LU TP 19-03
January 2019

Bachelor's Thesis

Analysing Raman spectra of crystalline cellulose degradation by fungi using artificial neural networks

Theoretical Physics

Author:
Sebastian Hutteri

Supervisor:
Carl Troein

LUND
UNIVERSITY

Abstract

This thesis investigates the use of artificial neural networks for classifying Raman spectra of partially degraded cellulose samples by fungal species. A multilayer perceptron configuration of 4 hidden layers and 128 hidden nodes was able to classify a set of 60 samples with an overall prediction accuracy of 0.55.

Results show that data resolution is an important factor when optimizing classifier performance, and that a resolution of 1.0 cm^{-1} gave the highest performance.

We found that choosing suitable parameters for the asymmetric least squares smoothing (ALSS) correction is of relevance when attempting to optimize classifier performance, and that an ALSS smoothness value of $\lambda = 10^5$ gave the highest performance.

Results also indicate that some fungal species and control treatments have stronger signatures in certain spectral regions. *Gloeophyllum sp.*, *Coprinellus angulatus* and NaOH treatments had the most accurate probability distribution and may therefore be considered to cause the most unique cellulose modification.

This thesis shows promising results for artificial neural networks to be utilized for classifying Raman spectra of partially degraded cellulose samples.

Popular science description

We are surrounded by fungi. They are in the soil, on our skin and even in the air we breathe. There are millions of different fungal species on Earth, but what do they do? In nature, some fungi play an important role as decomposers, replenishing their environment with nutrients by breaking down organic matter. These "saprotrophic fungi" are capable of obtaining nutrients and energy from organic matter, such as leaves, seeds, stems, logs, roots, etc. and each of these species may even have their own unique set of mechanisms for doing this. Studies have shown that brown-rot and white-rot fungi do indeed differ in their methods of breaking down cellulose, but are these methods unique enough to tell them apart from other fungi? Is it possible that every fungal species has its own signature? We don't know.

Figure 1: White rot in a birch tree. Sten Porse, license: CC-BY 3.0.

Raman spectroscopy, a technique commonly used for analysing and identifying materials, can provide spectral representations of cellulose samples which have been partially broken down by fungi. Such representations, which are associated with the molecular properties of the degraded samples, could potentially be viewed as "fungal fingerprints". However, Raman spectroscopy can produce large amounts of data which may be too complex, or too time consuming, for humans to analyse meticulously. In such situations, artificial neural networks can be utilized.

Artificial neural networks can be described as computational models, vaguely inspired by the human brain, capable of acquiring and maintaining information. These networks can easily be provided the spectral data of the cellulose samples which they will learn to associate with their corresponding species, all by themselves. If training is successful, a network will be able to correctly identify a sample it has never seen before. Artificial neural networks may therefore provide a tool powerful enough to further help us understand the similarities and differences between fungal species, as well as their mechanisms.

Acknowledgements

I would like to sincerely thank Carl Troein and Dimitrios Floudas for their support and enthusiasm during this project.

Contents

1 Introduction

1.1 Cellulose and saprotrophic fungi

Cellulose is the most abundant organic polymer on earth and is widely recognized for its industrial use. While chemically simple, cellulose is intermolecularly complex [1] and a deeper understanding of the crystalline structure within the plant cell wall could be beneficial for several industrial applications. In nature, saprotrophic fungi play a significant role as decomposers of organic matter, but the mechanisms they employ to decompose cellulose are not well-understood. By improving our understanding of the fungal decay of cellulose, we can learn new methods of breaking down cellulose for biofuel production [2] as well as develop new ways to prevent rot and decay in wood [3].

Two related types of saprotrophic fungi capable of decaying wood are the species belonging to white-rot and brown-rot fungi. Studies have shown that they differ in their mechanisms of breaking down cellulose, yet we still know very little about the strategies they employ, especially those of brown-rot fungi [1]. We know even less about litter-decomposing fungi, which are harder to examine due to their heterogeneous environment, even though they seem to share physiological traits with white-rot fungi [1].

Spectral analysis of partially decomposed cellulose from fungi of different decay types could provide additional information about these fungal mechanisms and help us further understand the structure of cellulose, as well as the evolution of nutritional strategies used by fungi.

1.2 Raman spectroscopy

Raman spectroscopy is a technique commonly used for analysing and identifying materials. It relies on the energy shifts of photons as they interact with the molecular vibrations of a sample material and scatter inelastically. The frequencies of these vibrations depend on the mass, arrangement and bond strength of the atoms in the molecule. Raman spectroscopy can therefore provide a spectral representation of the molecular structure of a material [4].

Raman spectra are typically represented as the intensity of the inelastically scattered light with respect to its frequency (wavenumber).

1.3 Neural networks

Artificial neural networks can be described as computational models, vaguely inspired by the human brain, capable of acquiring and maintaining information. They are used in several areas of application such as function approximation, pattern recognition and prediction [5].

The artificial neuron can be seen as the processing unit of the neural network and operates by collecting incoming signals $x_1, x_2, ..., x_n$ and multiplying them by their corresponding synaptic weights $w_1, w_2, ..., w_n$. The sum of the weighted signals is passed through an activation function f, resulting in an output signal y which is forwarded to the next layer of neurons [5]. A model of the artificial neuron is shown in Fig. 2. It is also common that the neuron contains an *activation bias*, which acts as a threshold for the weighted sum before being passed through the activation function [5]. Such a bias is not used during this project. Throughout this paper, the terms "neuron" and "node" are used interchangeably.

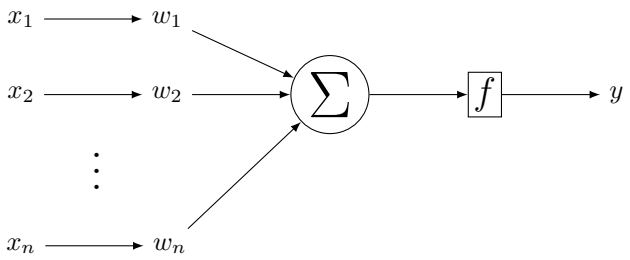

Figure 2: Model of an artificial neuron. Incoming signals $x_1, x_2, ..., x_n$ are multiplied by their corresponding synaptic weight $w_1, w_2, ..., w_n$ and summed together. The weighted sum is then passed through an activation function f, resulting in an output signal y.

The following two expressions describe the mathematical operation of the artificial neuron [5].

$$u = \sum_{i=1}^{n} w_i \cdot x_i \tag{1}$$

$$y = f(u) \tag{2}$$

The most basic neural network model is the single-layer perceptron, and some of its most relevant components, including those already mentioned, are listed below.

- *Synaptic weights* controlling the significance of forwarded signals.
- *Neural layers* making up the network structure (input, output and hidden layers).
- *Neurons or nodes* collecting the incoming weighted signals.
- *Activation functions* modulating the incoming signals within the nodes and adding non-linearity.
- *Error function* calculating the error between produced output and desired output.
- *Optimization algorithm* adjusting synaptic weights to minimize the error.

While the single-layer perceptron consists only of one neural layer, one can construct a more complex network model, capable of mapping any nonlinear continuous function [6], by adding more layers. The multilayer perceptron, as shown in Fig. 3, is considered one of the most versatile network models function [6] and is the model we used during this project.

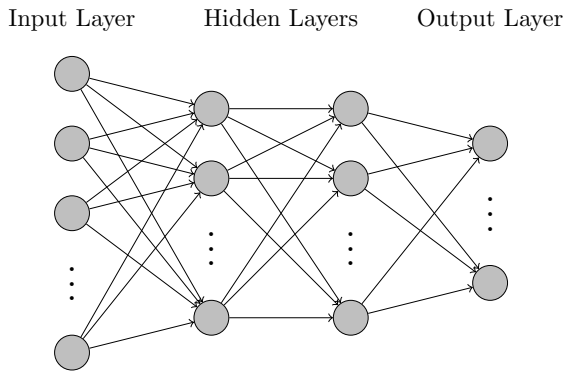

Figure 3: General model of a feed-forward neural network. Neurons (circles) are connected between layers via synaptic weights (arrows).

The synaptic weights within the multilayer perceptron are adjusted by the process of *supervised training*, which requires every data input to have a desired output [6]. During this type of training, the weights are updated by an optimization algorithm in order to minimize the error between the produced output and the desired output.

Optimization algorithms used for multilayer perceptrons are often variations of the *backpropagation* algorithm. This algorithm calculates the gradient of the output error, with respect to the weights within the network, and adjusts the weights in the opposite direction of the gradient. This process, when repeated, results in a gradual error reduction until a desired error value has been reached or until a selected number of iterations, or *epochs*, have been completed [6].

Variations of the backpropagation algorithm are often implemented to prevent the output error from converging to a local minima.

When minimizing the output error, there is a risk of allowing the network to become too closely fitted to the training data. This condition is called *overfitting* and makes it difficult for the network to generalize and make predictions for new data. Overfitting can be prevented by different regularization techniques [6].

1.4 Cellulose samples

Experiments by our collaborator Dr. Dimitrios Floudas at the Department of Biology at Lund University have produced spatially resolved Raman spectra of isolated cellulose that has been partially decomposed by different species of fungi and several control treatments, see Fig. 4. The data consist of 60 separate files from measurements of 16 different cellulose samples within the wavenumber region 1200-1550 cm^{-1}, with a resolution of approximately 2 measurements per cm^{-1}. Most samples were measured three times at different regions within the sample, with 66 spectra included in every file. However, some samples were measured seven times at different regions with 18 spectra included in every file. Each data file represents a complete measurement at a unique cellulose sample region. Out of the samples, nine represent fungal species while the rest are control samples, as shown in Table 1.

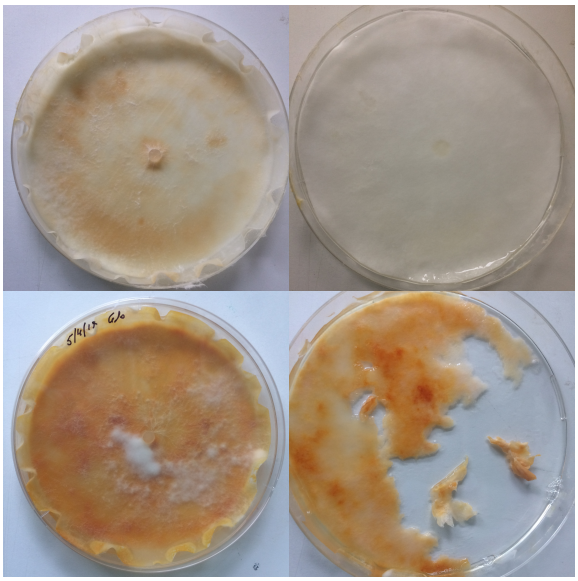

Figure 4: Images of cellulose, partially degraded by *Phanerochaete laevis* (top) and *Gloeophyllum sp.* (bottom), after 40 days of incubation. Images show before (left) and after (right) a majority of the surface fungal mycelium has been removed.

Cellulose samples

Fungal species	Abbreviation	Spectra
Agrocybe pediades (LD)	AGP	3 x 66
Tricholomella constricta (LD)	CAC	3 x 66
Coprinellus angulatus (LD)	C	3 x 66
Gymnopus confluens (LD)	GYM	3 x 66
Gloeophyllum sp. (BR)	G	3 x 66
Leucoagaricus leucothites (LD)	LEPSP	3 x 66
Phanerochaete laevis (WR)	PHALA	3 x 66
Psilocybe cf. subviscida (LD)	PSS	3 x 66
Tetrapyrgos nigripes (LD)	TEN	3 x 66
Control		
Autoclaved paper	AUP	3 x 66
Non-inoculated paper autoclaved at 20°C	CON20	3 x 66
Non-inoculated paper autoclaved at 25°C	CON25	3 x 66
24 hour enzyme treatment (CellicCTec2)	EnzC2_24h3	7 x 18
48 hour enzyme treatment (CellicCTec2)	EnzC2_48h2	7 x 18
3 hour NaOH3(3 M) treatment	NaOH3M	7 x 18
Non-inoculated and non-autoclaved paper	OP	3 x 66

Table 1: Table of cellulose samples, abbreviations and number of spectra. Fungal decay type is also denoted: litter decomposers (LD), brown-rot (BR) and white-rot (WR).

2 Methods

All of the code was written in Python 3 and is included in the Appendix.

2.1 Data processing

The available Raman data suffered from irregular spacing between data points as well as inconsistent wavenumber values between spectra. In order to extract equivalent data points from every spectrum, the data was approximated in Python using quadratic polynomial interpolation, which also allowed us to adjust the resolution of the data provided to the neural network. However, a resolution of 0.67 cm^{-1} was used initially. We inspected the interpolation by plotting the interpolated data on top of the raw Raman data, as shown in Fig. 5.

The approximated data was also corrected in Python using asymmetric least squares smoothing (ALSS) [7] to account for baseline variations among spectra. The ALSS algorithm has two parameters, asymmetry p and smoothness λ, which both have to be tuned for the data at hand. The asymmetry parameter, which affects the distribution of the differences between spectrum and baseline, was set to $p = 10^{-3}$, since this value has been shown to work well for correcting Raman spectra with positive peaks [7]. The smoothness parameter, which affects the intensity relation between approximated spectral peaks and baseline as seen in Fig. 6, generally works well for $10^2 < \lambda < 10^9$ when correcting spectra with positive peaks [7]. Initially, we set this value to $\lambda = 10^4$ with the intention of tuning the parameter after the neural network had been constructed. Fig. 7 shows a comparison between raw and ALSS corrected Raman spectra.

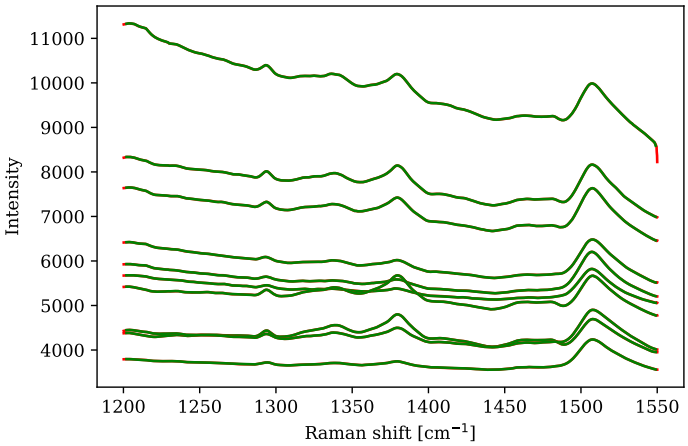

Figure 5: Example of first ten Raman spectra of *Gloeophyllum sp.* with interpolated data (green) on top of raw data (red).

As seen in Fig. 5, the first spectra generally have the highest baseline intensity, which then decreases for the following spectra during the measurement.

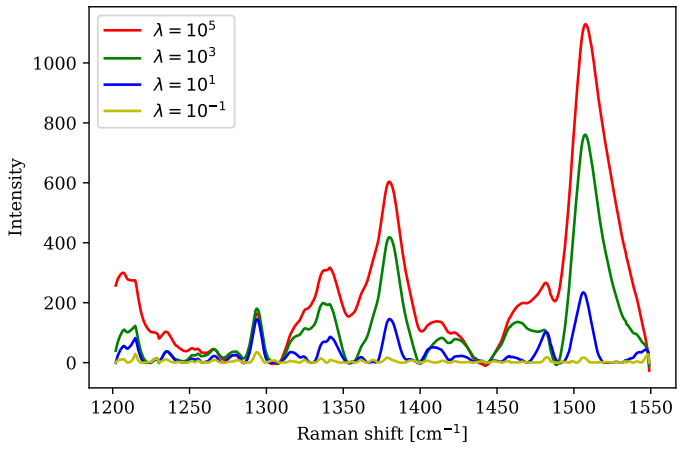

Figure 6: Comparison between Raman data corrected with different values for the ALSS smoothness parameter. The graphs show the first spectrum of *Gloeophyllum sp.* with ALSS asymmetry $p = 10^{-3}$. The input data resolution is 0.67 cm^{-1}.

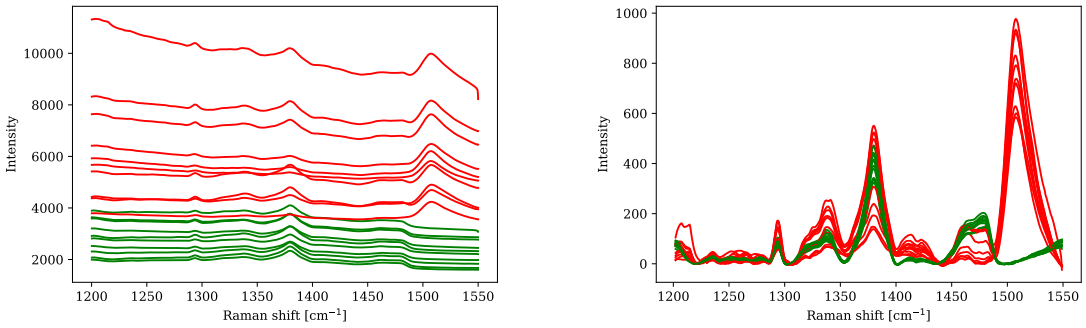

Figure 7: Comparison between raw (left) and ALSS corrected (right) Raman data. The graphs show the first ten spectra of *Gloeophyllum sp.* (red) and *Tetrapyrgos nigripes* (green) with ALSS asymmetry $p = 10^{-3}$ and ALSS smoothness $\lambda = 10^4$. The input data resolution is 0.67 cm^{-1}.

2.2 Constructing neural networks

The artificial neural networks applied in this project were constructed in Python using the deep learning library Keras [8], since this library enables fast implementation and experimentation. Our goal was to find a network configuration based on the multilayer perceptron model, capable of multi-class classification with high performance and with moderate computational cost. The different types of fungal species and control treatments will, from now on, be referred to as *classes*.

2.2.1 Model selection and hyperparameters

Initially, the number of nodes within the input and output layer were adjusted for data compatibility and for giving an output matrix of 16-dimensional vectors, with each dimension representing a specific class.

The ReLU function (rectified linear unit) was applied as the activation function for all hidden layers, including the input layer, since this function has been shown to work well for classification purposes [9]. The ReLU function is defined as $f(x) = max(0, x)$.

The softmax function was applied as the activation function for the output layer in order to give a probabilistic output distribution. The softmax function is defined as $f(y_i) = \frac{e^{y_i}}{\sum_j e^{y_j}}$.

Adam (adaptive moment estimation) [10] was applied as the optimization algorithm with a learning rate of 10^{-4} and the categorical cross entropy function was applied as error function. The categorical cross entropy function is defined as $f(p, q) = -\sum_x \log(q(x))$.

The number of epochs was set to 100.

2.2.2 Training and evaluation

Networks were trained and evaluated separately for every data file with the following procedure:

We removed a single data file from the data set with the intention of re-introducing it for classification after the network had been trained on the remaining data. The network was also provided a matrix of unit vectors, representing the desired output of the training data. When training was

completed, the isolated file was re-introduced for classification and the network generated a probabilistic prediction distribution among the 16 classes for every spectrum within the file. A final and normalized probability distribution was calculated from the class with the highest probability of every spectrum within the file.

The procedure was repeated for every data file and the overall fraction of the probability distributions being in the correct classes was interpreted as *classifier performance*, and given as a value between 0 and 1. Fig. 12 in the appendix shows a heatmap matrix of the probability distribution for the different classes, where the rows show the actual class and the columns show the network's predictions. Since there are multiple data files representing the same class, the heatmap matrix show the average probability distribution for every class. The classifier performance can be calculated from the average value of the diagonal entries of the heatmap matrix.

2.2.3 Hidden layers and nodes

In order to find a satisfactory network model, we systematically evaluated the performance using different network configurations. The effects on classifier performance by the number of hidden layers and hidden nodes within the network were investigated, as shown in Fig. 8. For the main part of the investigation, a network configuration of 4 hidden layers and 128 hidden nodes per layer was used.

2.3 Further investigations

We wanted to see if the ALSS correction could affect the performance of the network classifier. The network was therefore provided with multiple sets of data, corrected with different values for the ALSS smoothness parameter, as shown in Fig. 9. Classifier performance was also evaluated for different values of input wavenumber resolution, as shown in Fig. 10. The resolution regulates the number of equidistant data points provided to the network.

We also investigated the classifier performance when providing the network with data from three isolated spectral regions: 1250-1301 cm^{-1}, 1301-1445 cm^{-1} and 1445-1491 cm^{-1}. The two outer regions have been shown to contain signatures typical for amorphous cellulose [11]. Classifier performance for these three regions, as well as for all of them, are shown in Fig. 11 and Table 2.

3 Results

3.1 Layers and hidden nodes

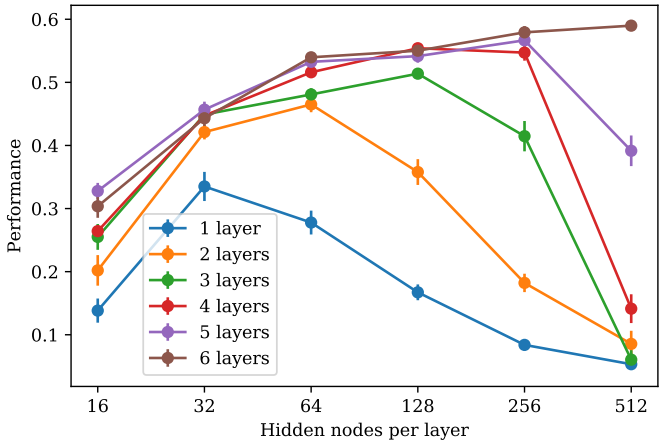

Figure 8: Mean classifier performance with respect to number of hidden layers and nodes with standard error bars. Within every configuration, an equal number of nodes is used for every hidden layer and the input data resolution is 0.67 cm^{-1} for the region 1202-1549 cm^{-1}. Each configuration is trained and evaluated five times with 100 epochs.

As seen in Fig. 8, there is a pattern in the classifier performance when adding nodes to a network with a fixed number of layers and epochs. The peak in performance is reached at an increasingly higher number of nodes as the number of hidden layers increases. The computational cost of adding layers to the network increases exponentially since they are added geometrically. However, the performance plateaus and decreases. A configuration of 4 hidden layers and 128 hidden nodes per layer was chosen for the following investigations, since such a configuration yields a high performance at a moderate computational cost.

3.2 ALSS smoothness

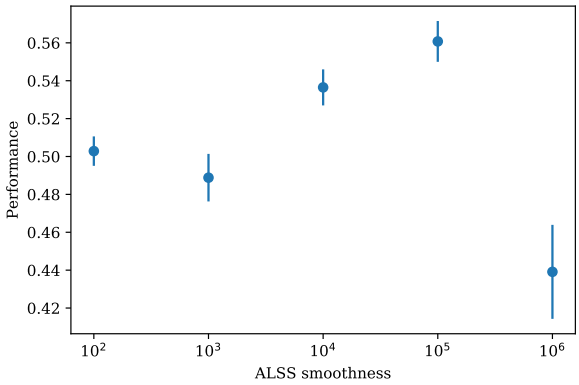

Figure 9: Mean classifier performance with respect to ALSS smoothness with standard error bars. The networks are trained and evaluated five times with 4 hidden layers and 128 hidden nodes per layer.. The input data resolution is 0.67 cm^{-1} for the region 1202-1549 cm^{-1}.

As seen in Fig. 9, the classifier performance is the highest for an ALSS smoothness value of 10^5. However, a value of 10^4 was chosen for the following investigations, since we did not want to modify the spectra too much.

3.3 Data resolution

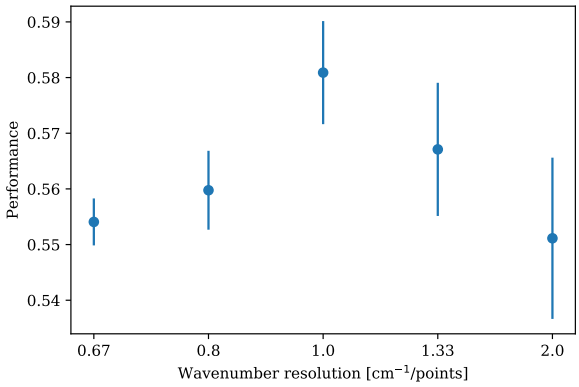

Figure 10: Mean classifier performance with respect to wavenumber resolution for the region 1202-1549 cm^{-1} with standard error bars. The networks are trained and evaluated five times with 4 hidden layers and 128 hidden nodes per layer.

Interestingly, a wavenumber resolution of 1.0 cm^{-1}, as seen in Fig. 10, produces the highest classifier performance with a network configuration of 4 hidden layers and 128 hidden nodes. The standard error decreases with higher resolution.

3.4 Spectral regions

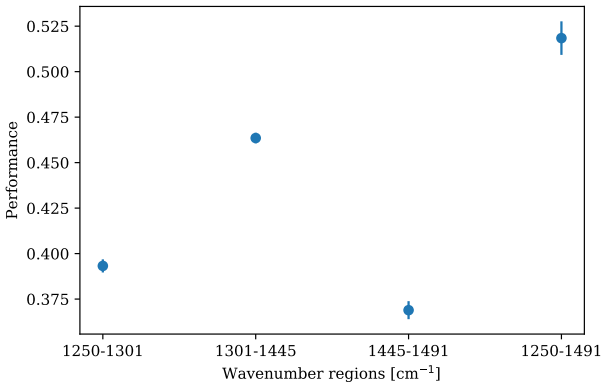

Figure 11: Mean classifier performance for spectral regions with standard error bars. The networks are trained and evaluated five times with 4 hidden layers and 128 hidden nodes per layer. Input data resolution is 1.5 points per cm^{-1}.

Out of the three single regions, overall classifier performance is the highest for region 1301-1445 cm^{-1}, as seen in Fig. 11. However, the best performance is achieved when using all three regions together.

Classifier performance				
	Region [cm^{-1}]			
Class	1250-1301	1301-1445	1445-1491	1250-1491
AGP	0.42	0.53	0.27	0.56
CAC	0.16	0.11	0.25	0.22
C	0.37	0.87	0.55	0.80
GYM	0.24	0.26	0.22	0.28
G	0.71	0.78	0.90	0.91
LEPSP	0.37	0.31	0.30	0.47
PHALA	0.16	0.34	0.058	0.47
PSS	0.48	0.025	0.32	0.023
TEN	0.56	0.47	0.20	0.50
AUP	0.52	0.85	0.59	0.85
CON20	0.063	0.077	0.18	0.34
CON25	0.38	0.59	0.30	0.73
EnzC2_24h3	0.37	0.43	0.22	0.44
EnzC2_48h2	0.26	0.40	0.28	0.25
NaOH3M	0.98	0.99	0.79	0.96
OP	0.26	0.39	0.49	0.51

Table 2: Mean classifier performance for specific fungal species and control treatments, with network trained and evaluated five times on isolated spectral regions with 4 hidden layers and 128 hidden nodes. Input data resolution is 0.67 points per cm^{-1}.

As seen in Table 2, most classes have a varying classifier performance between the spectral regions and the performance for all three regions often correspond to the single region with the highest performance. However, some classes, such as *Gloeophyllum sp.* and NaOH, are easy to classify in all regions.

4 Discussion

The general approach for finding a suitable network model was trying different values of network hyperparameters and evaluating classifier performance. Unfortunately, hyperparameters may not be independent of each other and changing one parameter might therefore impact the significance of another. On the other hand, trying all combinations of hyperparameters would have taken too long. This is the reason why some of the parameters was set to a fixed value during the entire project, for example the number of epochs or the type of activation functions. Another approach could have been Bayesian optimization of hyperparameters.

Quadratic polynomial interpolation seems to work well for approximating irregular Raman data. The interpolation resolution had a significant impact on the classifier performance and the highest performance was, interestingly, not achieved when using the highest resolution. This could be due to the simplicity of the network and a more complex network could perhaps perform better with a higher resolution. Another explanation could be that having multiple Raman data points within a small interval does not add additional information about the sample, compared to having a single data point.

The smoothness parameter of the ALSS algorithm also had an impact on the classifier performance. Higher values for the smoothness parameter can help minimize errors and flaws from the measurement and enhance significant spectral signatures, but it can also eliminate the bio-chemical properties of the data, some of which could be of importance when attempting to differentiate between fungal species with similar spectral signatures. A value of $\lambda = 10^4$ seems suitable for this project, but a more complex network might perform equally well with a lower value. Surprisingly, a value of $\lambda = 10^2$ resulted in a fairly high classifier performance. The asymmetry parameter could potentially also affect the classifier performance, but no other value than $p = 10^{-3}$ was tested.

Increasing the number of layers for the network allowed for a higher classifier performance when finding a suitable number of nodes in every layer. However, the increase in performance was not linear with respect to the number of layers and the computational cost increased exponentially. A network configuration of 4 hidden layers and 128 hidden nodes therefore seemed appropriate for this project. However, with more time and greater computational resources, highly complex networks could be tested.

Network overfitting is often prevented by using regularization techniques, such as *early stopping* [6]. However, our approach for preventing overfitting was to limit the number of epochs to 100. A limited number of epochs could potentially prevent networks from reaching maximum performance, and further investigations could therefore be done using a different number of epochs and/or regularization techniques.

The middle spectral region gave the best performance out of the three isolated regions, when averaging the performance over all classes. This could be explained by the fact that this region was the widest. Another explanation could be that the middle region contains most of the spectral signatures associated with the fungal species and the control treatments.

Class specific performance shows that G (BR) was easy to classify in all three regions, while PHALA (WR) was easier to classify in the middle region. Other fungal species (LD) did not have clear similarities in which region they were the easiest to classify.

The approach for separating training and validation data was to select entire data files for training, i.e. not splitting data from the same region within the sample. Another approach could be to merge the data from all files and then randomly separating the training data from the validation data.

5 Conclusions

A multilayer perceptron configuration of 4 hidden layers and 128 hidden nodes was able to classify a set of 60 cellulose samples with an overall prediction accuracy of 0.55. This value can be interpreted as how certain the networks are that the overall classification is correct. Since 8 of the 9 fungal species had the highest probability fraction in the correct classes, using artificial neural networks for classifying cellulose samples degraded by fungi seems to be a promising method. Greater accuracy can most likely be achieved by further optimization of network architecture and hyperparameters.

However, the goal of the project was not to find a network configuration with a prediction accuracy of 1.0, since this would have made it difficult to analyse the results for similarities or differences between species. Additionally, some of the control samples are almost identical, for example the non-inoculated samples or the enzyme treatments, and these samples can therefore be expected to be difficult for the networks to classify correctly.

Species that are more easily classified correctly by the networks could be considered to cause the most unique cellulose modification. Similarly, species that are harder for the networks to classify

correctly could be considered to modify cellulose in ways similar to other species.

When looking at the heatmap matrix (Fig. 12 in Appendix), one can see that the networks managed to classify most of the samples correctly, with 11 out of the 16 classes having the highest probability fraction in the correct classes. G (BR), C (LD) and NaOH3M had the most accurate probability distribution and may therefore be considered to cause the most unique cellulose modification.

It was also found that PSS (LD) was incorrectly classified as CAC (LD) and GYM (LD). The associations between these three classes could also be seen when the network tried to classify CAC and GYM.

References

[1] Petr Baldrian. Chapter 2 enzymes of saprotrophic basidiomycetes. *British Mycological Society Symposia Series*, 28:19–41, 12 2008.

[2] Valdeir Arantes and Jack Saddler. Access to cellulose limits the efficiency of enzymatic hydrolysis: The role of amorphogenesis. *Biotechnology for biofuels*, 3:4, 02 2010.

[3] Barbara L Illman. Oxidative degradation of wood by brown-rot fungi. *Active Oxygen/Oxidative Stress and Plant Metabolism. American Society of Plant Physiologists, Rockville, MD*, pages 97–196, 1991.

[4] Peter J. Larkin. Chapter 1 - introduction: Infrared and raman spectroscopy. In Peter J. Larkin, editor, *Infrared and Raman Spectroscopy (Second Edition)*, pages 1 – 5. Elsevier, second edition edition, 2018.

[5] Ivan Nunes da Silva, Danilo Hernane Spatti, Rogerio Andrade Flauzino, Luisa Helena Bartocci Liboni, and Silas Franco dos Reis Alves. *Introduction*, pages 3–19. Springer International Publishing, Cham, 2017.

[6] Ivan Nunes da Silva, Danilo Hernane Spatti, Rogerio Andrade Flauzino, Luisa Helena Bartocci Liboni, and Silas Franco dos Reis Alves. *Multilayer Perceptron Networks*, pages 55–115. Springer International Publishing, Cham, 2017.

[7] Paul HC Eilers and Hans FM Boelens. Baseline correction with asymmetric least squares smoothing. *Leiden University Medical Centre Report*, 1(1):5, 2005.

[8] François Chollet et al. Keras. https://keras.io, 2015.

[9] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[10] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 12 2014.

[11] Steffen Fischer, Karla Schenzel, Klaus Fischer, and Wulf Diepenbrock. Applications of ft raman spectroscopy and micro spectroscopy characterizing cellulose and cellulosic biomaterials. *Macromolecular Symposia*, 223:41 – 56, 03 2005.

6 Appendix

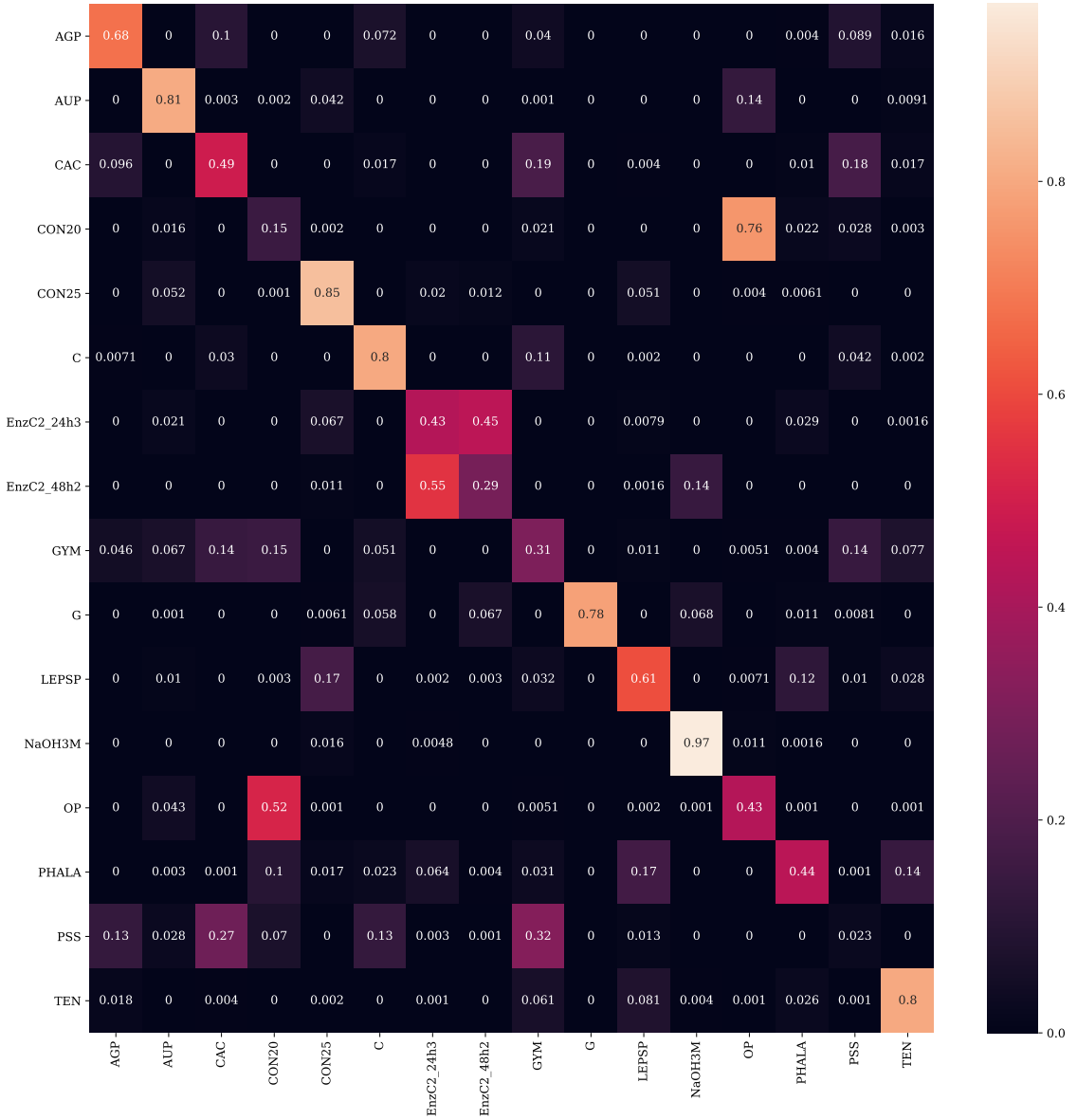

Figure 12: Heatmap matrix showing multi-class classification results. Rows show the actual class and columns show the neural network's probabilistic prediction. The networks are trained and evaluated five times with 4 hidden layers and 128 hidden nodes per layer. Input data resolution is 1.5 points per cm^{-1} for the region 1202 - 1549 cm^{-1}.

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from scipy.interpolate import interp1d
import os
import scipy as sp
from IPython.display import display
from ipywidgets import FloatProgress
import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation
from keras.optimizers import SGD, Adam, RMSprop
from sklearn.model_selection import train_test_split
import seaborn as sns
import datetime

class Classifier:
    def __init__(self, samplefolder, wavenum, smoothness, assymmetry, nruns, layers, nodes, act, lr, opt, loss, epochs):
        self.samplefolder = samplefolder
        self.wavenum = wavenum
        self.smoothness = smoothness
        self.assymmetry = assymmetry
        self.nruns = nruns
        self.layers = layers
        self.nodes = nodes
        self.act = act
        self.lr = lr
        self.opt = opt
        self.loss = loss
        self.epochs = epochs
        self.NSessions = [3, 3, 3, 3, 3, 3, 7, 7, 3, 3, 3, 7, 3, 3, 3, 3] #Number of sessions per species.
        self.NSpecies = len(self.NSessions) #Number of different species
        NSpectra = [66, 66, 66, 66, 66, 66, 18, 18, 66, 66, 66, 18, 66, 66, 66, 66] #Number of spectra per session.
        Identity = np.identity(self.NSpecies) #One Hot matrix with vectors for every species.
        Y = []
        for i in range(self.NSpecies):
            for j in range(self.NSessions[i]):
                Z = []
                for k in range(NSpectra[i]):
                    Z.append(list(Identity[i]))
                Y.append(Z)
        self.Y = Y
        self.SpeciesNames = ['AGP', 'AUP', 'CAC', 'CON20',
                             'CON25', 'C', 'EnzC2_24h3', 'EnzC2_48h2',
                             'GYM', 'G', 'LEPSP', 'NaOH3M',
                             'OP', 'PHALA', 'PSS', 'TEN'] #Labels for heatmap.

    def ALSS(self, y, niter=10,): #Normalization and baseline correction.
        L = len(y)
        D = sp.sparse.csc_matrix(np.diff(np.eye(L), 2))
        w = np.ones(L)
        for i in range(niter):
            W = sp.sparse.spdiags(w, 0, L, L)
            Z = W + self.smoothness * D.dot(D.transpose())
            z = sp.sparse.linalg.spsolve(Z, w * y)
            w = self.assymmetry * (y > z) + (1 - self.assymmetry) * (y < z)
        return z

    def FitData(self, fileindex): #Interpolation fitting.
        ALSSData = []
        Data = pd.read_csv('{}/{}'.format(self.samplefolder, fileindex), sep=';', header=None)
        for i in range(1, len(Data.columns)):
            f = interp1d(Data[0], Data[i], kind='quadratic')
            y1 = f(self.wavenum)
            y2 = self.ALSS(y1)
            y = y1 - y2
            ALSSData.append(y)
        return ALSSData

    def ProcessData(self):
        fp = FloatProgress(min=0,max=len(os.listdir(self.samplefolder)))
        display(fp)
        DataMatrix = []
        print('Storing data in DataMatrix...')
        for i in sorted(os.listdir(self.samplefolder)):
            DataMatrix.append(self.FitData(i))
            fp.value += 1
        self.DataMatrix = DataMatrix
        print('Data stored.')

    def XTrain(self, index): #Slicing data into training and validation sets.
        return np.concatenate(self.DataMatrix[:index] + self.DataMatrix[index + 1:])
    def YTrain(self, index):
        return np.concatenate(self.Y[:index] + self.Y[index + 1:])
    def XTest(self, index):
        return np.array(self.DataMatrix[index])
    def YTest(self, index):
        return np.array(self.Y[index])
```

```
    def ANN(self, index): #Constructing ANN.
        model = Sequential()
        model.add(Dense(self.nodes, activation=self.act, input_dim=len(self.wavenum)))
        for i in range(self.layers):
            model.add(Dense(self.nodes, activation=self.act))
        model.add(Dense(self.NSpecies, activation='softmax'))
        model.compile(loss=self.loss, optimizer=self.opt, metrics=['accuracy'])
        model.fit(self.XTrain(index), self.YTrain(index), epochs=self.epochs, batch_size=128)
        predict = model.predict(self.XTest(index)) #Predicts class by index for every spectra.
        predictclass = np.argmax(predict, axis=1) #Takes the max prediction of every spectra.
        predictcount = np.zeros(self.NSpecies) #Array of zeros.
        for i in predictclass:
            predictcount[i] += 1 #Every max prediction increases its species index value by 1.
        return predictcount/sum(predictcount) #Normalized.

    def SingleRun(self, run): #Single run of training and evaluating network.
        Pred = []
        for i in range(len(os.listdir(self.samplefolder))):
            Pred.append(self.ANN(i))
        Pred = np.array(Pred)
        Now = datetime.datetime.now()
        Stamp = ' {}{}{} {}.{}.{}'.format(Now.year, Now.month, Now.day, Now.hour, Now.minute, Now.second) #Time stamp.
        np.savetxt('{}/{}'.format(self.FolderStamp, 'Pred' + Stamp), Pred) #Saving data.
        Accuracy = []
        for i in range(self.NSpecies):
            Accuracy.append(sum(Pred[sum(self.NSessions[:i]):sum(self.NSessions[:i]) + self.NSessions[i]])/self.NSessions[i])
        Accuracy = np.array(Accuracy)
        AccuracyData = pd.DataFrame(Accuracy, columns=self.SpeciesNames, index=self.SpeciesNames) #Average accuracy for species.
        np.savetxt('{}/{}'.format(self.FolderStamp, 'Acc {}'.format(run)), Accuracy)
        plt.figure(figsize=(self.NSpecies,self.NSpecies))
        sns.heatmap(AccuracyData, annot=True)
        plt.savefig('{}/{}'.format(self.FolderStamp, 'Acc {}'.format(run) + '.png'))
        Perf = []
        for i in range(self.NSpecies):
            Perf.append(Accuracy[i][i])
        return sum(Perf)/self.NSpecies

    def Run(self): #Running the single run multiple times.
        Now = datetime.datetime.now()
        FolderStamp = 'Run {}{}{} {}.{}.{}'.format(Now.year, Now.month, Now.day, Now.hour, Now.minute, Now.second) #Time stamp.
        self.FolderStamp = FolderStamp
        os.makedirs(self.FolderStamp)
        parameters = 'WaveNumMin={}, WaveNumMax={}, WaveNumValues={}, ALSSSmoothness={}, ALSSAssymmetry={}, HiddenLayers={},
Nodes={}, Activation={}, LearningRate={}, Optimization={}, Loss={}, Epochs={}'.format(min(self.wavenum), max(self.wavenum),
len(self.wavenum), self.smoothness, self.assymmetry, self.layers, self.nodes, self.act, self.lr, self.opt, self.loss, self.epochs)
        p = open(self.FolderStamp + '/Parameters.txt', 'w+')
        p.write(parameters)
        p.close()
        fp = FloatProgress(min=0,max=self.nruns)
        display(fp)
        print('Running network...')
        PerfList = []
        for i in range(self.nruns):
            PerfList.append(self.SingleRun(i + 1))
            fp.value += 1
        np.savetxt(self.FolderStamp + '/Performance', PerfList)
        return PerfList

#HOW TO RUN CODE
#Pipeline = Classifier(SampleFolder, WaveNum, Smoothness, Assymmetry, NRuns, Layers, Nodes, Activation, LearningRate, Optimization, Loss,
Epochs)
#Pipeline.ProcessData()
#Pipeline.Run()
```