_labels, validation_split=0.2,␣
↪verbose=0, epochs=50)
    plot_loss(history)

    test_results = dnn_model.evaluate(test_features, test_labels, verbose=0)
    test_predictions = dnn_model.predict(test_features).flatten()
    r2= r2_score(np.asarray(test_labels).flatten(), test_predictions)
    mae = mean_absolute_error(np.asarray(test_labels).flatten(),␣
↪test_predictions)
    aic = RegscorePy.aic.aic(np.asarray(test_labels, dtype=float).flatten(), np.
↪asarray(test_predictions).astype(float), 4+2)
    rmse = sqrt(mean_squared_error(np.asarray(test_labels).flatten(),␣
↪test_predictions))
    return dnn_model, aic, r2, mae, rmse, test_predictions
```

[11]:
```
def plot_loss(history):
    plt.plot(history.history['loss'], label='loss')
    plt.plot(history.history['val_loss'], label='val_loss')
    # plt.ylim([0, 30])
    plt.xlabel('Epoch')
    plt.ylabel('Error')
    plt.legend()
    plt.grid(True)
    plt.show()
    # pass
```

[12]:
```
def add_layer(dets, hyper, prev):
    default = ['relu']
    try:
        layer = layers.Dense(dets, activation=hyper[0])(prev)
    except IndexError:
        layer = layers.Dense(dets, activation=default[0])(prev)
    return layer
```

[13]:
```
def build_and_compile_model(arch):
    # Adjust the number of hidden layers and neurons per layer that results in␣
 ↪best fit NN
    hidden_layers = []
    inputs = keras.Input(shape=(6,))
    norm_layer = layers.BatchNormalization()(inputs)
    hidden_layers.append(inputs)
    hidden_layers.append(norm_layer)
    for i in range(num_hidden):
        if arch[i] == 0:
            pass
        else:
            layer = add_layer(arch[i], arch[num_hidden:], hidden_layers[-1])
            hidden_layers.append(layer)
            layer = layers.Dropout(rate=0.2)(hidden_layers[-1])
            hidden_layers.append(layer)
    outputs = layers.Dense(1)(hidden_layers[-1])
    hidden_layers.append(outputs)
    model = keras.Model(inputs=inputs, outputs=outputs)

    model.compile(loss='mean_absolute_error',
                  optimizer=tf.keras.optimizers.Adam(0.001))
    return model
```

0.1.13 8.2 Implementation of Model Trainer

The following code sets up the necessary data to train the models. By commenting out the line:

list(product(*[l1, l2, l3, activ])), the parametric search is deactivated, and only the architecture specified in the next line is fitted

Additionally, some timing features are implemented. Passing a large parametric space can result in the program running for over 4 hours, training each model. Because of this, a progress meter is added so that the user can see how far along the program is.

[14]:
```
models = []
aic_scores = []
r2_scores = []
maes = []
rmses = []
```

```
l1 = np.linspace(32, 256, 5)
l2 = np.linspace(0, 256, 5)
l3 = np.linspace(0, 256, 5)
activ = ['relu', 'tanh']
# parametric_space = list(product(*[l1, l2, l3, activ]))
parametric_space = [[200, 64, 192, 'relu']]
print(parametric_space)
num_hidden = len(list(i for i in parametric_space[0] if isinstance(i, (int or␣
↪float))))
num_hyper = len(parametric_space[0]) - num_hidden
start_t = time.time()
c = 1
```

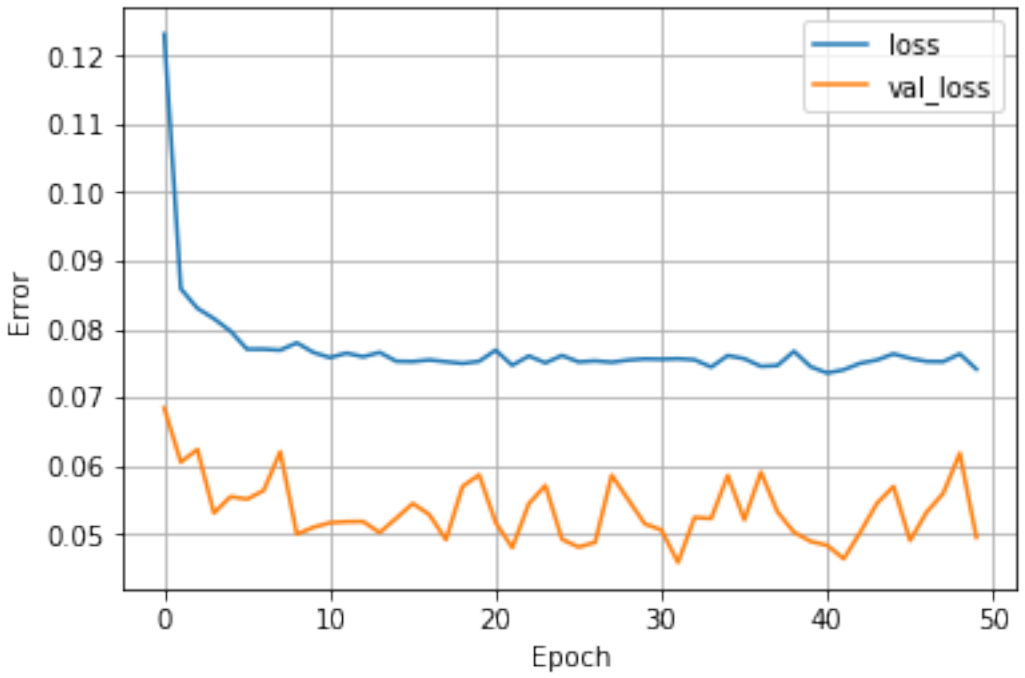
```
[[200, 64, 192, 'relu']]
```

0.1.14 9. Running the Training Model

This for loop passes each architecture specified in the parametric space into the build_and_compile_model() function and reports the accuracy metrics into their specific list. It also updates the progress each time a model is finished evaluation

[15]:
```
for arch in parametric_space:
    print('Progress: ' + str(c) + '/' + str(len(parametric_space)))
    dnn_model, aic, r2, mae, rmse, test_predictions = fit_and_evaluate(arch)
    models.append(dnn_model)
    aic_scores.append(aic)
    r2_scores.append(r2)
    maes.append(mae)
    rmses.append(rmse)

    curr_time = time.time()
    diff_t = curr_time - start_t
    t_per_model = diff_t / c
    num_mods_rem = len(parametric_space) - c
    t_rem = t_per_model * num_mods_rem
    print("Estimated Time Remaining: " + time.strftime('%H:%M:%S', time.
↪gmtime(t_rem)) + ' seconds')
    c += 1
```

```
Progress: 1/1
```




```
Estimated Time Remaining: 00:00:00 seconds
```

0.1.15 10. Selection of the Best Model

After the parametric seach is finished, the accuracy metrics are compiled into a csv file. The user must look through these results and pick the model which has the LOWEST AIC score and HIGHEST R-Squared.

The final 2 lines save the zeroth model in the parametric space for later use. During the parametric seach, this should be disabled because the zeroth model is likely not the most accurate. When the parametric search is disabled (only a single architecture is being fitted) this can be re-enabled to save the model.

[16]:
```
parametric_space_t = np.asarray(parametric_space).transpose().tolist()
output_data = [parametric_space_t[0], parametric_space_t[1],␣
 ↪parametric_space_t[2], aic_scores, maes, rmses, r2_scores]
output_data = np.asarray(output_data).transpose().tolist()
print(output_data)
oput = pd.DataFrame(output_data, columns=['L1', 'L2', 'L3', 'AIC', 'MAE',␣
 ↪'RMSE', 'R2'])
# print(oput)
# oput.to_csv('Parametric_space_study.csv', index=False)
print(models[0].summary())
out_path = str(cwd.parent) + '/models/trial0.3.h5'
# models[0].save(out_path)
```

```
[['200', '64', '192', '-42396.7584440718', '0.04872919971817529',
'0.21202562322987348', '0.8151878327057019']]
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 6)]               0
_________________________________________________________________
batch_normalization (BatchNo (None, 6)                 24
_________________________________________________________________
dense (Dense)                (None, 200)               1400
_________________________________________________________________
dropout (Dropout)            (None, 200)               0
_________________________________________________________________
dense_1 (Dense)              (None, 64)                12864
_________________________________________________________________
dropout_1 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_2 (Dense)              (None, 192)               12480
_________________________________________________________________
dropout_2 (Dropout)          (None, 192)               0
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 193
=================================================================
Total params: 26,961
Trainable params: 26,949
Non-trainable params: 12
_________________________________________________________________
None
```

0.1.16 11. Setup of Class Structure

This code has been written in an object-oriented fashion so that it may easily be packaged and shipped into production later.

First, we set up the class structure and import the internal libraries used.

[17]:
```
import random

import tensorflow as tf
import numpy as np
from itertools import product
from statistics import mean, median
```

[22]:
```
class CoralPrediction:
    def __init__(self):
        self.model = None
        self.params = np.zeros(6)
```

```
        self.test_lat = []
        self.test_long = []
        self.test_depth = []
        self.test_temp = []
        self.test_sal = []
        self.test_oxy = []

    def set_model(self, modelpath):
        self.model = tf.keras.models.load_model(modelpath)

    def predict(self, params):
        self.params = params
        missing = []
        for i in range(len(self.params)):
            if type(self.params[i]) != float and type(self.params[i]) != int:
                missing.append(i)

        if 0 in missing:
            self.test_lat = np.linspace(-90, 90, 90)
        else:
            self.test_lat = [self.params[0]]

        if 1 in missing:
            self.test_long = np.linspace(-180, 180, 180)
        else:
            self.test_long = [self.params[1]]

        if 2 in missing:
            self.test_depth = np.linspace(0, 3000, 50)
        else:
            self.test_depth = [self.params[2]]

        if 3 in missing:
            self.test_temp = np.linspace(-2, 28, 20)
        else:
            self.test_temp = [self.params[3]]

        if 4 in missing:
            self.test_sal = np.linspace(0, 41, 20)
        else:
            self.test_sal = [self.params[4]]

        if 5 in missing:
            self.test_oxy = np.linspace(0.2, 132, 40)
        else:
            self.test_oxy = [self.params[5]]
```

```
        cond_list = list(product(*[self.test_lat, self.test_long, self.
↪test_depth, self.test_temp, self.test_sal, self.test_oxy]))
        if len(cond_list) >= 4000:
            cond_list = random.sample(cond_list, 4000)
        count = 0
        predictions = []
        for cond in cond_list:
            if count % 100 == 0:
                print(count, ' completed out of: ', len(cond_list))
            predictions.append(self.model.predict([list(cond)])[0][0])
            count += 1
        plt.hist(predictions)
        plt.title("Distribution of Predictions")
        plt.xlabel("Likelihood %/100")
        plt.ylabel("Count")
        return mean(predictions), median(predictions)
```

The CoralPrediction class has a few properties and methods. Once the object has been instantiated in the main.py function, the user will assign a trained tensorflow model to the predictor using the set_model function.

Then, the user will pass the sea state conditions into the predictor model with the predict function. The code will evaluate the conditions and return the probability of coral being able to grow given the inputs.

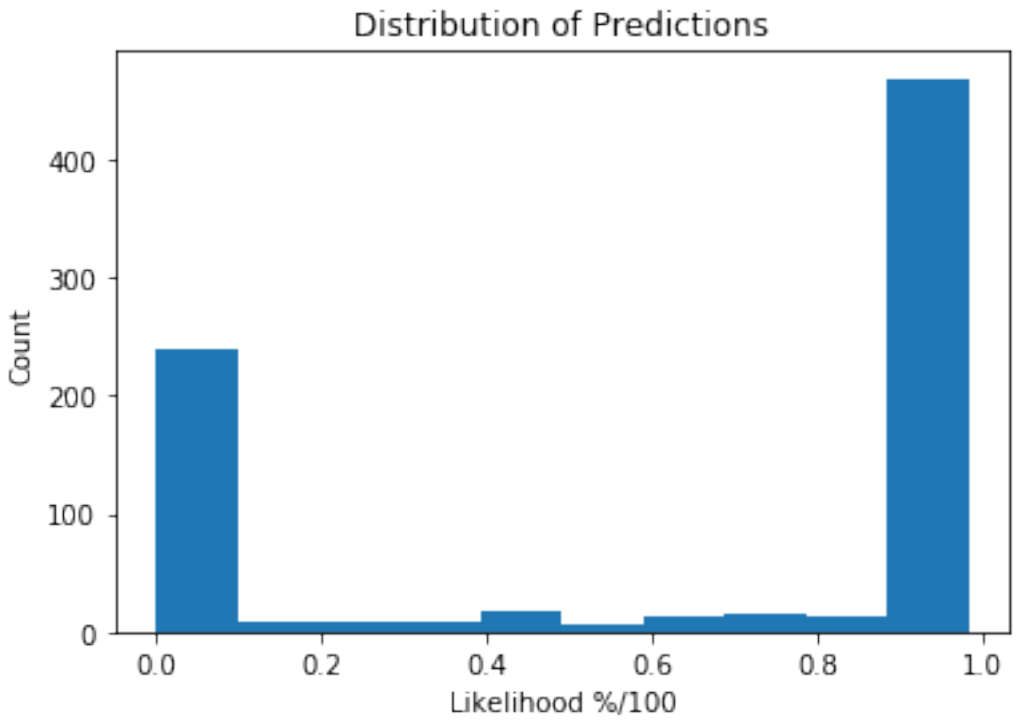
Some precautions have been implemented to assist the user in the case that not all metocean data is known. This will be discussed further in section 3.

0.1.17 12. MAIN.py, Runnable Function

[23]:
```
import os
import pathlib
# import CoralClass


def __main__(modelpath, conditions):
    CoralPredictor = CoralPrediction()
    CoralPredictor.set_model(modelpath)
    mean_pred, med_pred = CoralPredictor.predict(conditions)
    print('\n----------------------------------------')
    print('The average likelihood of coral growth is: ', mean_pred*100, '%')
    print('The median likelihood of coral growth is: ', med_pred*100, '%')


# if __name__ == '__main__':           Failed implementation of command-line␣
↪run functionality
#     import sys
#     args = sys.argv[2:]
```

```
#     modelpath = sys.argv[1]
#     pythonname = sys.argv[0]
#     __main__(modelpath, args)

# change this to change the model that will be used to predict the coral growth
modelpath = str(pathlib.Path(os.getcwd()).parent) + '/coral-prediction/models/
 ↪trial0.3.h5'

# Enter the metocean conditions here   [Latitude, Longitude, Depth (m),␣
 ↪Temperature (C), Salinity (ppt), Dissolved O2 (umol/kg)]
# Enter "None" if the data is unknown
conditions = [20, -150, 1000, None, 30, None]

__main__(modelpath, conditions)
```

```
0  completed out of:  800
100  completed out of:  800
200  completed out of:  800
300  completed out of:  800
400  completed out of:  800
500  completed out of:  800
600  completed out of:  800
700  completed out of:  800

----------------------------------------
The average likelihood of coral growth is:  63.69994878768921 %
The median likelihood of coral growth is:  98.32048416137695 %
```



This code allows the user to input conditions and see the output. First, the user must specify the path of a trained tensorflow model which will be used to predict the growth of the coral.

Then, the user inputs metocean conditions and runs the program.

The code will report the mean and median likelihood that coral can grown in the given conditions. If all the parameters are specified, the mean will equal the median (since there is only 1 data point that is predicted)

0.1.18 13. Probablistic Modelling

In many real-world cases, not all the data will be known for the location of interest. For example, a climatologist may be studying the South Pacific for coral growth around the Great Barrier Reef. The scientist will have a GPS coordinate of interest (latitude and longitude) and likely will know the ocean depth at that location. However, the temperature, salinity and dissolved oxygen content may be unknown. How can the research continue if the input data is incomplete?

This is solved with probabilistic prediction. In the case that some inputs are not known (are entered as "None" in the inputs) the code will sweep through a range of possible values based on the training data that was earlier used. For each value in the range, the program will predict the coral growth probability. Once all the possible values have been simulated, the code will take the average and median probability and report it back to the user.

0.1.19 14. Final Conclusions

This project has been successful in predicting the ability of coral to grow in certain metocean conditions by merging data from multiple sources. Success was defined as having an accuracy of greater than 80%, as found when evaluating the R Squared value.

There are multiple real-world applications of this project. Oceanographic researchers can use the predictions in order to plan coral discovery and monitoring expeditions. Additionally, conservation groups can predict the likelihood of coral damage, given changes in local and global climate.

In the future, a higher resolution dataset could improve the accuracy of the model further. Including more variables (dissolved, CO2, ocean current speed, pH, presence of other marine life) would increase understanding of factors impacting coral growth. Finally, deeper netwrok architectures may improve the accuracy of the model, while still using the present datasets.

0.1.20 15. References

- https://www.fisheries.noaa.gov/national/habitat-conservation/deep-sea-coral-habitat
- https://www.kaggle.com/noaa/deep-sea-corals
- https://www.ncei.noaa.gov/data/oceans/woa/WOA18/DATA/