Report for FYS-STK4155: Project 3

Laurent Fontaine, Maziar Kosarifar, Maksym Brilenkov and Knut Hauge Engvik

December 2019

Abstract

In this project, we have developed...

Contents

1 Introduction

In science, each discovery starts with the question and Machine Learning (ML) is not an exception. Therefore, this project was dedicated to end-to-end data analysis, where we decided to describe, use and process the freshwater microbial ecology data sampled from 72 lakes of southern Norway and Sweden as a part of COMSAT project [**?**] [link for COMSAT project?].

The dataset composed of two tables - bacterial *amplicon sequence variance* (ASV) and *environmental metadata* - and overall contains more than 15000 columns, with 72 data points each. The main advantage here is that both tables are matched by observation; thus, both can be used as either input or an output. However, the actual amount of data points limits us in terms of predictive algorithms we can use in order to achieve high accuracy scores.

Overall, the dataset seemed to be an excellent candidate from both scientific and educational point of views. Hence, as our research questions, we decided to (1) identify patterns of variation in bacterial community composition along environmental gradients, (2) predict environmental metadata variables from bacterial community composition and (3) predict presence or absence of bacterial ASVs from environmental metadata.

After careful speculation about the nature of the data set, we presumed that *Random Forest Classifier* together with *Extreme Gradient Boosting* (XGBoost) algorithm are viable methods to address these tasks. Among other benefits, the methods greatly reduce the overfitting, and can help highlight the most relevant features of the areas under study. Moreover, we have also implemented the so-called *Siamese Neural Network* (SNN) described by *Koch at el* [**?**] [Koch, Siamese Neural Networks for One-shot Image Recognition] using the combination of manual coding and `Keras`[1] - the *Deep Learning* (DP) Library developed for fast and easy Neural Network (NN) implementation. In addition, we also used codes for *Feed Forward Neural Network* (FFNN) developed in project 2 [**?**] [Laurent? Everybody?] and implemented `Keras` analog to compare the results with Random Forests and XGBoost.

The report organized as follows. In section 2, we briefly introduce the concepts of Random Forests, XGBoost and SNN, and how to work with them. In section 3, we explained the implementation of before mentioned algorithms and the key features of the code we have developed. The main results of the pipeline run are presented in section 4. Section 5 is reserved for discussion. The codes are listed in the Appendix.

[1] https://keras.io/

2 Methods and algorithms

In this section, we briefly describe the theory behind approaches we have chosen to analyze the before mentioned data set (for more deep description, please refer to [**?**] [lecture notes, previous reports]).

2.1 Decision tree

Decision Trees are the main building blocks for *Random Forests* and are essentially the supervised learning algorithm, which can be used for both classification and regression tasks and which uses a tree-like graph structure to make respective predictions. Hence, the term "tree".

As in the real trees, every decision tree starts with the *root node*, which then grows into *interior nodes* (the test on the attribute) with the final *leaf nodes* (the class label), which are connected by the so-called *branches* (the outcome of the test) [**?**] [Morten, lectures]. The entire path from root to leaf is the classification rule.

The simplified version of training the tree is as follows [**?**] [Morten, lecute slides]:

- Present a dataset containing of a number of training instances characterized by a number of descriptive features and a target feature;
- Train the decision tree model by continuously splitting the target feature along the values of the descriptive features using a measure of information gain during the training process;
- Grow the tree until we accomplish a stopping criteria create leaf nodes which represent the predictions we want to make for new query instances;
- Show query instances to the tree and run down the tree until we arrive at leaf nodes.

2.1.1 Regression tree

To grow the *regression* tree we first split the predictor space, x_1, x_2, ..., x_p into J distinct non-overlapping regions, R_1, R_2, ...,R_J (which are usually represented by high-dimensional rectangles for simplicity), where we make the same prediction for each observation which falls into the same region [**?**] [Morten, Random Forests]. Therefore, out goal is to find such R_1, R_2, ..., R_J, which minimize the MSE

$$\text{MSE} = \sum_{j=1}^{J} \sum_{i} \left(y_i - \bar{y}_{R_j}\right)^2, \tag{2.1}$$

where $\bar{y}_{R_j}$ is the mean response for the training observations within the box j.

It is not possible, however, to consider all possible combinations to partition feature space into the boxes; thus, we are using the so-called *top-down* approach [**?**]: starting at the top of the tree, we will split the predictor space into two new brunches, look into the best result between the two and then continue splitting again further down the tree (see [**?**] for more details).

This approach is straightforward, but leads often to overfitting and complicated trees. To avoid this, we are using procedure called *pruning* [**?**] [Random Forests]: once the tree T_0 has grown, we remove several sub-nodes in order to obtain a sub-tree. Going even further, we can consider the sequence of trees (defined by non-negative tuning parameter, α), instead of every possible sub-tree. This process is called *cost complexity tuning* and mathematically means to make

$$\sum_{m=1}^{\bar{T}} \sum_{i} \left(y_i - \bar{y}_{R_m}\right)^2 + \alpha \bar{T}, \tag{2.2}$$

as low a possible. Here $\bar{T}$ is the number of terminal nodes of the tree T, R_m is the the subset of predictor space corresponding to the m-th terminal node.

When $\alpha = 0$, the sub-tree T will simply equal T_0, but, as α increases, the pruning appears to be in a nested and predictable fashion; thus, resulting in an easily obtainable sequence of sub-trees, as a function of [**?**] [Random Forests]. The value of α can be selected via cross-validation.

2.1.2 Classification tree

To grow the *classification* tree we use similar approach as in case of regression. However, instead of MSE used for binary splitting we are now using *classification error rate* [**?**] [Morten, Random Forests]

$$p_{mk} = \frac{1}{N_m} \sum_{i} I\left(y_i \neq k\right) = 1 - p_{mk}, \tag{2.3}$$

where we define PDF, p_{mk}, to represent the number of observations of a class k in a region R_m with N_m observations.

Alternatively, we can also use either *gini index*

$$g = \sum_{k=1}^{K} p_{mk}\left(1 - p_{mk}\right), \tag{2.4}$$

or *information entropy*

$$s = -\sum_{k=1}^{K} p_{mk} \log p_{mk}, \tag{2.5}$$

as they are more sensitive to node purity.

To set up of decision tree, we are using the *Classification and Regression* (CART) algorithm [] [Morten], which splits the data set into two subsets via single feature k and some threshold value t_k. It then tries to minimize the *cost function* (see e.g. [**?**] [lecutre sabout gradient algorithms] for more details)

$$C(k, t_k) = \frac{m_{\text{left}}}{m} G_{\text{left}} + \frac{m_{\text{right}}}{m} G_{\text{right}}, \tag{2.6}$$

in case of classification. Here $G_{\text{left/right}}$ measures the impurity of the left/right subset and $m_{\text{left/right}}$ is the number of instances in the left/right subset.

In case of regression, the cost function is [**?**] [Random Forest]

$$C(k, t_k) = \frac{m_{\text{left}}}{m} \text{MSE}_{\text{left}} + \frac{m_{\text{right}}}{m} \text{MSE}_{\text{right}}, \tag{2.7}$$

with

$$\text{MSE}_{\text{node}} = \frac{1}{m_{node}} \sum_i \left(\bar{y}_{\text{node}} - y_i\right)^2, \quad \bar{y}_{\text{node}} = \frac{1}{m_{node}} \sum_{i \in \text{node}}, \tag{2.8}$$

2.1.3 Pros and Cons

Overall, there are a lot of advantages to use decision trees [] [Morten, Random Forests]:

- No feature normalization needed;
- Can handle both continuous and categorical data;
- Can model nonlinear relationships;
- Can model interactions between the different descriptive features;
- Can be displayed graphically, and are easily interpreted;

However, there are also disadvantages [] [Morten]:

- Generally do not have the same level of predictive accuracy as some other regression and classification approaches;
- If continuous features are used the tree may become quite large and hence less interpratable;
- Are prone to overfit the training data and hence do not well generalize the data;

- Small changes in the data may lead to a completely different tree;
- Unbalanced datasets may lead to biased trees;
- Features with many levels may be preferred over features with less levels.

In order to increase the predictive performance of trees, the simple approach is to stuck together many decision trees, using methods like random forests and boosting.

2.2 Random Forest Classifier

As it was mentioned above, the *Random Forest* is a model based on aggregating many decision trees. The name "random" stems from the two main concepts [**?**] [Random Forests]:

- Random sampling of data points - we build a number of decision trees on bootstrapped training samples. In this way, although each individual tree may have high variance, the entire forest will have lower variance (without bias increase). At test time, predictions are made by averaging the predictions of each decision tree.
- Only a subset of all predictors, m, is considered for splitting each node in each decision tree from the full set of predictors, p. Usually, it is set to [**?**] [Random Forests]: $m \approx \sqrt{p}$ for classification.

Random forest combines hundreds or thousands of decision trees, trains each one on a different set of the observations and then makes a final prediction based on averaging the predictions on each individual tree. The overall algorithm can be described as follows [**?**] [Morten?]:

1. For $b = 1$ to B:
   a) Draw a bootstrap sample Z^* of size N from the training data;
   b) Grow a random-forest tree T_b to the bootstrapped data, by recursively repeating the following steps for each terminal node of the tree, until the minimum node size n_{min} is reached:
      i. Select m variables at random from the p variables;
      ii. Pick the best variable/split-point among the m;
      iii. Split the node into two daughter nodes;
2. Output the ensemble of trees $\{T_b\}_1^B$.

2.3 Extreme Gradient Boosting

Gradient Boosting (GBoost) is an algorithm, based on a decision tree approach. The key difference here is that it combines weak classifiers to create a good classifier via series of iterations [**?**] [Morten, Random Forests]. Mathematically speaking, this means that we define a cost function [**?**] [Morten, Random Forests]:

$$C(f) = \sum_{i=0}^{n-1} L\big(y_i, f(x_i)\big), \quad \left(\text{e.g., squared-error function: } \sum_{i=0}^{n-1} \big(y_i - f(x_i)\big)^2\right), \tag{2.9}$$

where $y_i$is our target, $f(x_i)$ is a function to model it, and then try to minimize it using the following algorithm [**?**] [Morten, Random Forest]:

1. Initialize the zero estimate of $f_0(x)$;
2. For $m = 0$ to M, compute:
   a) compute the negative gradient vector $\mathbf{u}_m = -\partial C(\mathbf{y}, \mathbf{f})/\partial \mathbf{f}(x)$ at $f(x) = f_{m-1}(x)$;
   b) fit the so-called *base-learner* to the negative gradient $h_m(u_m, x)$;
   c) update the estimate $f_m(x) = f_{m-1}(x) + \nu h_m(u_m, x)$;
3. Compute the final estimate: $f_M(x) = \sum_{m=1}^{M} \nu h_m(u_m, x)$.

The work "extreme" implies usage of modern multiprocessing algorithm to make the gradient descent as efficient as possible. All this combines into XGBoost library [**?**] [Tianqi Chen, Carlos Guestrin,XGBoost: A Scalable Tree Boosting System], which is highly scalable, efficient, flexible and portable[**?**] [Morten, Random Forests].

2.4 Siamese Neural Networks

In project 2, we have already described in a detail the FFNN (see, e.g. [**?**]). In this project, we are using the more advanced approached based on the hybrid NN - *Siamese Neural Network* (SNN) - which is used when faced with a classification problem and small datasets.

SNN have shown good results when employed in image recognition ([**?**] [Hadsell, FaceNet]). The basic premise of these networks is that the feature space is mapped to some metric space where the images of input vectors can be compared. Vectors whose images are “close” in the given metric can be thought of as similar. The goal then, is to train the network to map the different categories to different “areas” of the metric space and, in a sense, learn the concept of sameness.

The name "siamese" stems from the structuring of the network. It consists of a base network, which can have any structure, that acts as a function mapping input vectors into an

N dimensional space, where N is the number of output nodes for the base network. For a twin network, two (three for triplets) instances of this base network are created with shared weights and biases. The network can thus take two (or three) different vectors as input and the conjoined base networks will output a corresponding number of N dimensional representations. Then follows a merging layer where the distances between the images of the input vectors are calculated. This output is then fed to a layer that outputs a similarity score between each vector.

One method to train a twin network is to first pick one representative, referred to as an anchor, from each class. Then, for each anchor, a_i, and datapoint, x_j, the pair, (a_i,x_j), is created. If the pair belongs to the same category they are labeled 1 and if they belong to different categories the pair receives the label 0. These pairs, along with their labels, will then serve as training data for the network. The network is then updated according to a loss function that pushes inputs from different classes apart while clustering similar inputs. Hadsell *et al* [**?**] proposes the *Contrastive Loss* function:

$$y(\text{dist}(a,x)^2 + (1-y)(\max(m - \text{dist}(a,x),0))^2, \tag{2.10}$$

which is basically a combination of two different loss functions. If the label $y = 1$, then the distance is minimized, if $y = 0$, it is maximized up to a margin m.

For triplet networks, the method is somewhat similar. For each data point x_i, pick one representative from the same category, a_s, and one representative from a different category, a_d. Then, update the shared network according to the *Triplet Loss* function from `FaceNet`:

$$\max(\text{dist}\,(a_s,x_i)^2 - \text{dist}\,(a_d,x_i)^2 + m, 0)\,, \tag{2.11}$$

Again, m is a margin beyond which the network will stop updating.

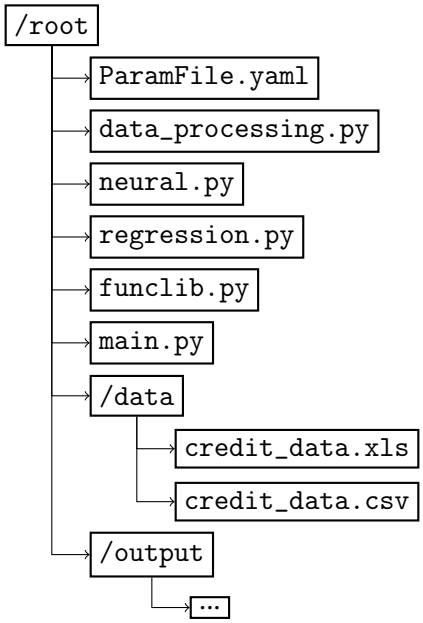
Sources: Hadsell et al 2006 http://yann.lecun.com/exdb/publis/pdf/hadsell-chopra-lecun-06.pdf FaceNet https://arxiv.org/abs/1503.03832v3

3 Algorithms implementation

In this section, we describe the pipelines written to, first, process the data sets and, second, to apply techniques described in previous section.

3.1 File Organisation

Overall, the program structured as follows:



- `root` - root folder, contains all the files;
- `data` - folder which contains all data used in the current project;
- `output` - the folder which contains all output (e.g. plots) of the resulting fitting etc.;
- `ParamFile.yaml` - parameter file used to configure neural network;
- `data_processing.py` - the module used to process the Credit Card data and Franke function;
- `neural.py` - module to instantiate and apply all techniques necessary for Neural Network to run successfully;
- `regression.py` - module to run linear and logistic regression analysis;
- `funclib.py` - module which contains all functions used in the program;

- `main.py` (`main.ipynb`) - the entry point of the program (`jupyter` notebook for nicer output).

Before running the program, choose your preferred configuration inside the parameter file. You can decide on type of the task (Classification or Regression), the input data and output folder, the number of hidden layers, type of activation function for hidden layers and the output layer, number of neurons in hidden layer and output layer (for classification it should be 2, for regression 1), number of epochs to loop over, regularisation parameter and learning rate and so on.

Run the program either with main.py or main.ipynb. The latter will give you nicer view on results as it supports the markup.

3.2 Data Preparation

[either here or inside results]

3.3 Random Forests

[part of the code corresponding to random forests]

3.4 XGBoost

[code corresponding to XGBoost]

3.5 Neural Networks

[Here is to talk about Feed Forward Neural Net with Keras and Siamese Neural Network]

4 Results

In this section, we discuss the data set ...

4.1 Data exploration

Chosen dataset. This study was performed on freshwater microbial ecology data generated by sampling 72 lakes from southern Norway and Sweden (fig. 4.1). It was part of a project designated COMSAT. The dataset comprises two tables. One consists of counts of amplicon sequence variants (ASV) for bacteria, while the other contains environmental metadata (table 4.2). Each observation in either table corresponds to a lake. Both tables are matched by observation and can thus be used as input and output for each other. Bacterial ASVs can be treated as a proxy for the abundance of bacterial species. These ASV counts can also be converted to binary with 0 equal to 0 and values above 0 set to 1 in order to study presence/absence patterns.

Figure 4.1: Freshwater lakes from southern Norway and Sweden sampled for the COMSAT project. Secchi depth is displayed to provide a general impression of the longitudinal gradient in the dataset.

Site	ASV1	ASV2	ASV3	ASV4	ASV5	...	Latitude	Longitude	Altitude	Area	Depth	Temperature	Secchi	O2	CH4	pH	TIC	SiO2	KdPAR
10000_Hurdalsjøen	18464	5231	6963	7563	9516	...	60.37648	11.04077	176	32.81	20.0	17.03	6.50	0.9044194	11.797343	6.870	0.82230	3312	0.62
10001_Harestuvatnet	15296	58728	30659	1614	17059	...	60.19323	10.71212	234	1.98	13.0	15.85	4.50	0.8468347	72.674567	7.365	4.05800	3783	0.89
170B_Gjersjøen	13356	52215	25810	1367	14586	...	59.78970	10.77485	40	2.64	22.0	19.65	3.30	0.8131012	52.953904	7.685	8.08500	3563	0.95
170_Gjersjøen	16227	53747	26456	2823	3119	...	59.78970	10.77485	40	2.64	22.0	19.65	3.30	0.8131012	52.953904	7.685	8.08500	3563	0.95
180_Øgderen	52862	4887	1361	14854	25616	...	59.71388	11.41303	133	12.66	9.5	18.61	1.10	0.8406025	85.639780	7.225	2.66800	1125	1.60
189_Krøderen	18830	53461	50015	12664	13253	...	60.13485	9.75860	133	43.91	14.0	15.44	2.80	0.8582522	29.100059	6.695	0.81360	2499	0.82
191_Rødbyvatnet	43828	7657	1836	34800	20517	...	59.58175	10.48715	118	1.16	10.0	18.55	2.10	0.8527711	260.596931	7.535	3.09200	2063	1.32
214_Gjesåssjøen	10532	588	9275	19181	5315	...	60.68167	11.99235	176	3.98	3.5	19.63	1.15	0.8360833	97.561306	7.070	1.73200	2924	2.27
2252_Rotnessjøen	14088	39265	35086	11061	7228	...	60.49690	12.34120	260	1.09	26.0	16.55	1.95	0.7350632	41.956068	6.635	0.77310	5559	1.08

Figure 4.2: Subsets of the ASV and metadata tables. The columns to the right show the first 5 ASVs in decreasing order of abundance while the columns to the left show linearly independent metadata variables that can be treated as explanatory to bacterial community composition.

Research questions. It was chosen in this study to (1) identify patterns of variation in bacterial community composition along environmental gradients, (2) predict environmental metadata variables from bacterial community composition and (3) predict presence or absence of bacterial ASVs from environmental metadata. The first objective was pursued using pairs of dissolved organic matter water content as input with corresponding pairwise bacterial community Bray-Curtis distances as output. The second objective used the full bacterial community composition – ASV – table as input and single environmental metadata variables as outputs, yielding a separate model for each predicted metadata variable. The third objective used a subset of environmental variables as input to predict a subset of the bacterial community composition table converted to presence/absence values.

4.2 Data processing

Data filtering, formatting and transformation. The dataset contained only continuous variables. There were missing values in the environmental metadata, which were replaced by intrapolation using Multivariate Imputation by Chained Equations (Buuren Groothuis-Oudshoorn 2011). This decision to keep observations with missing values is motivated by the very small number of observations in the full dataset compared to the number of descriptors which would make it even harder to train models successfully, were the number of observations to be reduced any further. The ASV table was scaled using ranging (set values

to interval of 0-1) due to this transformation yielding somewhat better clustering in terms of ecological meaningfulness compared to subtraction of mean and division by standard deviation (Legendre Legendre 2012). Environmental metadata were used with or without scaling, the former being performed by subtracting the mean and dividing by standard deviation.

Dimensionality reduction. In order to extract patterns from the ASV data without using all predictors, pairwise distances between observations were used. Since the counts for most ASVs across most observations is 0, it was necessary to use an asymmetrical coefficient in order to avoid inflated similarity between observations. The Bray-Curtis distance was used accordingly. In the case of the environmental metadata, groups of linearly dependent variables had all but one variable kept, while a subsequent pruning was performed in the same manner where groups of variables presented variance inflation factors (VIF) above a threshold of 20 ([] [ter Braak Smilauer 2002]). Correlations among environmental variables were visualized using a heatmap (fig. 4.3).

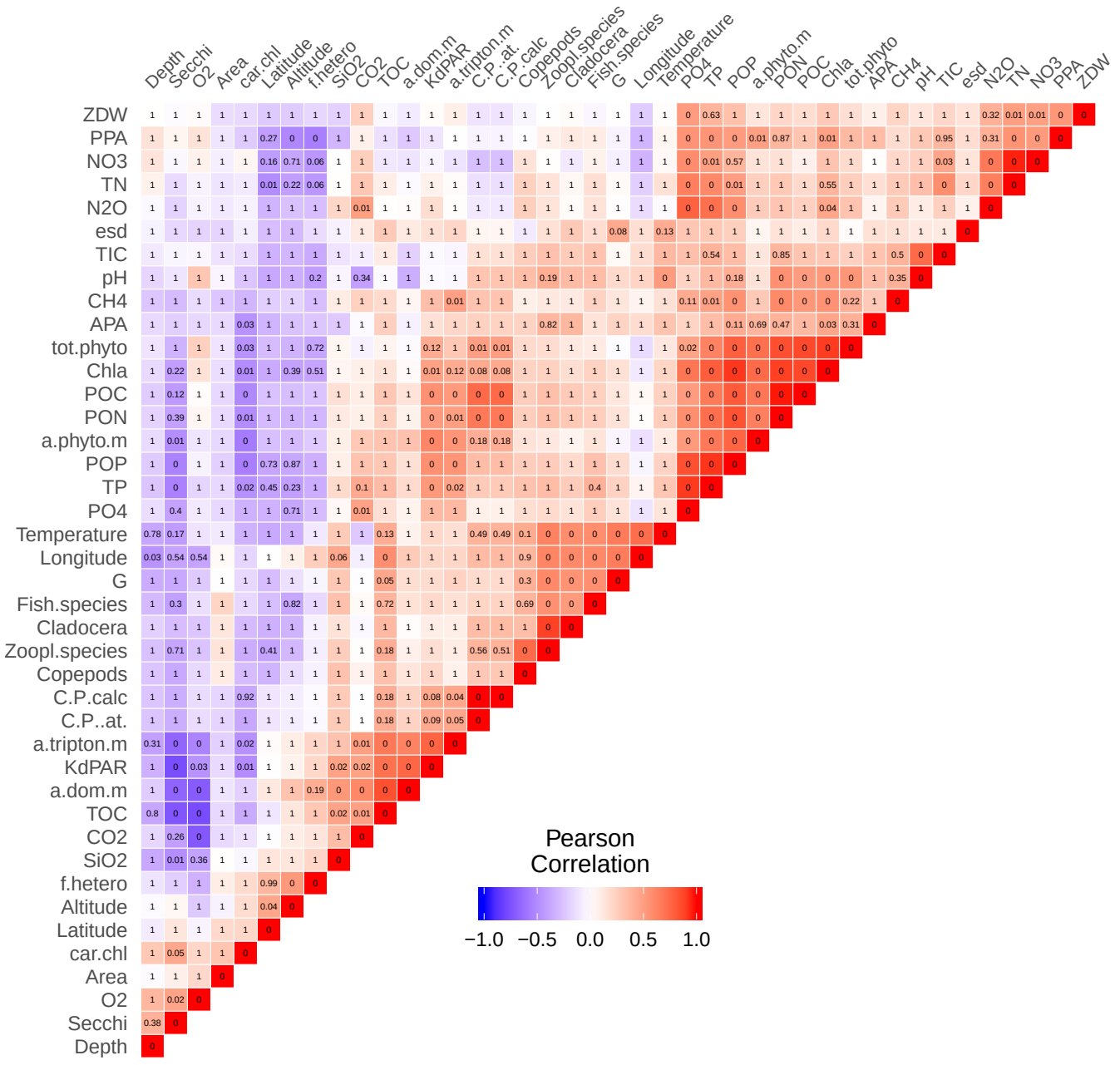


Figure 4.3: Pearson correlations among environmental metadata variables. Correlation values are displayed by color; the values printed in tiles indicate p-values for correlation significance.

4.3 XGBoost

Regression using `scikit-learn`'s implementation of XGboost was used to predict bacterial community Bray-Curtis distances along the dissolved organic matter (DOM) gradient ("`a.dom.m`" in the environmental metadata table). The model was trained on data split into 80% training and 20% test sets. The same analysis was performed with own code for a feed forward neural network. After training the respective models, response data was generated for a meshgrid of values matching the minimum-maximum range of the DOM gradient. The idea here was to see how well XGboost performed for generating a model used for *intrapolation*. Intrapolated outputs were also generated with ordinary least-squares regression to appreciate how well an XGboost model can contain complex structures in the data where linear models cannot.

The patterns in both XGboost and project 2 FFNN [**?**] [Laurent report 2] predicted values are similar. XGboost is however much faster when training the model. The linear model poorly captured the complex structures in data revealed by the XGboost and neural network regressions (fig. 4.4). While the predicted values from the XGboost model seem correct on the grid of values from the original dataset, the model returns some type of step function when predicting values for a meshgrid containing (x,y) values not present in the training dataset.

SV2	ASV3	ASV4	ASV5	...	Latitude	Longitude	Altitude	Area	Depth	Temperature	Secchi	O2	CH4
231	6963	7563	9516	...	60.37648	11.04077	176	32.81	20.0	17.03	6.50	0.9044194	11.797343
3728	30659	1614	17059	...	60.19323	10.71212	234	1.98	13.0	15.85	4.50	0.8468347	72.674567
2215	25810	1367	14586	...	59.78970	10.77485	40	2.64	22.0	19.65	3.30	0.8131012	52.953904
3747	26456	2823	3119	...	59.78970	10.77485	40	2.64	22.0	19.65	3.30	0.8131012	52.953904
887	1361	14854	25616	...	59.71388	11.41303	133	12.66	9.5	18.61	1.10	0.8406025	85.639780
3461	50015	12664	13253	...	60.13485	9.75860	133	43.91	14.0	15.44	2.80	0.8582522	29.100059
657	1836	34800	20517	...	59.58175	10.48715	118	1.16	10.0	18.55	2.10	0.8527711	260.596931
588	9275	19181	5315	...	60.68167	11.99235	176	3.98	3.5	19.63	1.15	0.8360833	97.561306
9265	35086	11061	7228	...	60.49690	12.34120	260	1.09	26.0	16.55	1.95	0.7350632	41.956068

Figure 4.4: Bray-Curtis community composition distances by DOM gradient. a) Bacterial community composition (BCC) Bray-Curtis distances by DOM (raw data). b) XGboost-Predicted BCC Bray distances by sites' DOM. c) OLS (SVD) regression-predicted BCC Bray distances by sites' DOM, DOM 0.01 step meshgrid. d) NN-smoothed XGboost-predicted BCC Bray distances by sites' DOM, DOM 0.01 step meshgrid. e) NN-predicted BCC Bray distances by sites' DOM, DOM 0.01 step meshgrid. f) XGboost-predicted BCC Bray distances by sites' DOM, DOM 0.01 step meshgrid.

4.4 Random Forest

[Maziar]

We have tried to eliminate the arguments with high correlation rates, and so the input arguments should be rather independent, but since there are only 72 observations available, which should be divided between the training and testing dataset, using Random Forest Classifier would be the best choice.

We have tried to use the physical features of each location to predict the existence or lack of certain microbiome, to find the most important feature. We also attempted to predict certain features of each location based on the concentration of microbiomes in that location.

Based on the result, it seems that Random Forest Classification is better at predicting the existence or lack of microbiomes given the physical features measured in that location.

4.5 Neural Networks

4.5.1 Siamese NN

[Knut]

For the siamese network approach, the results were, on the whole, disappointing. While the networks certainly showed signs of learning, getting good results on the training data, they displayed little to no ability to generalize. Attempts to alleviate overfitting by adding dropout layers or adjusting hyper parameters, did not lead to improvements on predictive ability beyond the level of random guessing.

With our choice of data set, it is not obvious whether this failure is due to the methods employed or a lack of significant correlation between the predictors, microbial community composition, and the various target variables such as phosphate concentrations or temperature at the sample sites.

4.5.2 FFNN

[maksym, Keras implementation?]

...

5 Conclusions

References

[1] Buuren, S. van, Groothuis-Oudshoorn, K. (2011). mice: Multivariate Imputation by Chained Equations inR. Journal of Statistical Software, 45(3). https://doi.org/10.18637/jss.v045.i03

[2] Legendre, P., Legendre, L. F. (2012). Numerical ecology(Vol. 24). Elsevier

[3] Hadsell et al 2006 http://yann.lecun.com/exdb/publis/pdf/hadsell-chopra-lecun-06.pdf

[4] FaceNet https://arxiv.org/abs/1503.03832v3

[5]