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

As the last project for Machine Learning class, we decided to ...

Chosen dataset:

The report organized as follows. In section **??**, I briefly sintroduce the concept of NN and how to train them. In section 3, I explain how I implemented the algorithms and also who the results of the pipeline run. The codes are listed in the Appendix.

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

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

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

[Maziar]

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 XGBoost

[Laurent]

2.4 Siamese Neural Networks

[from Knut]

One strategy to levy the flexibility and power of neural networks when faced with a classification problem and small data sets, is to make a so called siamese network. Such networks have shown good results when employed in image recognition (ref 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 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 network 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 each datapoint x_j pair (a_i,x_j) is created. If the pair belong 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:

$$y(\mathrm{dist}(a,x)^2 + (1-y)(\max(m-\mathrm{dist}(a,x),0))^2\,, \tag{2.9}$$

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

The method is somewhat similar for triplet networks. For each data point x_i, pick one representative from the same category, a_s, and one representative from a different category,

a_d. Update the shared network according to the triplet loss function from FaceNet Triplet loss :

$$\max(\text{dist}\,(a_s, x_i)^2 – \text{dist}\,(a_d, x_i)^2 + m, 0)\,, \tag{2.10}$$

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

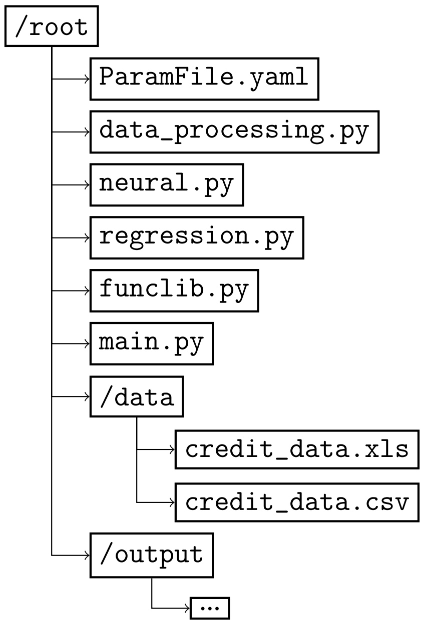
Sources: Hadsell et al 2006 http://yann.lecun.com/exdb/publis/pdf/hadsell-chopra-lecun-06.pdf FaceNet https://arxiv.org/abs/1503.03832v3

3 Algorithms implementation

In this section, I 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's 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 processing

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.

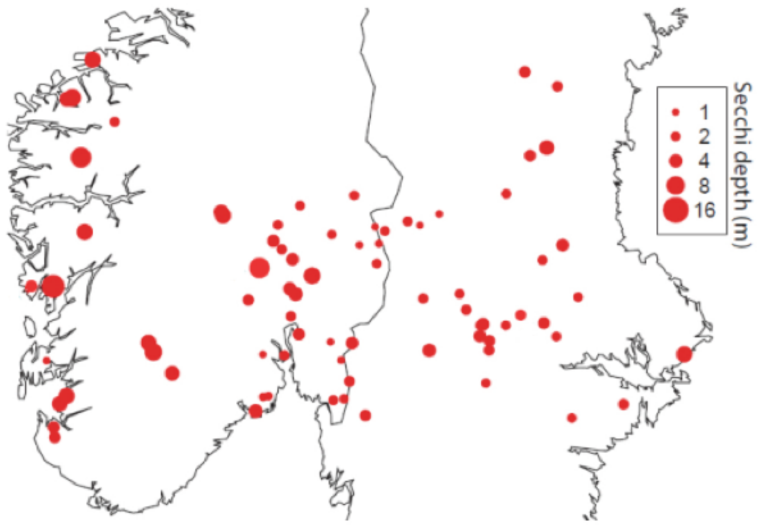
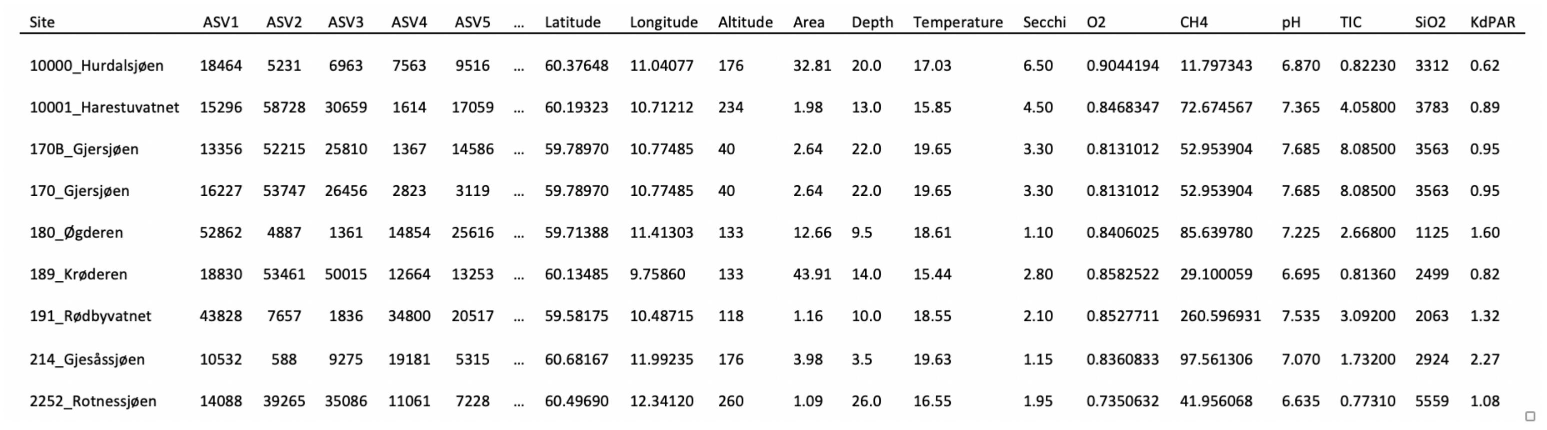


Figure 4.1: Freshwater lakes from southern Norway and Sweden sampled for the COMSAT project. Secchi depth is displayed to provide a general impression of the longitudinal gradient in the dataset.



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.

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.

4.2 Random Forest

[Maziar]

Based on the type of the problem we are trying to solve, we presumed that the Random Forest Classifier is a viable method to do so. Among other benefits of this method, it minimises the problem of overfitting, and can help highlight the most relevant features of the areas under study.

The algorithm used for Random Forest Classifier is as follows:

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.

4.3 XGBoost

[Laurent]

...

4.4 Siamese

[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.

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]