IMPERIAL UNIVERSITY OF HOKKAIDO




**MASTER THESIS**
(5 March 2018 - 11 June 2018)

*Author*

**Aurélien PÉLISSIER**
**Master 2 - Nanophysics**

# Feature Selection as Reinforcement Learning Applied to Raman spectra for cancer diagnosis

**Supervisor : Atsuyoshi NAKAMURA**


HOKKAIDO UNIVERSITY, SAPPORO, JAPAN

Sapporo, June 2018

# Abstract

## Feature Selection as Reinforcement Learning applied to Raman spectra for cancer diagnosis

Over the past decades, there has been a very active research in the development of Raman spectroscopy techniques for oncological applications, and there is currently an interest at finding the most relevant wavenumbers in the Raman spectra for disease identification. To this end, information theory is used to study correlation between the wavenumbers, and high redundancy between many wavenumbers is observed. A hierarchical clustering algorithm is then trained to classify the wavenumbers in different information clusters. Furthermore, feature selection methods are applied to Raman spectra to find the most informative wavenumbers for diseases diagnosis, and two different feature selection approaches based on reinforcement learning and bandit strategies are presented: Greedy-SR and FUSE-2 ; both algorithms are tested on various dataset and proves to be remarkably more efficient than major filtering approaches. Using a feature set evaluation based on a $k$ nearest neighbors classifier, we find that 5 wavenumbers are enough to diagnosis follicular thyroid cancer with 98 % accuracy.

***Acknowledgement***

*I would like to sincerely thank Atsuyoshi Nakamura, my thesis advisor, for receiving me in his team during these 3 months of my internship as well as for his guidance and support through my approaches, the project would not have been possible without his continuing involvement and input. I also thank Osaka group and Koji Tabata who provided the Raman data along with detailed explanations. Finally I express my gratitude to all my lab-mates who never hesitated to make themself available when help was needed.*

Joseph Ding
B3 Student

Yuki KOJO
B4 Student

Shumpei Tsubakino
M2 Student

Mariko Tai
B4 Student

Tomonori Mizuuchi
B4 Student

Naoki Ito
B4 Student

# Outline :

## Notations

| | |
|---|---|
| $\mathcal{F}$ | Feature set |
| $F \subset \mathcal{F}$ | Feature subset |
| $\mathcal{L}$ | Training set |
| $\mathcal{V} \subset \mathcal{L}$ | Small subset of $\mathcal{L}$ |
| $f \in \mathcal{F}$ | Feature in the feature set |
| $d = \|F\|$ | Number of features in the feature subset |
| $n = \|\mathcal{L}\|$ | Number of examples in the training set |
| $m = \|\mathcal{V}\|$ | Number of examples in $\mathcal{V}$ |
| $k$ | Number of nearest neighbors |
| | |
| $\mathcal{T}$ | Feature lattice |
| $N_F \in \mathcal{T}$ | Node corresponding to feature subset $F$ |
| | |
| FS | Feature Selection |
| MAB | Multi Armed Bandit |
| ACC | Accuracy |
| AUC | Area Under ROC Curve |
| $k$-NN | $k$ Nearest Neighbours |
| HAC | Hierarchical Agglomerative Clustering |

# Introduction

Raman spectroscopy is a powerful characterization technique that provides information on molecular vibrations and crystal structures, it is non destructive and has relatively high spatial resolution, which makes it widely used in physics, chemistry and biology. Over the past decades, there has been a very active research in the development on Raman techniques for oncological applications [1], such as for example the early detection of pre-malignant lesions. Particularly, there is a considerable clinical requirement for a noninvasive real-time Raman probe [2] that can perform accurate and repeatable measurement of pathological state.

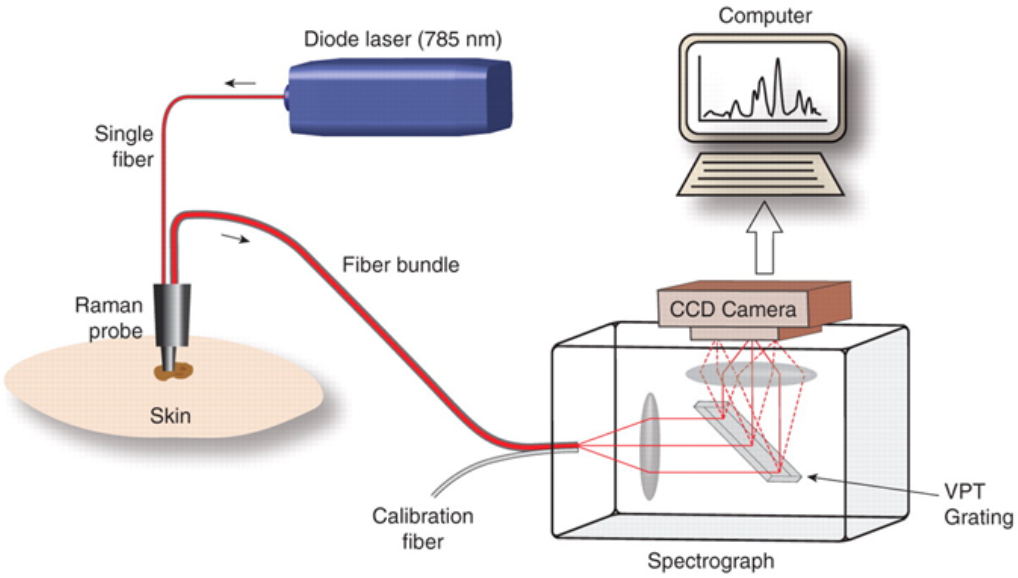


**Figure 1:** In vivo Raman spectrometer system for cancer diagnosis [2].

Machine learning has proven notably useful regarding clinical applications, because it can provide highly accurate disease diagnosis as well as personalized treatment (e.g. Google DeepMind Health [3]), which makes the combination of Raman spectroscopy with machine learning particularly successful. There is currently an interest at finding the most relevant wavenumbers in the Raman spectra for disease diagnosis because it could speed up Raman measurements [4] and simplify algorithms training, thus making the pathological diagnosis faster. To this end, a feature selection methods has to be applied to Raman spectra to find the most informative wavenumbers for diseases diagnosis.

The aim of this work is not only to perform wavenumber selection on Raman spectra, but also to develop novel general methods for feature selection. We summarize in Chapter I some basic notions in computer science and more specifically machine learning so that non specialist can understand the work presented in this thesis. In Chapter II, we present two different feature selection approaches based on Reinforcement learning and bandit strategies. The first method, known as Greedy, starts from the empty feature set and repeatedly add the best additional feature to the set until no additional feature further improves the set evaluation. The other method, known as FUSE, uses a Monte Carlo tree search with bandit strategy to look for the optimal feature set. Finally, Chapter III is dedicated to the Raman spectroscopy applied to living cells, and focus on the wavenumbers selection for follicular thyroid cancer diagnosis.

# I Preamble for non specialist

### Computational complexity

In theoretical computer science, the computational complexity refers to the quantity of resources needed (generally time or storage) by an algorithm to be executed. As the amount of needed resources varies with the input, the complexity is given as a function of the input size $n$, and commonly written as $O(f(n))$, which correspond to the asymptotic quantity of resources needed as a function of its input size. Computational complexity theory allows classification of computational problems according to their inherent difficulty, which makes it a key aspect in algorithm analysis.

As an example, the security of our online transactions (RSA cryptosystem) rests on the assumption that factoring integers with a thousand or more digits is practically impossible. It is because the most efficient classical factoring algorithm currently known (General number field sieve) has an exponential asymptotic runtime to the number of digits $d$: $O(e^{d^{1/3}})$. However in 1995, Shor introduced a quantum algorithm [5] (today known as the Shor's algorithm), which has an asymptotic runtime polynomial in $d$: $O(d^3)$, making quantum computers a threat to our current transaction systems.

### Fixed Budget vs Fixed Confidence

When it is practically impossible to find the exact solution of a computational problem in a reasonable time, the algorithm is looking for an approximated solution. The search is then stopped when the stopping condition for the algorithm is fulfilled, which can be spitted into two different types:

- **Fixed budget:** The algorithm is given a certain amount of fixed computational budget (for example time) and returns its output when it used all of it.
- **Fixed confidence:** The algorithm has an unlimited budget but stop when the current solution has a probability of at least $\delta$ to be the exact solution, where $\delta$ is the confidence bound fixed by the user.

## I.1 Machine learning

**Machine Learning (ML)** refers to a technology where computers learn to perform a task without being explicitly programmed to do so. This approach is in stark contrast with the traditional procedure of explicitly defining a sequence of steps to be carried out in order to reach a specified goal. It also represents a key advantage in applications such as pattern recognition, where it may be very difficult to capture the rules that allow humans to discriminate patterns [6]. One can distinguish three main machine learning types, namely supervised, unsupervised and reinforcement learning (Figure 2).

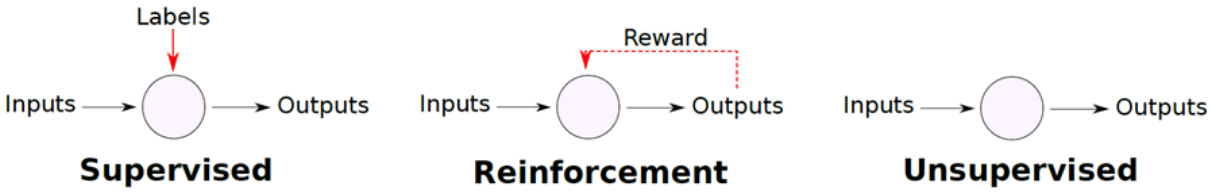


**Figure 2:** Different types of machine learning algorithms, adapted from [7].

- The most common machine learning approach is **Supervised Learning** [6], where labeled training data, i.e. a set of inputs and desired outputs, is fed to the algorithm. The goal of the learning algorithm is to infer a function that maps the inputs to the desired outputs. This class of algorithms can be applied to perform regression (e.g. linear, logistic, polynomial) and classification tasks (e.g. decision trees, support vector machines).
- In contrast to this, **Unsupervised Learning** deals with unlabelled data where no output is provided to the algorithm. Instead, the goal is to extract the underlying structure of the input data, by either clustering the input samples (e.g. k-means clustering, hierarchical clustering), obtaining a lower-dimensional representation of the data (e.g. PCA, ICA) or estimating the underlying density distribution (e.g. maximum likelihood estimation, nearest-neighbors).
- Finally, in **Reinforcement Learning**, the system is trained to make a sequence of actions in order to maximize its performance, defined in terms of a cumulative reward (e.g. monte carlo methods, temporal difference learning).

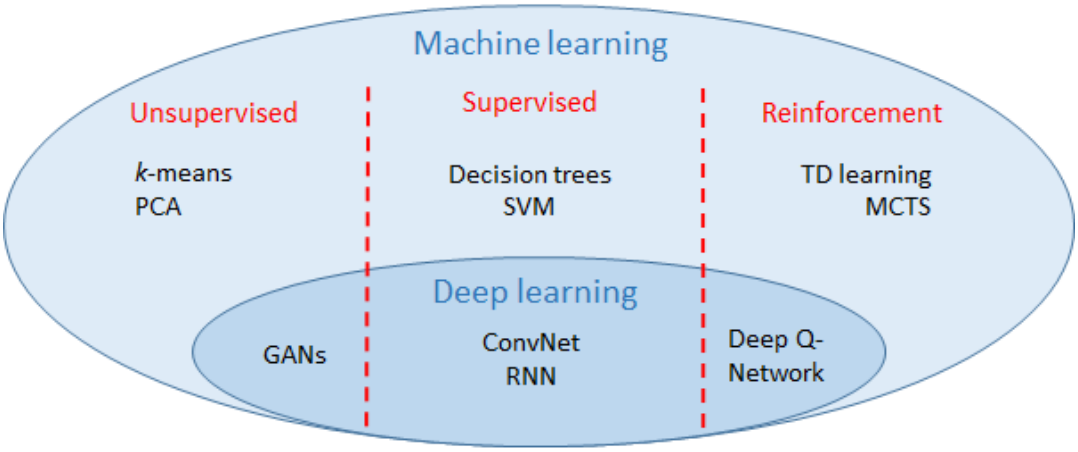


**Figure 3:** Overview of some widely used machine learning and deep learning algorithms classified according to their learning frameworks.

### Classification vs Regression problem

In machine learning, classification is about predicting a label and regression is about predicting a quantity. A classification task involves taking an input and labelling it as belonging to a given class, so the output is categorical (or discrete). On the other hand, a regression task involves the prediction of a continuous valued output.

### Training set and test set

When one wants to build a model, the dataset is usually splitted into at least two subdatasets: the training set and the test set. The model is initially trained on the training set, and the test set is then used to provide an **unbiased** evaluation of the model.

### Overfitting

Overfitting appears when a model learns the detail and noise in the training data to the extent that it negatively impacts the performance of the model on new data. This means that the noise or random fluctuations in the training data is picked up and learned as concepts by the model, but the problem is that these concepts do not apply to new data and negatively impact the models ability to generalize. This is a very important concern in machine learning because overfitting is sometimes not easily detectable and the negative impact on the model can be considerable.

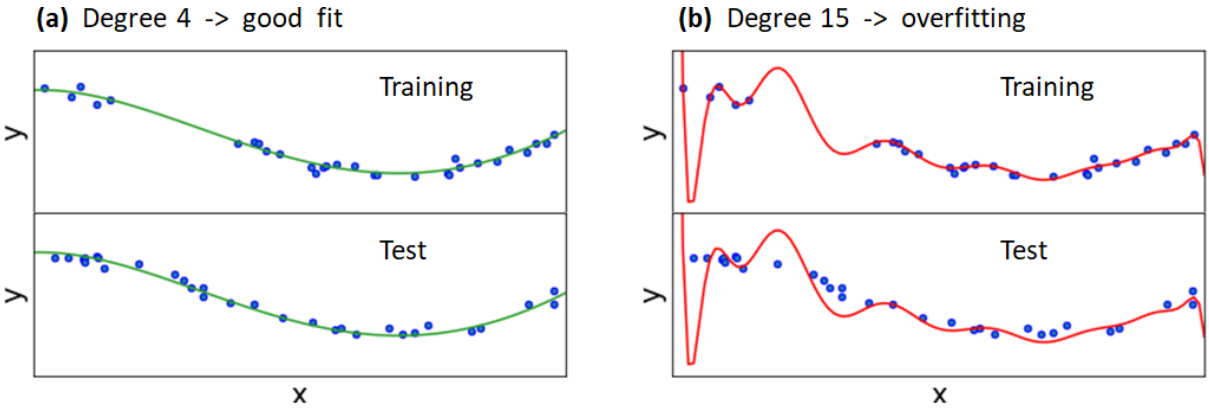


**Figure 4:** (a)Well fitted data and (b)Overfitted data shown on both the training and test set. A higher degree polynomial function fits the data better on the training set but then perform poorly on the test set. The training set and test set both contains 30 examples (blue points) and the continuous line represent the fitted function.

In particular, overfitting generally appears when the number of features is relatively large compared to the number of examples (Figure.4, 16 features are used for only 30 training examples). In practice, having a training set at least 5 to 10 times larger than the number of feature does not generally leads to over fitting. Although there exist various techniques to reduce the risk of overfitting, the best way to avoid it is to simply get more training data or to decrease the number of features.

## I.2 Motivation for Feature Selection

**Feature Selection (FS)** refers the process of selecting a subset of relevant features for the construction of a model. Decreasing the number of feature have the advantage of reducing overfitting, simplifying models, and also involve shorter training time which makes it an unavoidable concern in machine learning. It is important to highlight that **redundant** and **irrelevant** features are two distinct notions, an irrelevant feature does not gives any useful information, while a redundant feature is relevant but does not bring additional information because the information is already contained in other inputs.

### An overview of existing methods

The feature selection methods are typically presented in three classes based on how they combine the selection algorithm and the model building:

- **Filter methods:** They proceed by independently scoring features using some score function (based on correlation with the labels), and usually return a ranking of the features rather than a feature subset, the feature subset is then selected by choosing the top $f$ ranked features. One advantage of such method is that it is relatively fast and also independent from the prediction model, however filter methods tend to select redundant variables because they generally do not account for the feature inter-dependencies.
- **Wrapper methods:** They evaluate feature subsets by training a predictive model with the selected features and checking how well it performs when making predictions (the score is for example the error rate of the model). This makes possible the detection of the interactions between features and guaranty that the selected feature set is well adapted specifically to our model. The main downsides of such methods is that it increases the risk of overfitting when the number of observations is insufficient, and also that it involves a relatively heavy computation time when the number of features is large.
- **Embedded methods:** They perform feature selection as part of the model construction process with the combination of learning and feature selection through prior or posterior regularization. They tend to be between filters and wrappers in terms of computational complexity.

Feature selection techniques should be distinguished from feature extraction. Feature extraction creates new features from functions of the original features, whereas feature selection methods return a subset of the original feature set.

# II Feature selection as Reinforcement Learning

## II.1 The Multi-Armed Bandit problem applied to feature selection

In probability theory, the multi-armed bandit problem is a problem in which we have to find how to allocate a fixed limited budget between alternative possible action in order to maximize the total expected gain. The crucial trade-off in bandit problem is between the exploitation of the arm that has the highest expected payoff and the exploration of the other arms.

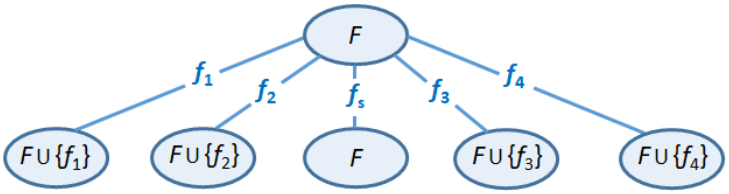


**Figure 5:** Example of a MAB arm choice for feature selection with 4 possible new features.

For our feature selection problem, it means that from a feature subset $F$, the agent has multiples choices and have to select the feature which maximize its expected reward after a limited number of visit. it can either add a new feature $f \in \mathcal{F} \setminus F$, or decide to stop there. Formally, we introduce a virtual stopping feature $f_s$, where the agent can chose the arm $f_s$ to avoid choosing any additional features.

### The feature lattice

Given a feature set $\mathcal{F}$, we define a graph $\mathcal{T}$ for which each node $N_F \in \mathcal{T}$ correspond to a feature subset $F \subset \mathcal{F}$, the number of nodes in the graph is then $|P(\mathcal{F})| = 2^{|\mathcal{F}|}$. For a node $F$, we define the parent node set $ParentNodes(F)$ and the child nodes set $ChildNodes(F)$ as follow:

$$\begin{aligned} ChildNodes(F) &= \{F_c \subset \mathcal{F},\ \exists f \in \mathcal{F} \setminus F,\ F_c = F \cup \{f\}\} \\ ParentNodes(F) &= \{F_p \subset \mathcal{F},\ \exists f \in F,\ F_p = F \setminus \{f\}\} \end{aligned} \tag{1}$$

We also define the depth of a node $N_F$ in the feature lattice as $d = |F|$, the node then has $d$ parents and $|\mathcal{F}| - d$ children:

$$\begin{aligned} |ChildNodes(F)| &= |\mathcal{F}| - |F| \\ |ParentNodes(F)| &= |F| \end{aligned} \tag{2}$$

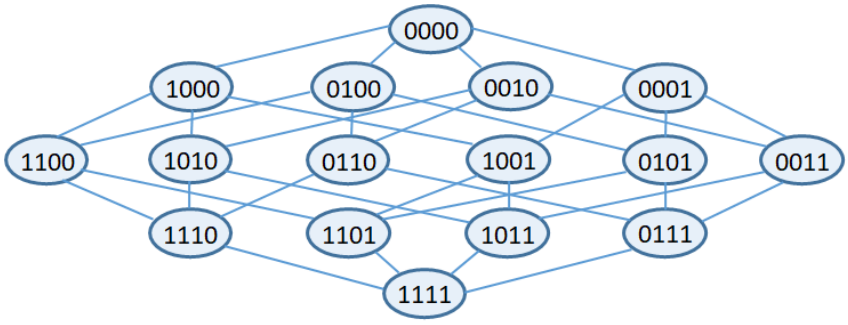


**Figure 6:** Example of a feature lattice with a feature set of cardinal $|\mathcal{F}| = 4$, containing $2^4 = 16$ nodes.

### II.1.1 The reward : How to evaluate a feature subset ?

The search for the optimal arm requires a scoring metric that grades a subset of features, and the choice of this evaluation metric heavily influences the algorithm. Because a computationally heavy reward considerably affects the performance of the algorithm, it is also important to account for the computational cost of the evaluation, and this section discuss the choice of the reward evaluation for our problem.

#### Evaluation with statistical methods (Filter metrics)

The evaluation of a feature set can be done by different approach. For example, the Correlation-based Feature Selection (CFS) [8, 9] is using information theory to account for features-labels correlation as well as the features-features correlations. The score of a feature subset $F$ is evaluated by calculating its merit:

$$\text{Merit}(F) = \frac{f\overline{r_{cf}}}{\sqrt{d + d(d-1)\overline{r_{ff}}}} \tag{3}$$

Where, $\overline{r_{cf}}$ denotes the average value of all feature-classification correlations, and $\overline{r_{ff}}$ the average value of all feature-feature correlations. The advantage of such methods is that it is independent from the prediction model, and also that it accounts for redundancy in the feature subset. Furthermore, the merit computation has a complexity of $O(nd^2)$ [9] (where $n$ is the training set size), which is not heavy when the number of features $d = |F|$ is not relatively large.

#### Evaluation with a classifier (Wrapper metrics)

Another way to evaluate a feature subset is to train a classifier with the selected features and see how well it performs when making predictions on the examples. The reward can then be taken as the Accuracy (ACC) or as the Area under Curve (AUC) of the classifier (see A.1). The most commons classifier are summarized in Table 7:

| Algorithm | Classification/Regression | Training | Prediction |
|---|---|---|---|
| Decision Tree | C+R | $O(n^2d)$ | $O(d)$ |
| Random Forest | C+R | $O(n^2dn_{trees})$ | $O(dn_{trees})$ |
| Gradient Boosting ($n_{trees}$) | C+R | $O(ndn_{trees})$ | $O(dn_{trees})$ |
| SVM (Kernel) | C+R | $O(n^2d + n^3)$ | $O(n_{sv}d)$ |
| k-Nearest Neighbours (naive) | C+R | – | $O(nd)$ |
| Nearest centroid | C | $O(nd)$ | $O(d)$ |
| Neural Network | C+R | ? | $O(dn_{l1} + n_{l1}n_{l2} + ...)$ |
| Naive Bayes | C | $O(nd)$ | $O(d)$ |

**Figure 7:** Most common classifier with their respective training and prediction complexity.

Along with the computational cost, it is also important to account for the boundaries of the classifiers. While the Naive Bayes and Nearest centroid classifier has very cheap computational cost, their predictions has very limited boundaries, making them very interesting for some specific training set, but not ideal in the general case. On the other hand, Random forest or Neural Networks has proven very accurate for considerably complex boundaries, but their computational cost is too heavy for our feature set evaluation.

The $k$-Nearest Neighbour ($k$-NN) classifier can make accurate predictions on relatively complex boundaries, and also have the advantage of requiring no training. Thus, it seems to be a good trade-off between computational cost and boundary's complexity.

### Dealing with large training sets

The computational complexity of one prediction for the $k$-NN classifier is $O(nd)$, and to classify each example in the training set, $n$ predictions are necessary. Thus, the complexity for evaluating a feature subset with $k$-NN is $O(n^2 d)$, and this polynomial computation time becomes a problem when dealing with large data-set.

To solve this problem, it has been proposed [10] to make predictions on an uniformly selected small subsamble of $\mathcal{L}$, denoted $\mathcal{V}$, instead of the whole training set. The complexity of the reward computation then becomes $O(mnd)$ (with $m = |\mathcal{V}|$). By doing so, the feature set evaluation indeed have some variance, but still gives a reliable score for the feature subset, while considerably reducing the computational cost ($m$ is usually taken as 100 or even 1000 times smaller than $n$).

### The k-Nearest Neighbour reward

We have chosen to evaluate the feature subsets with the AUC (details section A.1) on a k-NN classifier, and the reward attached to a feature subset $F$ is computed as follows:

**Algorithm** FUSE , Greedy-SR

**function** REWARD ($F$)

- Compute $\mathcal{V}$ by uniformly selecting $m$ examples in the training set $\mathcal{L}$.

**for** each labeled example $(x, y)$ in $\mathcal{V}$ **do**

- Compute the Euclidean distances to other examples in $\mathcal{L}$ based on features in $F$.
- Find the $k$ nearest neighbours $\mathcal{N}_{\mathcal{F},k}(x)$ and count for the number of positively labeled examples among these neighbours: $s_F(x) = |\{x \in \mathcal{N}_{\mathcal{F},k}(x),\ \ y > 0\}|$.

- Compute the Reward (Area Under the ROC curve) as follow:

$$V = \frac{\left|\left\{(x, x') \ \in \mathcal{V}^2,\ \ s_F(x) < s_F(x')\ ,\ \ y < y'\right\}\right|}{|\{(x, x') \ \in \mathcal{V}^2,\ \ y < y'\}|} \tag{4}$$

**return** $V$

Reward calculation for Feature Selection performed by algorithms FUSE and Greedy-SR.

The optimized implementation details as well as a detailed complexity study are discussed in B.2. The nearest neighbour search is done considering Euclidean distances, but other distances, like Cosine similarity, are also possible. To gain computation time, it is also worth to consider alternative nearest neighbour search, like the KD-tree algorithm, or approximated solution like Local Sensitive Hashing (LSH). These possible improvements are discussed in II.2.4).

## II.2 Finding the best node in the feature lattice

Now that we have a metric to evaluate a feature subset, we want to find the one which maximize the score (= finding the best arm in the MAB problem). The simplest strategy is to test each possible feature subset to find the one with the highest score, however, as the number of subset to test grows exponentially with the number of features: $|P(\mathcal{F})| = 2^{|\mathcal{F}|}$, it becomes computationally intractable for all but the smallest of feature sets. Thus, it is necessary to define a strategy to find an approximate optimal solution in a reasonable amount of time.

### II.2.1 An overview of existing strategies

As shown in part II.1, all the features subset can be associated to a node that is part of the feature lattice, and different strategies can be used to explore the graph.

- **Greedy Method**:
  This is the strategy used by most of the feature selection algorithm. It is relatively simple and generally leads to relatively good approximated solution in a reasonable time. It consists of making step by step the optimum local choice until a stopping condition is fulfilled.

- **Monte Carlo Search Tree (MCST)** :
  While relatively unused for feature selection, MCST is widely used when it comes to game play algorithm (such as Go or Chess). The algorithm is playing a relatively large number of random games and select the node which maximize the reward at the end of the search.

#### The Exploration / Exploitation trade-off

The crucial trade-off in bandit problem is between the **exploitation** of the arm that has the highest expected payoff and the **exploration** of the other arms. The strategy that determine if an arm has to be pulled or not is called the tree policy. From the current node, each arm are rated with a score which account for both the current arm average and uncertainty, and the one with the highest score is pulled. As an example, the two most widely used policies are:

- **Upper Confidence Bound (UCB)** [11]:
  For a node with a total of $N$ pulls, each arm is rated considering its average $\hat{\mu}_a$ and its number of pull $t_a$. The UCB score is supported by a robust theory based on Hoeffding's inequality.

$$\text{UCB} = \hat{\mu}_a + \sqrt{\frac{\ln(N)}{2t_a}} \tag{5}$$

- **Thomson sampling** [12]:
  Each arm is associated to a Beta distribution $P_{\alpha,\beta}(x)$ and a number is randomly pulled from this distribution at each round, the arm with the highest pulled value is then played. The parameters $\alpha$ and $\beta$ are updated at each iteration according to the outcomes.

$$P_{\alpha,\beta}(x) = \frac{1}{B(\alpha,\beta)} x^{\alpha-1}(1-x)^{\beta-1} \tag{6}$$

Where $\mathrm{B}(\alpha,\beta)$ is the Beta function to ensure that the total probability integrates to 1.

### II.2.2 Greedy algorithm with successive elimination (Greedy-SR)

The forward Greedy strategy , as illustrated on Figure.10, is an algorithm paradigm that makes the locally optimal choice at each stage starting from the root node. For our feature selection problem, it means starting from the empty feature set and repeatedly adding the best additional feature to the set until no additional feature further improves the set evaluation.

Even though Greedy strategies does not in general find an optimal solution, they always lead to locally optimal solutions that approximate a global optimal solution in a reasonable time. Thus Greedy strategies are widely used in Feature Selection problems because they are relatively simple to implement and provide a reasonable solution.

However, if the reward function does not always return the same score for a specific feature subset (which is our case since the reward is computed each time with a different subsample of $\mathcal{L}$), then it might be difficult to apply greedy strategy as the real value of each arm can not be fully known. Fortunately, it is possible to estimate the probability that one arm is better than the other with a defined confidence $\delta$.

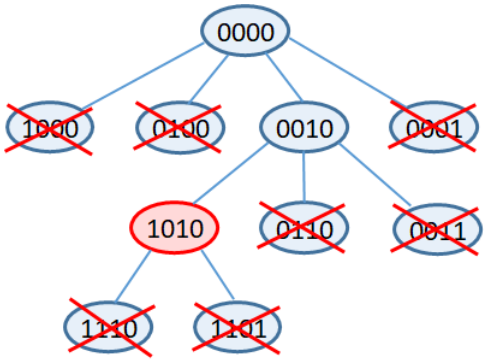


**Figure 8:** Example of a Greedy search with a feature set of cardinal $f = 4$. The feature subset 1010 is selected because its children have lower scores.

#### Dealing with the limited budget

The probability distribution of each arm is not known, and we only have a limited number of iteration (budget) to find the best one. Instead of applying the same number of iteration to each arm and select the one with the highest average at the end, we can use the limited budget in a more clever way and eliminate arms when the likelihood that they are not the best becomes larger than a threshold $\delta$ [13].

#### Hoeffding's inequality and arm elimination

Under the condition that the reward has a [0,1] value, the probability $P$ that the estimated arm mean $\hat{\mu}$ is within $\varepsilon$ of the true mean $\hat{\mu}_{\text{true}}$ is bounded by the Hoeffding's inequality (this inequality holds for any distribution):

$$P\left(|\hat{\mu}_{\text{true}} - \hat{\mu}| > \varepsilon\right) < 2e^{-2n_\mu \varepsilon^2} = \frac{1-\delta}{2} \tag{7}$$

$n_\mu$ denotes the number of time the arm $\mu$ has been selected.

Then, The value of $\hat{\mu}_{\text{true}}$ is known with confidence $\delta$ for:

$$\varepsilon = \sqrt{\frac{\log\left(4/(1-\delta)\right)}{2n_\mu}} \tag{8}$$

By checking the inequality $(\hat{\mu}_1 + \varepsilon < \hat{\mu}_2 - \varepsilon)$? at each iteration, we can keep track on the arms mean values and uncertainties and decide whether or not it should be eliminated. For example, one could decide to eliminate the arm when the probability reach a confidence threshold of $\delta = 99\%$.

### The Greedy-SR algorithm

We propose to use the bandit elimination policy described in [13] to perform the Greedy search in the feature lattice. The elimination strategy in [13] is initially proposed with a Thomson sampling policy, but we have chosen to handle the exploration / exploitation trade-off with UCB, as it is more adapted when the reward function takes continuous values (which is the case of the $k$-NN reward).

We introduce the Greedy strategy with Successive Reduction of arm as Greedy-SR: At each stage, we deal with the limited computational budget with an elimination policy based on Hoeffding's inequality to find the best child. At each iteration, the exploration / exploitation trade-off is handled by the UCB policy, and the search is stopped when the selected child has an average lower than its parent (it means that adding more feature does not further improve the feature subset evaluation).

We denotes $N_F$ the node associated to the feature subset $F$, it contains all of the information about its children (such as average, number of visits, etc..), more details are available in II.2.3.

**Algorithm** Greedy-SR

**procedure** MAIN
  Initialize $\mathcal{T} = \varnothing$
  *Find Best Child* ($\mathcal{T}$, $\varnothing$, 0)

**function** FINDBESTCHILD ($\mathcal{T}$, $F$, $\hat{\mu}_{\text{parent}}$)
  $\mathcal{T} \leftarrow \mathcal{T} \cup \{N_F\}$
  *SuccessiveElimination* ($F$) ▷ Explore the children
  $f_{\max} \leftarrow \underset{f \in ChildNodes(N_F)}{\operatorname{argmax}} \hat{\mu}_{F,f}$ ▷ Select the child with highest average
  **if** $\hat{\mu}_{F,f_{\max}} < \hat{\mu}_{\text{parent}}$ **then** ▷ Stop the search if child is weaker
    **return**
  **else**
    *Find Best Child* ($\mathcal{T}$, $F \cup \{f_{\max}\}$, $\hat{\mu}_{F,f_{\max}}$)

**function** SUCCESSIVEELIMINATION ($F$)
  **while** within computational budget per node **do**
    $f^* \leftarrow \underset{f \in ChildNodes(N_F)}{\operatorname{argmax}} \text{UCB}(F, f)$ ▷ Select arm that maximize UCB
    $V \leftarrow$ *Reward* ($F \cup \{f^*\}$) ▷ Pull arm
    Update $N_F$ ▷ Update the node with $V$
    **for** Child $f$ in $ChildNodes(N_F)$ **do** ▷ Check for children to eliminate
      **if** $(\hat{\mu}_{F,f} + \varepsilon_{F,f} < \hat{\mu}_{F,f^*} - \varepsilon_{F,f^*})$ **then**
        $ChildNodes(N_F) \leftarrow ChildNodes(N_F) \setminus \{f\}$

The function *FindBestChild* is recursive, it is calling itself until the stopping condition is fulfilled. After the algorithm execution, the best feature subset is the one at the end of the tree $\mathcal{T}$.

### II.2.3 UCT-based Feature Selection (FUSE)

It has been proposed in [10] to find an approximated optimal feature subset with a one player game approach relying on the Monte-Carlo Tree search UCT (Upper Confidence Tree [14]). The paper [10] also propose some improvement to UCT to deal with the finite unknown horizon (the target number of relevant features) and the huge branching factor of the search tree (when the number of features is large). This algorithm have the advantage of converging to the general optimal feature subset for infinite budget, and this section provide the details about how the FUSE algorithm works.

The algorithm aims at finding the best approximated optimal feature subset in a limited budget. To do so, it iterates within the available computational budget with the following process:

- **Selection**: Starting at the root node, the UCT child selection policy is recursively applied to descend through the tree until an unvisited or a final node is reached.
- **Simulation**: If unvisited, the node is added to expand the tree, and simulation is run according to the random policy to produce an outcome.
- **Backpropagation**: The simulation result is backpropagated through the selected nodes to update their statistics.

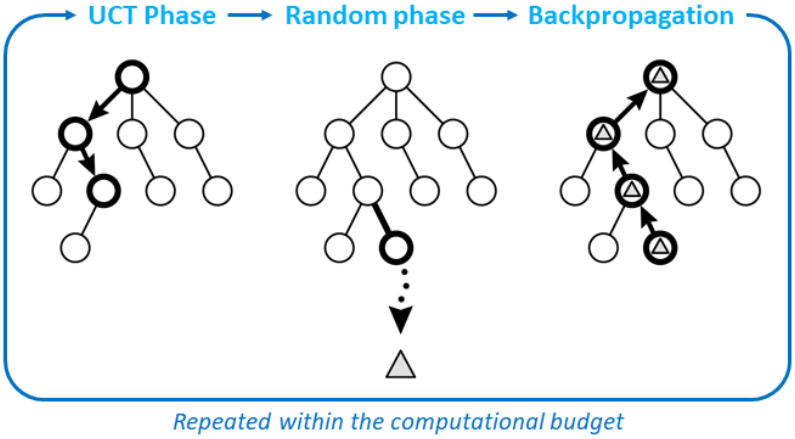


**Figure 9:** One iteration of the FUSE search:
(1) Selection
(2) Simulation
(3) Backpropagation
Adapted from [15].

After the search, the approximated optimal node is taken as the node at the end of the most visited path.

#### The Nodes and stopping feature

The FUSE algorithm is seeking for the best feature subset in the feature lattice, which contains $2^{|\mathcal{F}|}$ nodes $N_F$, each associated with a feature subset $F$ (see part II.1). When the search algorithm is at a current node $N_F$, it has the possibility to select a child node $N_{F_c}$:

$$\text{with} \quad F_c \in \{F_c \subset \mathcal{F},\ \exists f \in \mathcal{F} \setminus F,\ F_c = F \cup \{f\}\} \tag{9}$$

In addition to its children, we also consider for each node a virtual stopping feature $f_s$, which allows the search to stop at the current node instead of adding new features to the subset. The stopping feature is included in the feature set $f_s \in \mathcal{F}$, and can thus be chosen any time during the search. If the stopping feature is chosen, then the node is considered final, and the selection phase is stopped to evaluate the node.

**Object** for FUSE, Greedy-SR

**Class** Node$(F) = N_F$

| | |
|---|---|
| $F$ | ▷ The feature subset |
| $T_F$ | ▷ The number of visit |
| $t_{F,f}$ | ▷ The number of visit of each child node |
| $\hat{\mu}_{F,f}$ | ▷ The reward average of each child node |
| $\hat{\sigma}_{F,f}$ | ▷ The reward variance of each child node |
| $AllowedFeatures$ | ▷ The allowed features subset |

A node $N_F$ contains all the necessary information to perform the search during the UCT phase.
The word *Class* refers to object oriented languages like `C++`.

### The UCT selection phase

During the selection phase, all the child nodes are evaluated according to their UCB1-tuned score and the one that has the maximum value is selected. Even though there is no theoretical proof supporting UCB1-tuned, it demonstrated better performances than the original UCB when dealing with large number of arms [16].

$$\text{UCB1-tuned}(F, f) = \hat{\mu}_{F,f} + \sqrt{\frac{c_e \ln(T_F)}{t_{F,f}} \min\left(\frac{1}{4}, \hat{\sigma}^2_{F,f} + \frac{2\ln(T_F)}{t_{F,f}}\right)} \tag{10}$$

| | |
|---|---|
| $T_F$ | Number of times node $F$ has been visited |
| $t_{F,f}$ | Number of times feature $f$ has been selected from node $F$ |
| $\hat{\mu}_{F,f}$ | Average reward collected when selected feature $f$ from node $F$ |
| $\hat{\sigma}_{F,f}$ | Variance of the reward collected when selected feature $f$ from node $F$ |
| $c_e$ | Exploration parameter (set to 2 in the original UCB formula) |

### The Random phase

When an unvisited node is selected ($T_F = 0$), it is necessary to produce an outcome to evaluate the arm. However, as the number of features contained in the optimal feature subset is not known, it is not wise to stop the random phase always at the same depth. To better deal with this finite unknown horizon, a good way is to uniformly select features until the stopping feature is selected [10]. At each round in the random phase, the stopping feature is selected with probability $1 - q^d$, where $q \in [0, 1]$ is a parameter of the algorithm.

### Backpropagation

The simulation result is backpropagated through the selected nodes to update their statistics and thus inform future UCT decisions. The backpropagation in the FUSE algorithm is only updating the nodes in the current path, and does not update all node contained by $F$ due to the high branching factor of the graph.

### The Many-armed Issue

The problem when dealing with a large number of feature is that the algorithm must resist to over-exploration at each level of the tree, and basic UCT has demonstrated poor performances when dealing with a huge branching factor.

### Allowed Features

To have a better control on the exploration, a discrete heuristic has been proposed [17], it restricts the number of considered child nodes depending on $T_F$. Formally, a new child is added whenever $\lfloor T_F^b \rfloor$ is incremented, where $b \in [0, 1]$ is a parameter of the algorithm. The child node to be added to $AllowedFeatures$ is the one maximizing its g-RAVE score among the remaining child nodes that are not already in $AllowedFeatures$.

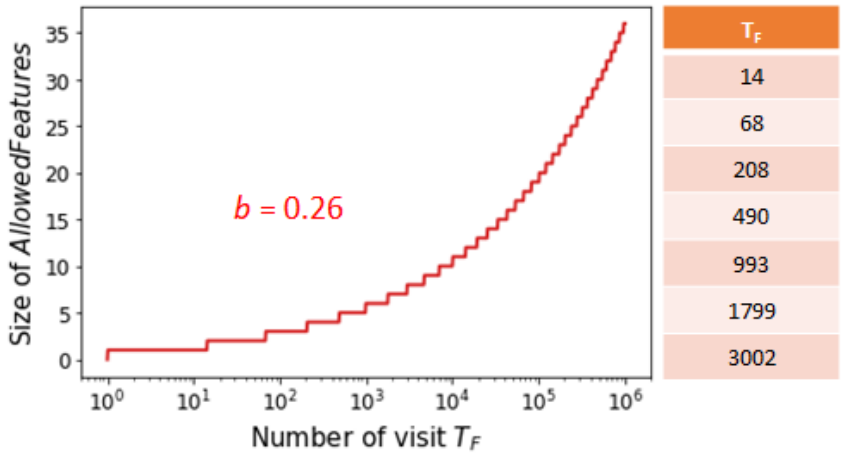


**Figure 10:** size of $AllowedFeatures$ as a function of the number of visit $T_F$ for $b = 0.26$. The table on the right represent the value of $T_F$ for which a feature is added.

For $b = 0.26$, after $10^6$ iterations, the number of allowed arms is *only* 36, whereas it is not uncommon to see large training with more than 1000 features. This shows how important the impact of RAVE is on the search because it will determine which features are explored and which are not.

**Algorithm** FUSE

**procedure** DISCRETEHEURISTIC ($N_F$)
  **if** $\lfloor T_F^b \rfloor - \lfloor (T_F - 1)^b \rfloor \neq 0$ **then**
    $f^* \leftarrow \underset{f \notin AllowedFeatures}{\text{argmax}} \text{RAVE}_{F,f}$
    $AllowedFeatures \leftarrow AllowedFeatures \cup \{f^*\}$

**RAVE score**

The discrete policy adds the number of considered child nodes logarithmically with the number of visit (II.2.3), which means that, for a limited budget, only few of the child nodes are actually considered at each node, making the choice of the child nodes to be considered a major concern, particularly when the number of redundant features is large.

The selection of new nodes can benefit from any knowledge gained within the search. It has been proposed [18] to define a RApid Value Estimation (RAVE) score for the features, that are used to focus the search and avoid a (hopeless) uniform exploration of the feature space. The global RAVE score of a feature $f$ is defined as follow:

$$\text{g-RAVE}_f = \text{average}\{V(F),\ f \in F\} \tag{11}$$

The problem is that g-RAVE only provides a global indication on feature relevance, but can not account for redundancy relatively to the current node, thus it also makes sense to consider the feature conditionally to those selected within the current node [10], yielding to the $\ell$-RAVE factor:

$$\ell\text{-RAVE}_{F,f} = \text{average}\{V(F_t),\ F \subset F_t,\ f \in F_t\} \tag{12}$$

To calculated the RAVE score, [10] propose to consider both g-RAVE and $\ell$-RAVE with the following expression:

$$\begin{aligned} &\text{RAVE}_{F,f} \ = \ (1 - \beta_{F,f}) \ . \ \ell\text{-RAVE}_{F,f} \ + \ \beta_{F,f} \ . \ \text{g-RAVE}_f \\ &\text{with} \ \ \beta_{F,f} = \frac{c_l}{c_l + t_{F,f}} \end{aligned} \tag{13}$$

| | |
|---|---|
| $t_{F,f}$ | Number of iterations involved in $\ell$-RAVE$_{F,f}$ computation |
| $c_l$ | g-RAVE vs $\ell$-RAVE weight parameter |

When the information about $\ell$-RAVE$_{F,f}$ is inaccurate due to a low number of computation, we consider the g-RAVE score instead, but gradually account for $\ell$-RAVE$_{F,f}$ as its value gets more reliable.

**Algorithm** FUSE

**procedure** MAIN
  Initialize $\mathcal{T} = \{N_\varnothing\}$
  Initialize $\forall f \in \mathcal{F}$, g-RAVE$(f) = 0$
  **while** within computational budget **do**
    $V \leftarrow$ *Iterate* $(\mathcal{T},$ g-RAVE, $\varnothing)$

**function** ITERATE $(\mathcal{T}$, g-RAVE , $F)$
  **if** $f_s \in F$ **then**
    $V \leftarrow$ *Reward* $(F \setminus f_s)$
    Update g-RAVE
  **else**
    Initialize $N_F =$ Node$(F)$
    **if** $T_F \neq 0$ **then**
      *DiscreteHeuristic* $(N_F)$
      $f^* \leftarrow \underset{f \in AllowedFeatures(N_F)}{\text{argmax}}$ UCB1-tuned$(F, f)$
      $V \leftarrow$ *Iterate* $(\mathcal{T}$, g-RAVE , $F \cup \{f^*\})$
    **else**
      $\mathcal{T} \leftarrow \mathcal{T} \cup \{N_F\}$
      $V \leftarrow$ *IterateRandom* (g-RAVE , $F)$
    Update $N_F$
  **return** $V$

**function** ITERATERANDOM (g-RAVE , $F)$
  **while** rand(0,1) $< q^{|F|}$ **do**
    $f^* \leftarrow$ uniformly select feature in $\mathcal{F} \setminus (F \cup \{f_s\})$
    $F \leftarrow F \cup \{f^*\}$
  $V \leftarrow$ *Reward* $(F)$
  Update g-RAVE
  **return** $V$

The function *Iterate* is recursive, it is calling itself until the stopping feature is selected or a new node is visited, backpropagation can then occur by backward recursive calls. After the algorithm execution, the best feature subset is taken as the one at the end of most visited path.

### II.2.4 FUSE-2: Proposals to improve FUSE

This sections showcases the limitations of the FUSE algorithm [10], and propose various improvements to enhance its performances. The improved version of FUSE algorithm is introduced as FUSE-2.

#### The stopping feature problem

Efficiently evaluating the RAVE score of the stopping feature is not an easy problem. If we simply consider the stopping feature as a normal feature for its g-RAVE calculation:

$$\text{g-RAVE}_{f_s} = \text{average}\{V(F)\} \tag{14}$$

Then roughly half of the features has a g-RAVE score higher than the stopping features, and this means that in average, $|\mathcal{F}|/2$ child nodes will be explored before the stopping feature is selected (because of the discrete heuristic policy discussed in II.2.3). if the number of feature is large, this behavior is very harmful for the algorithm because it explores at very high depth and leads to extremely unbalanced tree.

Let $o$ be the optimal number of features and $r$ the number of relevant features, the number of redundant features is thus $r_u = r - o$. If $r \ll |\mathcal{F}|$, then all of the relevant features has a g-RAVE score higher than the stopping features.

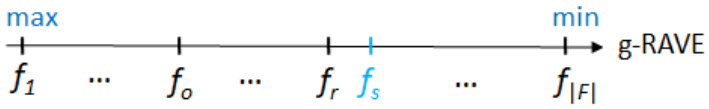


**Figure 11:** The g-RAVE score of the stopping feature $f_s$ is lower than the one of all relevant features ($f_1..f_r$).

This means that all of the relevant features will be explored before the stopping feature is added, discarding the fact that they might be redundant. The fact that the algorithm keep exploring deeper until all the redundant features are selected is not what we hope for.

A way to limit the depth search of the lattice would be to increase the value of $\text{g-RAVE}_{f_s}$. In [10], it is suggested to consider a different RAVE score for the stopping feature at each depth of the nodes:

$$\text{g-RAVE}_{f_s^{(d)}} = \text{average}\{V(F),\ |F| = d + 1\} \tag{15}$$

While the g-RAVE score for depth $d > o$ slightly increases because the score of lower depth nodes is not accounted, it is still not solving the problem. It seems like it is not possible to correctly evaluate the stopping feature RAVE score, thus we propose the following.

Since it is preferable for the algorithm to search at low depth rather than high depth (at least at the beginning), we propose to consider for the stopping feature $f_s$ an infinite g-RAVE score.

$$\text{g-RAVE}_{f_s} = \infty \tag{16}$$

Thus the stopping feature will always be the first to be added to $AllowedFeatures$, and even though the expansion might be slow at the beginning, the tree will gradually explore at higher depth and stay balanced.

### Accurate node evaluation

The problem caused by always exploring first the stopping feature is that, at the beginning on the search, the node average is only represented by it's associated feature subset and not by it's children. Thus, the nodes are not accurately evaluated and the search then allocates unnecessary resources to irrelevant nodes because of miss-evaluation.

To solve this problem, we force the search to perform random exploration from the node a certain amount of time $T_{\text{rand}}$ before allowing the selection of any child nodes, so that the node average always accurately represent the real value of the node.

**Algorithm** FUSE-2

**procedure** DISCRETEHEURISTIC ($N_F$)
  **if** ( $T_F > T_{\text{rand}}$ ) **and** ( $\lfloor (T_F - T_{\text{rand}})^b \rfloor - \lfloor (T_F - T_{\text{rand}} - 1)^b \rfloor \neq 0$ ) **then**
    $f^* \leftarrow \underset{f \notin AllowedFeatures}{\text{argmax}} \text{RAVE}_{F,f}$
    $AllowedFeatures \leftarrow AllowedFeatures \cup \{f^*\}$

Improved version of the discrete heuristic described in II.2.3.

### Efficient Backpropagation

While the nodes usually have only one single parent in trees, they can have multiple in lattices. Thus, by updating information only in the current path instead of all the ancestors, we are missing an opportunity to propagate the information in a more efficient way. We propose to update all the parent nodes instead of only the current path during the bakpropagation.

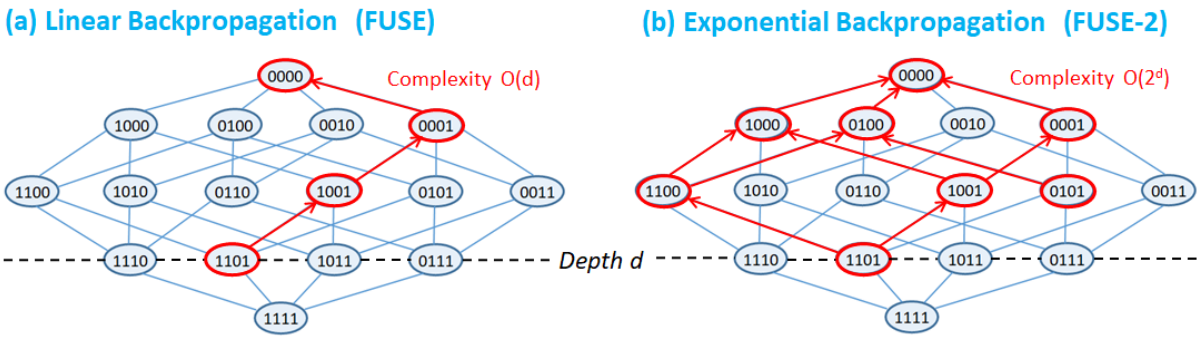


**Figure 12:** (a) Linear and (b) Exponential backpropagation in a feature lattice.

Assuming that, in average, the number of node's updates to find an optimal feature subset is constant, exponential backpropagation will theoretically make the algorithm converge up to $2^d/d$ faster than with linear backpropagation (in iteration unit).

Although the number of ancestors to update scales exponentially with the depth ($O(2^d)$), and can become problematic for high depth, it is in practice not limiting the algorithm, because as described above, we make sure that the FUSE-2 algorithm does not explore at high depth by enforcing the selection of stopping feature when exploring a new node.

### Elimination of irrelevant children

As explained in II.2.4, the number of children considered at each node is considerably limited. During the search, it is possible that arms that has been explored at the beginning turns out to be bad arms, but they will remain in the $AllowedFeatures$ set and thus prevent the algorithm to search for new child that might be more relevant.

To solve this problem, we propose to use a successive elimination policy based on Hoeffding's inequality as described in II.2.2. When the condition for elimination is fulfilled, the arm is removed from $AllowedFeatures$ and a new one is added.

**Algorithm** FUSE-2

**procedure** EliminationPolicy ($N_F$)

$\hat{\mu}_{\max} = \max_{f \notin AllowedFeatures} \hat{\mu}_{F,f}$

**for** $f$ in $AllowedFeatures$ **do**

**if** $(\hat{\mu}_{F,f} + \varepsilon_{F,f} < \hat{\mu}_{\max} - \varepsilon_{\max})$ **then**

$f^* \leftarrow \operatorname{argmax}_{f \notin AllowedFeatures} \text{RAVE}_{F,f}$

$AllowedFeatures \leftarrow AllowedFeatures \cup \{f^*\} \setminus \{f\}$

A node $N_F$ contains all the necessary information to perform the search during the UCT phase. The word *Class* refers to object oriented languages like `C++`.

Successive child elimination policy as described in II.2.2.

### Alternative $k$-NN search

The reward computation currently scales with $O(mn(d+1))$ for the naive $k$-NN search, and this can become heavy for large training set, we propose to use alternative an algorithm for the nearest neighbor search. The most commonly used are the following:

- **k-dimensional trees (KD-trees)** which has a query complexity of $O(\log(N)^d)$, it has proven relatively efficient, but suffer from the Curse of dimensionality, and can becomes worse than brute $k$-NN search if $d$ is too large. In practice, the KD-tree algorithm becomes irrelevant when the dimensionality is higher than $d \approx 15$ [19].
- **Local Sensitive Hashing (LSH)**[20] is an approximate searching method for high dimensional space. It is based on locality-preserving hashing functions, so points that are close to each other in space have a high probability to have the same hashing value.

### *Additional comment*

*Due to a lack of time, a rigorous analysis of the FUSE-2 algorithm performances compared to the one of FUSE has not been conducted yet, mainly because the code for the FUSE implementation has to be written almost from scratch. It is however one of the primary concerns for my further research. Particularly, I will try to show empirical evidences of the effectiveness of each of my proposal in FUSE-2 to that of the original FUSE.*

**Algorithm** FUSE-2

**procedure** MAIN
  Initialize $\mathcal{T} = \{N_\varnothing\}$
  Initialize $\forall f \in \mathcal{F}$, g-RAVE$(f) = 0$
  **while** within computational budget **do**
    $V \leftarrow$ *Iterate* $(\mathcal{T},$ g-RAVE, $\varnothing)$

**function** ITERATE $(\mathcal{T}$, g-RAVE , $F)$
  **if** $f_s \in F$ **then**
    $V \leftarrow$ *Reward* $(F \setminus f_s)$
    *UpdateParents* $(N_F)$
    Update g-RAVE
  **else**
    Initialize $N_F =$ Node$(F)$
    **if** $T_F < T_{\text{rand}}$ **then**
      *EliminationPolicy* $(N_F)$
      *DiscreteHeuristic* $(N_F)$
      $f^* \leftarrow \underset{f \in \mathit{AllowedFeatures}(N_F)}{\text{argmax}}$ UCB1-tuned$(F, f)$
      $V \leftarrow$ *Iterate* $(\mathcal{T}$, g-RAVE , $F \cup \{f^*\})$
    **else**
      **if** $T_F = 0$ **then**
        $\mathcal{T} \leftarrow \mathcal{T} \cup \{N_F\}$
      $V \leftarrow$ *IterateRandom* (g-RAVE , $F)$
      *UpdateParents* $(N_F)$
  **return** $V$

**function** ITERATERANDOM (g-RAVE , $F)$
  **while** rand(0,1) $< q^{|F|}$ **do**
    $f^* \leftarrow$ uniformly select feature in $\mathcal{F} \setminus (F \cup \{f_s\})$
    $F \leftarrow F \cup \{f^*\}$
  $V \leftarrow$ *Reward* $(F)$
  Update g-RAVE
  **return** $V$

**function** UPDATEPARENTS $(N_F)$
  **if** ($N_F$ already updated) **or** $(N_F = N_\varnothing)$ **then**
    **return**
  Update $N_F$
  **for** $N_{F_p}$ in *ParentNodes* $(N_F)$ **do**
    Update $N_{F_p}$

**Object** for FUSE-2

**Class** Node($F$) = $N_F$

| | |
|---|---|
| $F$ | ▷ The feature subset |
| $T_F$ | ▷ The number of visit |
| $\hat{\mu}_F$ | ▷ The reward average |
| $\hat{\sigma}_F$ | ▷ The reward variance |
| $ChildAddress$ | ▷ The tree addresses of the child nodes |
| $AllowedFeatures$ | ▷ The allowed features subset |
| $\ell$-RAVE$_{F,f}$ | ▷ The local RAVE score of all child nodes |
| $t_{F,f_s}$, $\hat{\mu}_{F,f_s}$, $\hat{\sigma}_{F,f_s}$ | ▷ The stopping feature information |

## II.3 Tests on benchmark datasets

The Greedy-SR and FUSE-2 Algorithm described above has been implemented in `C++` and tested on benchmark dataset to check for their relevancy and suitability. It is challenging because the search in the feature lattice involve very large tree and high branching factors, details about the code and implementation choices are described in B.1.

For the following studies, the experimental setup consist of an Intel i5 CPU, 3.2 GHz, 4 Cores with 8GB physical memory. The reward is computed with a 5-NN classifier (see A.2). For FUSE-2, parameters $q = 0.98$, $b = 0.26$, $c_e = 2$, $c_l = 20$ and $m = 50$ are used. For Greedy -SR, due to the relatively computationally cheap cost, no subsampling is used for the reward computation which makes it deterministic ($m = n$), so we compute each child only once and no extra parameter is needed.

### II.3.1 Test on a simple artificial data set

The first step to test the implementation of our algorithms is to try them on a simple artificial dataset. A dataset with 200 features, containing 3 features $x, y, z$, plus 7 redundant and 170 random features is generated. To build the data set, 2000 examples are generated by randomly selecting uniformly $(x, y, z)$ in $[0, 1]^3$, and a linear classification function is used: $f(x, y, z) = (0.1x - 0.8y + 0.6z > 0)$? to calculate the labels. The weight of the feature $x$ is voluntarily set to 0.1 to check if the algorithm is able to find it even though it's relatively weak compared to other features.

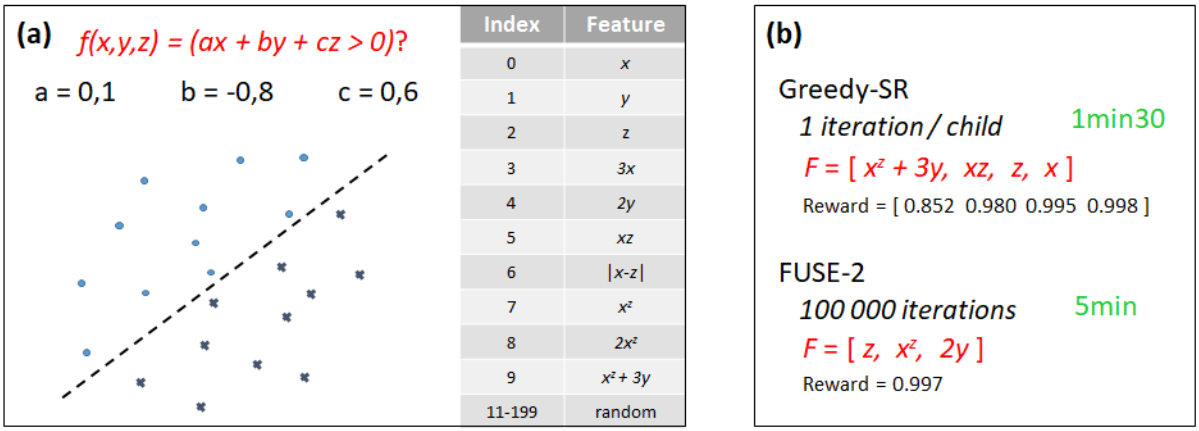


**Figure 13:** (a) Artificial linear dataset generated for testing feature selection algorithm. (b) Results from FUSE-2 and Greedy-SR algorithm.

For our generated dataset, a perfect feature subset should contain only 3 features, where $x$, $y$ and $z$ should all be present in at least one feature. Results on figure.13 shows that the FUSE-2 Algorithm found a perfect feature set after roughly 100 000 iterations. However, regarding the Greedy-SR algorithm, although the first 3 selected features selected are correct, one unnecessary one has been added. One can also notice that the two method selected different features, showing that there is no particular advantage at selecting one relevant feature more than the other.

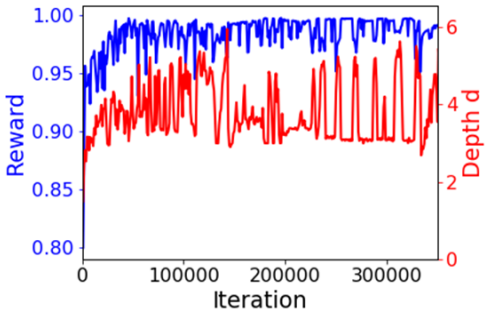


**Figure 14:** FUSE-2 reward averaged over the last 1000 iterations as a function of the number of iterations.

To explain why the two feature selection methods selected different feature set, we have to understand how the reward calculation is affected by redundancy in the feature set. In the ideal case, adding a redundant feature should not change the $k$ nearest neighbor classification, and as a result, adding a redundant feature leads to the same evaluation, for example $V(\{x, y\}) = V(\{x, y, 2x\})$. However, as the training set contains a final number of examples, it is possible that adding a redundant feature does slightly increase the expected reward due to random fluctuation in the classification. As an example, the Greedy-SR added one redundant feature because the reward increased from 0.9953 to 0.9982.

Regarding FUSE-2, figure.14 plot the evolution of the reward and depth of the search as a function of the number of iteration. During the first 50 000 iterations, there is a clear increase in the reward, but it quickly stagnates after that. At that moment the optimal feature subset has already been found, but as emphasized by the fluctuations in the red curve, the search keeps regularly exploring new feature subset with the hope of finding a better one.

It is also worth noticing that the reward of the selected feature set is 0.9970, whereas the one selected by Greedy-SR is 0.9982, and this shows us that FUSE-2 does not necessarily select the feature set with the highest reward. It looks like FUSE-2 does not in practice select additional features when the reward enhancement is too small. While this behavior might have a negative impact in particular cases, it actually makes the FUSE algorithm particularly robust to overfitting.

To understand this behavior, let's look at the node at the end of the most visited path after 1 000 000 iterations $F_{\text{best}}$. As expected, the highest child average reward is the one corresponding to the stopping feature $f_s$ (Figure.15). Now let's dive into the child node $F_{\text{best}} \cup \{5\}$: The average reward of the stopping feature of the child is higher, but it has been surprisingly selected only 500 times, which is relatively low compared to other child nodes (around 4000).

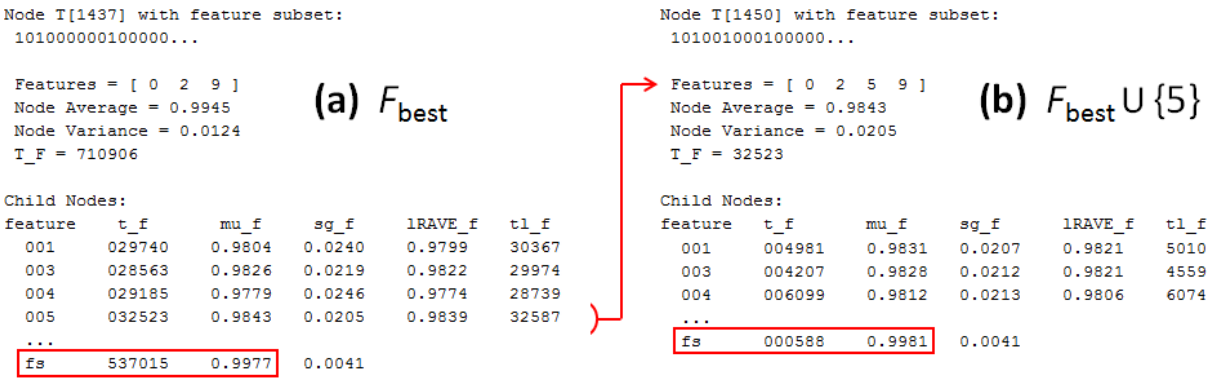


**Figure 15:** After 1 000 000 iterations, (a) node at the end of the most visited path $F_{\text{best}}$ and (b) one of its child nodes $F_{\text{best}} \cup \{5\}$.

This unexpected behavior comes from the exponential backpropagation: The values of the child nodes of $F_{\text{best}}$ are regularly updated even though the node was not selected during the UCT phase, thus preventing the stopping feature to be chosen. As a result, the high score of the stopping feature is *drowned* within the values of other child nodes, thus preventing the parent node from *seeing* that its child node is actually better. This seems to occur only when the improvement in the reward is relatively low, but further analysis are necessary to truly understand this phenomenon. One could also think of additional modifications to the FUSE-2 algorithm to prevent or reduce such behaviors.

### II.3.2 Benchmarks from NIPS 2003 feature selection challenge

As a proof of concept for our algorithms, we test them on the Madelon dataset [21], available in the UCI machine Learning Repository, which has been designed for the NIPS 2003 feature selection challenge [22]. It is an artificial 500-features dataset, where the target concept is set to five relevant features. The other 495 features involve 15 redundant features, built as linear combinations of the relevant ones, and the remaining features are irrelevant. The data set is provided with 2000 examples for the training and 600 examples for the test.

Feature selection is performed by Greedy-SR and FUSE-2 on the Madelon training set, and the obtained feature set are then used to train a 5-NN classifier, whose prediction performance is plotted on the Madelon test set (Figure.16). The results are also compared with two major filtering techniques: Fisher scoring [23] and ReliefF scoring, details about these methods are provided in section III.3.2.

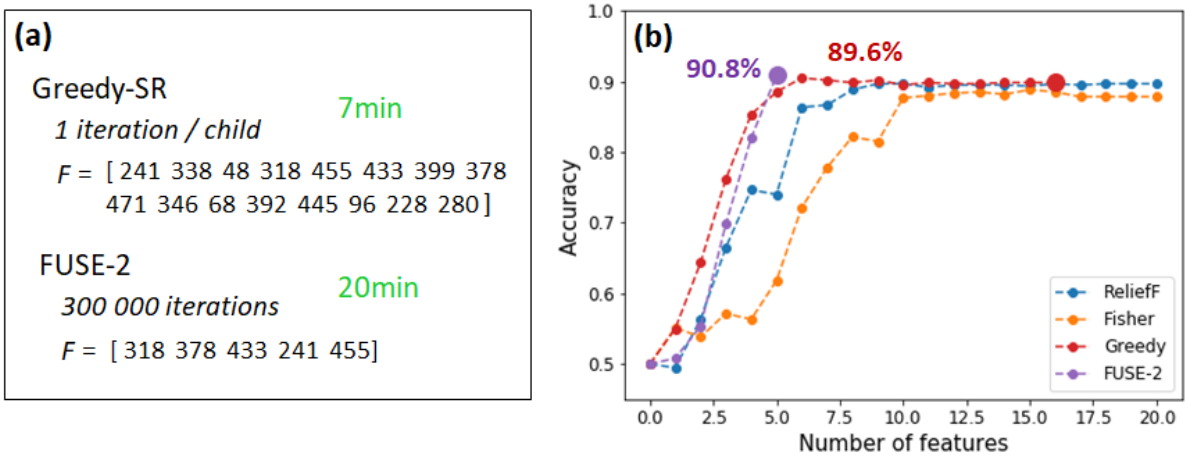


**Figure 16:** (a) Feature subset selected by the FUSE-2 and Greedy-SR algorithms, from the most to least important. (b) Comparison of different feature subset with a 5-NN classifier trained on the Madelon test set, the maximum accuracy of Greedy-SR and FUSE-2 is also written.

Even though both algorithms performed much better than simple filtering methods, an advantage of the FUSE-2 approach over Greedy strategy now becomes very clear: Greedy-SR selected 16 features and overfitted the data by adding features which slightly improved the reward on the training set, but actually had a negative impact on the test set. On the other hand, FUSE-2 selected the target concept of 5 relevant features, proving that FUSE did not overfitted the data.

One disadvantage of FUSE and Greedy-SR is that the feature subset evaluation is not deterministic (as a subsample is generated each time), thus the output feature subset will be different if the seed of the random generator is changed. Fortunately, although the number of iteration needed to find a good feature set might vary if the search happens to be particularly unlucky, experiments have shown that FUSE-2 outputs a similar feature subset after each trials. As for Greedy-SR, the algorithm is less influenced by the uncertainties in the reward computation thanks to its elimination policy,

# III Raman spectroscopy for cancer diagnosis

## III.1 Motivation of the project

The feature selection methods presented in this thesis were primarily developed to be applied to Raman spectra for cancer diagnosis, which is one of the concerns that aim to be solved within the following national 5 years project [24]:

*Development of Accelerated Measurement Technology for Cell Diagnosis by bridging Single-Cell Raman Imaging and Information Science.*

The technology is introduced in the diagnosis of cancer, but more generally, the project aims for the development of a molecular measurement technique in the fields of life science, medicine, and drug-discovery by enabling early identification of diseases that are difficult to diagnose.

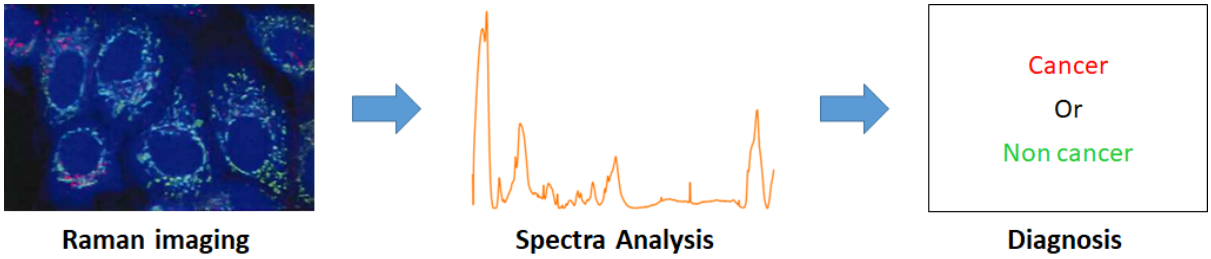


**Figure 17:** Simplified description of the project: (1)Raman image from Osaka group, (2)example spectrum extracted from measured data, (3)diagnosis performed by AI.

Lead-managed by Professor Tamiki Komatsuzaki, the project started in late 2016. Because it is highly interdisciplinary and requires skills in medicine, biology, physics and computer science, various team around Japan are cooperating.

### Osaka - Raman Group

*Osaka University, LASIE - Laboratory for Scientific Instrumentation and Engineering.*
Diriged by Associate Professor Katsumasa FUJITA, the group is developing novel technology for molecular Raman measurements that constantly feeds optimal conditions back to the measuring equipment. In particular, they are now working on a fast and accurate Raman measurement technique that can focus on specific wavenumbers and perform simultaneous measurements at different point in the sample.

### Kyoto - Medical Group

*Kyoto Prefectural University of Medicine, Department of Pathology & Cell Regulation.*
Led by Associate Professor Yoshinori Harada, the group provides samples from patients affected by various disease, and also have their own Raman measurement setup. They provided Cancer and non Cancer cells, and are currently experimenting Raman spectroscopy to diagnosis fat liver disease on rats.

### Sapporo - Information science Group

*Hokkaido University, Research Institute for Electronic Science.*
Also lead-Managed by Professor Tamiki Komatsuzak, the group is using information science and statistical mathematics techniques to analyze spectral imaging data. Particularly, machine learning algorithm are trained to perform cancer diagnosis with relatively high accuracy.

## III.2 Raman spectroscopy

Raman spectroscopy is a characterization technique that relies on the inelastic scattering of light and use the optical properties of the sample to gain information about its molecular composition and strain state with a high-spatial-resolution of few hundreds nanometers. It is also non destructive, which makes it widely used in both physics and chemistry.

Typically, the sample is illuminated with a laser beam, and the electromagnetic radiation from the illuminated spot is then collected by a detector. Because the inelastic Raman scattering (when the kinetic energy of the incident particle is not conserved, Figure.18) is very weak compared to the elastic Rayleigh scattered laser light (when the kinetic energy of the incident particle is conserved), the radiation at the wavelength corresponding to the laser line is filtered out before the rest of the light is collected by the detector (Figure.19). 2D Raman images are obtained by moving the position of the laser spot on the sample.

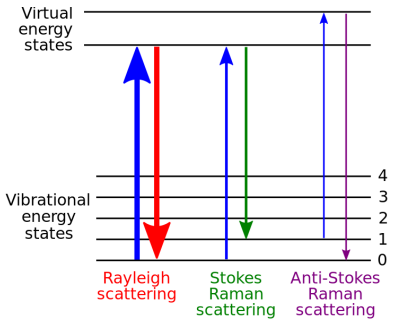


**Figure 18:** Energy-level diagram showing the states involved in Raman spectra.

One downside about this technique is that the typical acquisition time of a Raman spectrum for a single point is relatively high (several seconds), thus making a full Raman image considerably long to be completed (several hours or days). Therefore this technique seems unsuitable for live-cell imaging, and there is a need to shorten the image acquisition times of Raman microscopy for observation of living specimens.

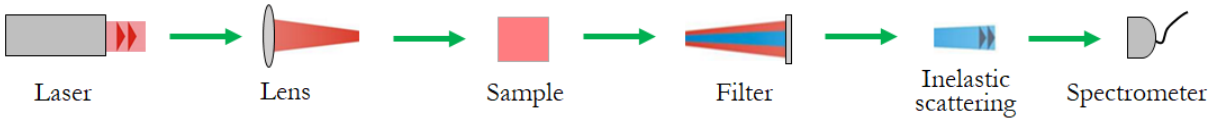


**Figure 19:** Typical standard Raman experimental acquisition.

### Spectrum from biological samples

Raman spectra are commonly plotted as a function of the Raman shift $\Delta\omega$, calculated from the wavelenght $\lambda$[nm] with $\Delta\omega = \left(\frac{1}{\lambda_0} - \frac{1}{\lambda}\right) \times 10^7\,\mathrm{cm}^{-1}$, where $\lambda_0$ refers to the wavelength of the excitation laser. Biological samples usually provides plenty of Raman peaks in a spectrum, each indicating the presence of a specific molecule (Figure.20), notably contained in lipids [25] and proteins [26] (more details in appendix C.1).

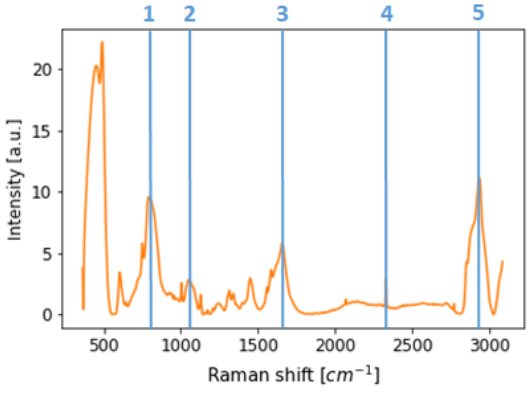


**Figure 20:** Typical Raman spectrum obtained from a living cell by Osaka group with a 532 nm CW laser.
(1) 750 $\mathrm{cm}^{-1}$ → cytochrome-$c$
(2) 1000 $\mathrm{cm}^{-1}$ → phenylalanine
(3) 1556 $\mathrm{cm}^{-1}$ → $O_2$
(4) 2331 $\mathrm{cm}^{-1}$ → $N_2$
(5) 2850 $\mathrm{cm}^{-1}$ → $CH_2$ bond of lipid

### III.2.1 Fast Raman measurement

The Osaka's Raman group recently developed a novel technology for Raman spectroscopy. They built a versatile Raman microscope equipped with both a slit-scanning excitation and a tracking system for dynamic imaging of molecules in living cells. The major advantage of their Raman microscope platform is its ability to perform chemically specific and sensitive Raman imaging at both high spatial and temporal resolutions for diverse applications in live-cell imaging. Their remarkable work has been published in Nature Protocols in 2013 [4].

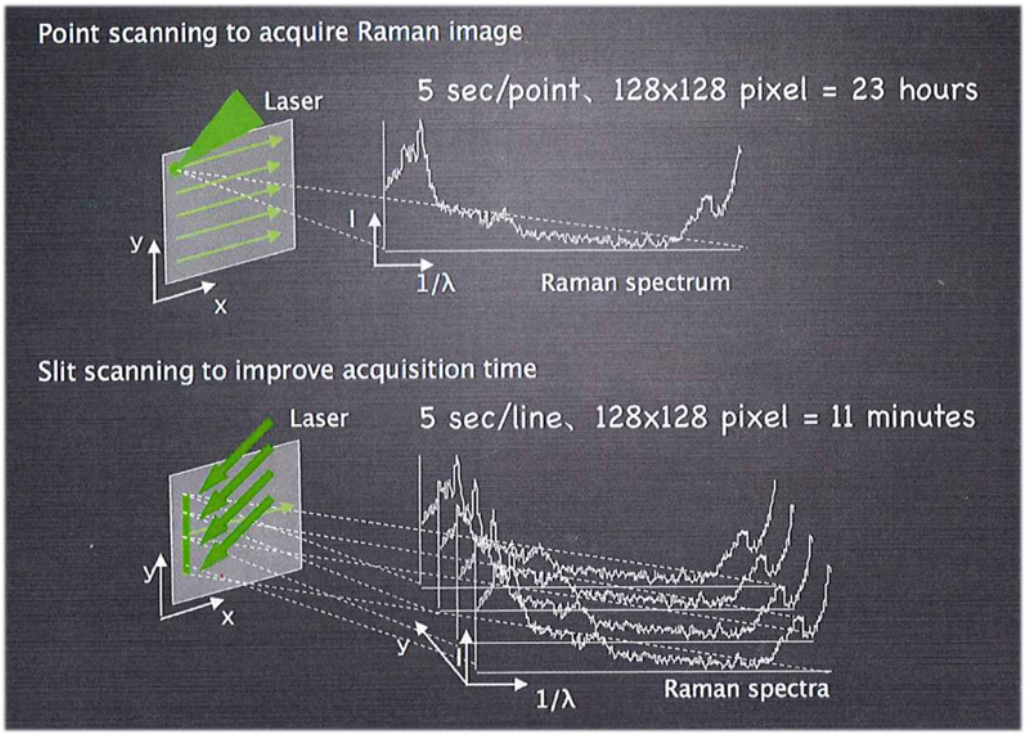


**Figure 21:** Difference between conventional (point scanning) and fast (slit scanning) Raman measurement, courtesy of Associate Professor Katsumasa FUJITA.

The main difference between Osaka's Raman setup from conventional Raman lies in the shape of the excitation laser ; standard Raman is based on point-shaped illumination while the Fast Raman method developed in Osaka uses line-shaped illumination. Thus, as emphasized Figure.21, the slit scanning setup is able to acquire up to 400 spectra in one exposure, resulting in a $\approx 400$ times faster imaging rate than conventional exposures. This considerable improvement makes it possible to observe living samples with imaging times of several minutes, which allows for example the observation of the change in molecular distribution of lipids, proteins or cytochrome-*c*.

### Motivation for wavenumber selection

Osaka's group is currently developing a method that can focus on specific wavenumbers and perform simultaneous measurements at different point in the sample:

→ By installing a special programmable mirror in the path of the laser, it is possible to measure different arbitrary points at once instead of a single line. However, due to the positional relationship of the measured points, the spectra are overlapping, and the information is lost at some wavenumbers. By limiting the number of wavenumber we want to measure, we can increase the number of points that are measured at one time, which makes the pathological diagnosis faster.

For cancer diagnosis, the CCD camera of the microscope currently record the wave numbers within the range $[362\,\mathrm{cm}^{-1},\ 3087\,\mathrm{cm}^{-1}]$. If it is possible to classify cancer and non cancer sufficiently with some wavenumbers, this would have two advantages:

- **Resolution:** The length in the wavenumber direction reflected on the CCD could be reduced by eliminating the wavenumber in the ranges that are not needed.
- **Speed:** The disease diagnosis could be sped up by measuring a small number of wavenumbers because it makes possible that more points are measured simultaneously.

### III.2.2 Raman data and pre-processing

The project is currently focusing on the diagnosis of Follicular thyroid cancer because is relatively difficult to visually diagnose, and there is a motivation to diagnose it using Raman spectroscopic imaging. The data provided by Osaka group are 3 dimensional matrices $M$ ($n_x \times n_y \times n_\omega$) containing a spectrum of $n_\omega$ wavenumbers at each position $(x, y)$. More specifically, 5 Cancer and 4 non cancer images has been provided, each with $n_x = 400$, $n_y = 240$ and $n_\omega = 840$. An example of a cancer measurement is shown below:

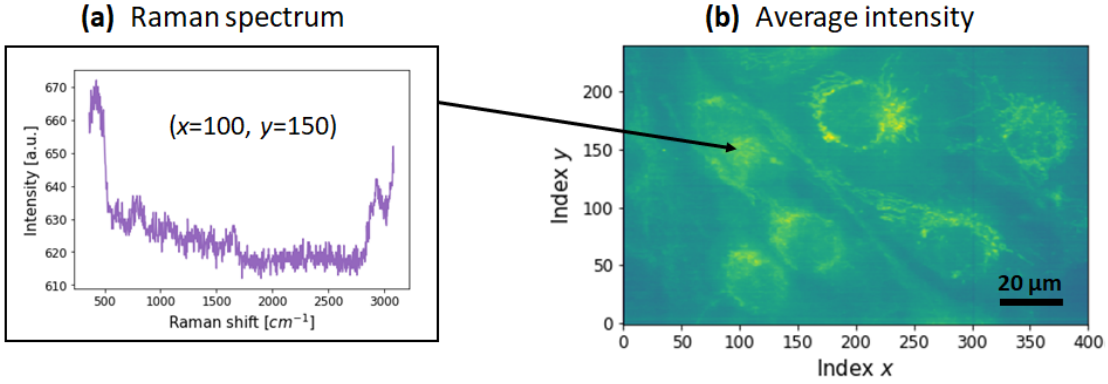


**Figure 22:** (a) Original Raman spectrum at position ($x = 100$, $y = 150$).
(b) Average intensity at each position ($x$,$y$) over all the wavenumbers.

Raman data originally contains strong noise, a considerable fluorescence background due to water and glass, as well as cosmic rays and measurement errors, thus an advanced pre-process is needed to obtain usable spectra. A pre-processing method of the Raman data from Osaka group has been implemented in `Python` by Assistant Professor Koji TABATA, and proceed as follow:

- **Noise:**
  The intensity of each wave number in the Raman spectrum is following a Poisson distribution, which creates a considerable noise. Denoising is performed by keeping the first 20 components of the measurement matrix $M$ after its Singular Value Decomposition (SVD).

- **Fluorescence background:**
  All the spectra contain a relatively large background do to glass and water Raman autofluorescence, the baseline removal is done with a recursive polynomial fitting, as described in [27].

- **Cosmic rays:**
  Some measured spectra in the matrix $M$ are altered by cosmic rays, they can be detected when the intensity at a specific wavenumber is at least 8 times the standard deviation higher than the average intensity at that wavenumber: $I(\Delta\omega) \geq 8\sigma(\Delta\omega) + \mu(\Delta\omega)$.

- **Differentiation of Cells and Backgrounds:**
  Within a Raman image, only a part of the spectra is actually from the cell, while the other spectra are from the background. We can differentiate them by looking at the wavenumber $\Delta\omega = 1450\,\mathrm{cm}^{-1}$, this region reflects the existence of $CH_2$, which is more present in cells region than background region.

### Pre-processed data

Processed spectrum as well as Cell / Background area of the original data (Figure.22) are plotted below. The figure highlight the efficiency of the data pre-processing, and we can clearly distinguish on the processed spectrum various peak, each corresponding to a specific molecule.

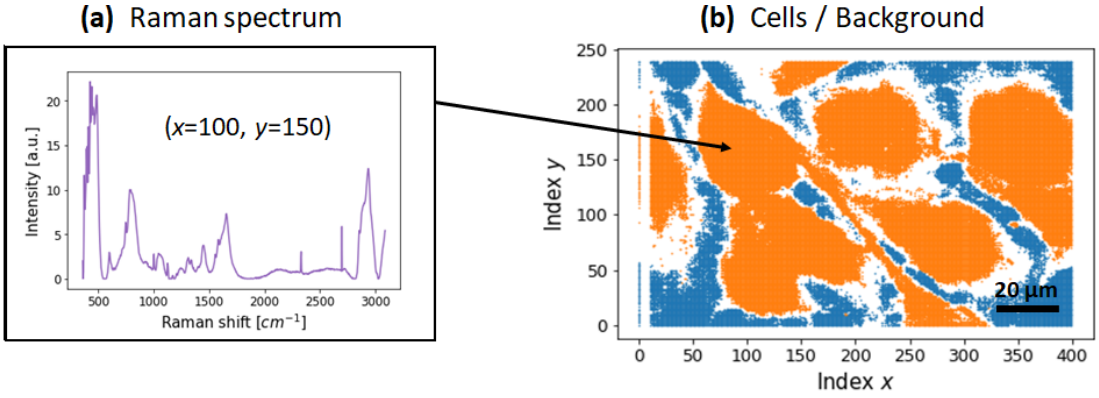


**Figure 23:** (a) Processed Raman spectrum at position ($x = 100$, $y = 150$).
(b) Cells (orange), Background (blue) and Undetermined (white) area in the Raman image.

To diagnosis diseases, we are only interested in the spectra from the cells, and not from the background. After having removed measurement error, cosmic rays and background on the 9 provided Raman images, we are left with **160 403** cancer cell spectra and **148 285** non cancer cell spectra, each with **840** wavenumbers.

### Cancer vs non Cancer

Before involving any statistical analysis or machine learning to the problem, the first thing we could think of is to look at the different spectra from cancer cells and non cancer cells to see if there is any difference that we can see to the naked eye. The figure below plot the average spectrum of all the cancer spectra and non cancer spectra, and there is no relevant difference that can be observed.

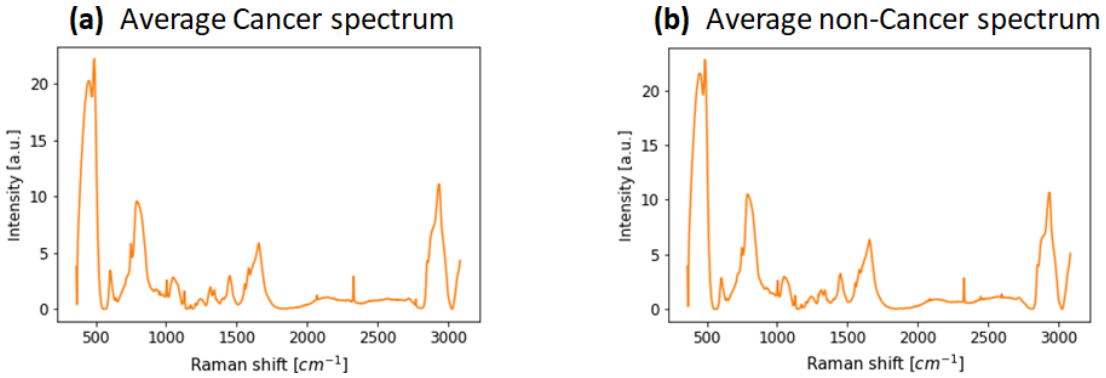


**Figure 24:** Average spectrum over all the cells position for (a) cancer cells and (b) non cancer cells, no clear difference can be seen from naked eye.

By training a machine learning algorithm with all the cancer and non cancer spectra, one can easily obtain accuracy higher than 99 % by using for example a Recurrent Neural Network. However training the algorithm with the 840 wavenumbers is relatively heavy, and as explained in the previous section, we want reduce the number of features to speed up the Raman measurements.

## III.3 Feature selection on Raman spectra

Feature selection on the Raman spectra from Osaka Group has been previously attempted by Assistant Professor Koji TABATA. He applied the Recursive Feature Elimination algorithm [28] with Guni feature importance of random forest as the criterion for removing features. Since Raman data has a large number of features and a large learning cost, the number of features has been reduced by 20 % at each iteration. The wavenumbers $\mathbf{1410\,cm^{-1}}$ and $\mathbf{1480\,cm^{-1}}$ has been obtained, however it is believed that at better set of wavenumbers can be obtained using more advance feature selection algorithms.

In this section, we aim at finding the most relevant set of wavenumbers for cancer diagnosis with Raman spectra. We apply different feature selection approaches and then compare their performances.

### III.3.1 The high features redundancy

Before applying feature selection to the Raman spectra, it is important to have an overview of the problem by studying redundancy within the wavenumbers, it will give us a better understanding of the question, and also some insights about how adapted some feature selection algorithms would be.

#### Mutual Information

In Shanon's Information theory, to quantifies the "amount of information" obtained about one random variable $X$ through the other random variable $Y$, one can calculate the Mutual Information (MI) between the two continuous variables, defined as follow:

$$MI(X,Y) = \iint_{X,Y} p(x,y) \log\left(\frac{p(x,y)}{p(x)p(y)}\right) dx\, dy$$

where $p(x,y)$ is the joint probability density function of $X$ and $Y$, and $p(x)$ and $p(y)$ are the marginal probability distribution functions of $X$ and $Y$ respectively. We can also define a normalized version of the mutual information as [29]:

$$I^*(X,Y) = \sqrt{1 - e^{-2\times MI(X,Y)}}$$

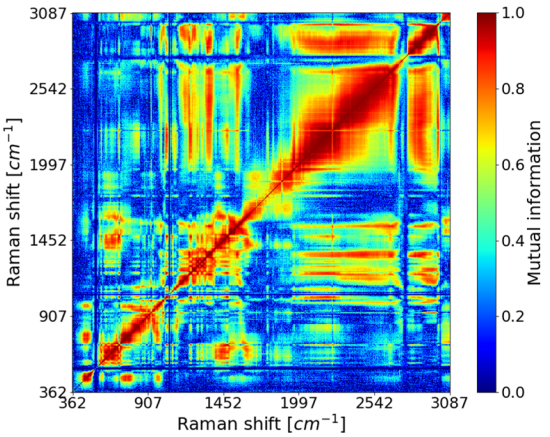


**Figure 25:** Estimated normalized Mutual information between wavenumbers in the Raman spectra, using 2500 examples. By definition, the mutual information of a wavenumber with itself if 1.

$I^* = 0$ means that $X$ and $Y$ are independent, while $I^* = 1$ indicates perfectly correlated variables. The mutual information for continuous target variables is estimated with a nonparametric methods based on entropy estimation from $k$-nearest neighbors distances [30] implemented in `Python` (`sklearn` library), and the calculation is performed between each wavenumbers using 2500 uniformly selected spectra from the original training set (we cannot use the full training set due to the high computational cost of mutual information estimation), results are plotted Figure.25.

The figure clearly emphasizes an important mutual information between some wavenumbers (close to 1), this means that, by knowing the value of one wavenumber, one can get information about the value of other wavenumbers. This figure also indicates that a relatively important part of the wavenumbers in the spectra are highly redundant.

### Information clusters

Now that we have calculated the mutual information between each wavenumber, we can regroup the wavenumbers within different information clusters, where two wavenumbers belonging to the same cluster would have a mutual information relatively close to one. We need to define a distance where two wavenumbers are at distance zero if and only if they are perfectly correlated. We set the distance between two wavenumbers $\omega_a$ and $\omega_b$ as follow:

$$d(\omega_a, \omega_b) = 1 - I^*(\omega_a, \omega_b) \tag{17}$$

There exist a large panel of unsupervised clustering algorithm, however as we are dealing with an unknown target number of clusters and an unusual distance metric, the possibilities are restrained. The Hierarchical Agglomerative Clustering (HAC)[31] method is well adapted to our problem because it can be used for any distance metric, gives different partitioning depending on the level-of-resolution we are looking at, and also have the advantage to be deterministic (always output the same result). The algorithm starts by treating all wavenumbers as singleton cluster, and then proceeds by merging pairs of clusters recursively until all have been merged into a single cluster that contain all wavenumbers. In order to decide which clusters should be combined, a measure of dissimilarity between sets of observations is required. For our problem, we define the distance between two clusters $A$ and $B$ as the maximum distance between their elements, this is known as the complete-linkage clustering criterion.

$$d(A, B) = \max\{d(\omega_a, \omega_b) \mid \omega_a \in A,\ \omega_b \in B\} \tag{18}$$

The HAC `Python` implementation of the `sklearn` library is used, and the clustering is performed for a threshold distance within the clusters of $d_{\max}$ (Figure.26). By setting this threshold distance, we make sure that all the wavenumbers within a cluster $A$ has a normalized mutual information greater than $I^*_{\min} = 1 - d_{\max}$ between each other.

$$\forall A \in Clusters, \quad \min\{I^*(\omega_{a_1}, \omega_{a_2}) \mid (\omega_{a_1}, \omega_{a_2}) \in A^2\} \geq I^*_{\min} \tag{19}$$

Figure.26 plot the HAC results for a threshold distance of $d_{\max} = 0.5$, and 63 clusters are obtained. However, by looking at the dendrogram Figure.26(b), it becomes clear that most of the clusters contains very few wavenumbers and thus are irrelevant. As highlighted on the Figure.26(c), there is actually few clusters that contains a relatively important number of wavenumber (as an example, only 8 clusters contain more than 30 wavenumbers). As expected, the information clusters mostly corresponds to wavenumbers which are close to each other in the spectra, but there is also some interesting relationship between wavenumbers that are far away to each other - cf. blue and green clusters in Figure.26(a), a deep knowledge of living cell Raman spectroscopy would be necessary to understand how these wavenumbers are related.

A more detailed study of the HAC result is conducted in appendix C.2, clusters are studied at different levels of precision and a video is also presented.

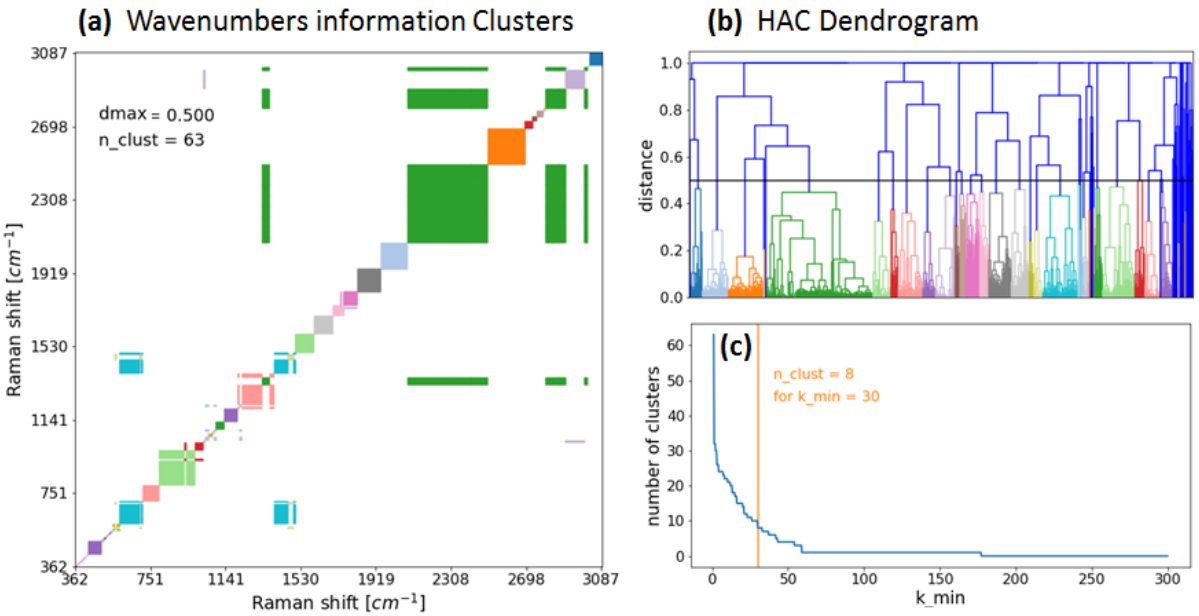


**Figure 26:** Result for the HAC algorithm with a threshold distance of $d_{\text{max}} = 0.5$. (a)The obtained information clusters are emphasized in different colors, (b)the corresponding dendrogram is plotted to highlight the cluster hierarchies, (c)the number of clusters with a size larger than $k_{\text{min}}$ as a function of $k_{\text{min}}$ is also plotted to highlight the relatively large number of irrelevant clusters. As an example (in orange), only 8 clusters among the 63 contains more than 30 wavenumbers.

One advantage of HAC is that we can choose the level-of-resolution we want to look at, but it is important to highlight that the threshold distance $d_{\text{max}}$ has been chosen arbitrary. The dendrogram presented above shows that there is no clear way to chose the appropriate threshold distance, it is because the merging behavior highly vary depending on the area in the spectrum. As for further research, one could perform sub-clustering of the most relevant clusters (e.g. the orange one on the dendogram) by eliminating the noisy wavenumbers that have a low mutual information with mostly all other wavenumbers (e.g. the right part in the dendogram).

Another remark is that the HAC algorithm merges the clusters in a greedy manner, which indicates that there might be other possible wavenumber arrangement that are more efficient and thus split the wavenumbers in less clusters. This mean that two wavenumbers belonging to different clusters are not necessary at a distance higher than $d_{\text{max}}$, and that the dendogram shown above should not be granted as an absolute and/or optimal clustering result.

### III.3.2 Simple filtering approaches

As a first approach for our feature selection problem, we apply simple filtering methods (see I.2) because it is computationally cheap and sometimes perform very well on specific problems. However, Since filtering approaches rank the features individually, they usually poorly account for the features redundancy ; and as explained in III.3.1, Raman spectra contain a relatively high number of redundant features. Thus we expect filtering methods to perform poorly, but it still gives us a good benchmark to compare with more advanced algorithm.

Two relatively commonly used filtering methods are the Fisher scoring [23] and ReliefF scoring [32] algorithms. The first one relies on Fisher information theory while the second one is based on the identification of feature value differences between nearest neighbor instance pairs, ReliefF scoring is also one of the rare filtering methods that account for inter-feature correlations. Because the two algorithms use very different approaches for the feature scoring, it is likely that they will lead to different feature rankings. Both algorithms are implemented in `Python` with the `skfeature` library [33], and the results are plotted below. We can see on the graph that the result from each method are relatively different

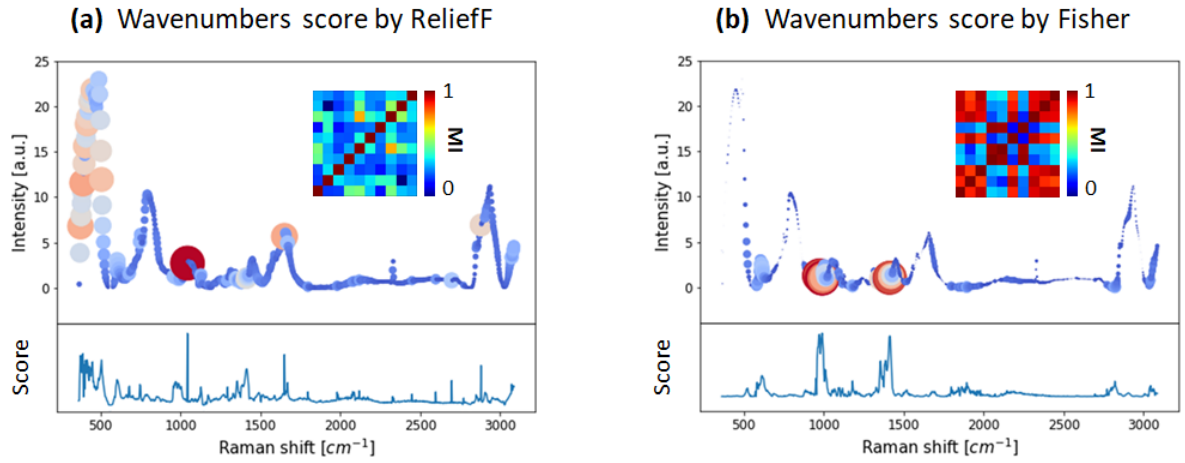


**Figure 27:** Wavenumber scores obtained by (a)Relieff and (b)Fisher scoring algorithm. The upper figure scatter a typical Raman spectra and the size/color of the markers represent the wavenumber score, the normalized mutual information between the top 10 ranked features is also plotted.

The normalized mutual information between the features selected by each methods is also plotted. While Fisher scoring algorithm selected features from only two clusters and thus many highly redundant features, the mutual information between the features from ReliefF scoring is remarkably low. This however does not mean that the features selected by the algorithm are good features, indeed the results presented in part III.3.4 indicate that most the the features selected by ReliefF are irrelevant.

### III.3.3 Greedy-SR and FUSE-2 results on Raman spectra

In this section, we apply the two previously presented feature selection algorithms in part II to the Raman spectra. The original training set contains around 300 000 Raman spectra, and it is computationally too heavy to run the feature selection algorithms on the full training set. Thus, to involve training times on the order of 1 h, we reduce the training set to $n = 5000$ examples. As explained in I.1, a training set involving around 5 to 10 times more example than the number of features is unlikely to be overfitted. $5000/840 \approx 6$ so the risk of overfitting is relatively low.

After the feature subset selection, the importance of each selected feature $f$ is calculated from the gain in the $k$-NN error rate when the feature $f$ was added to its parent node $F_p$ in the feature lattice. We chose to use a ratio instead of a difference because we want for example an improvement from 98 % to 99 % to be considered significant:

$$Score(f) = \frac{1 - V(F_p \cup \{f\})}{1 - V(F_p)} \tag{20}$$

Greedy-SR is run with 1 iteration per child without subsampling of the training set for the reward calculation because it is not too heavy computationally. FUSE-2 is run with the same parameters as described in section II.3, and results are displayed Figure.28.

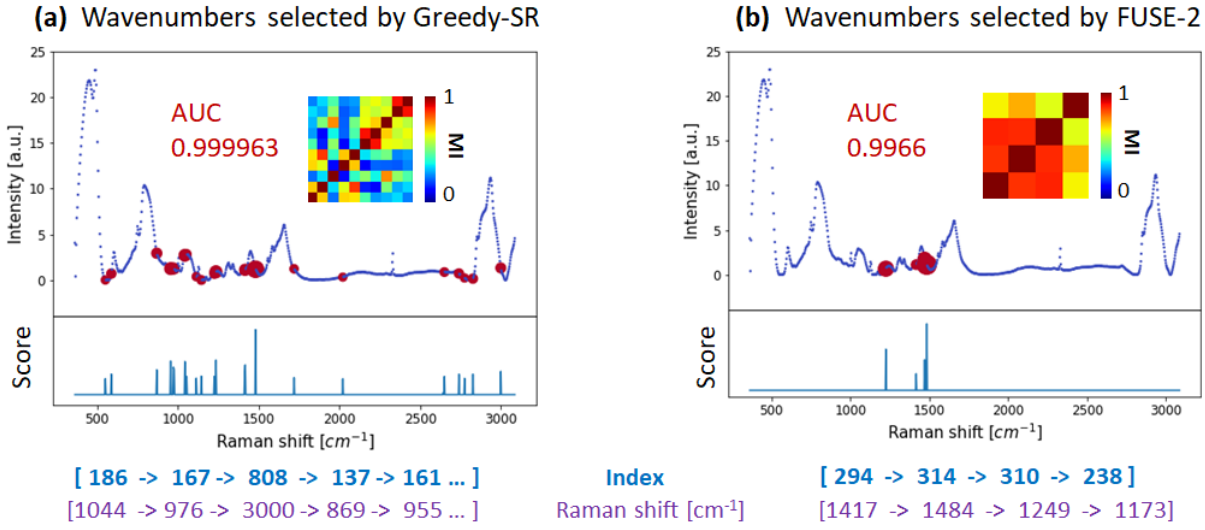


**Figure 28:** Results of the (a) Greedy-SR and (b) FUSE-2 algorithms on the Raman spectra, the circle size correspond to the feature importance calculated from the gain in the $k$-NN classifier error rate. The normalized mutual information between the first 10 features selected by Greedy-SR as well as the one selected by FUSE-2 is also plotted.

The first thing we observe is that the Greedy-SR algorithm selected much more features (26 features) than FUSE-2 (4 features). However the obtained reward (0.999963) is considerably better than the one obtained by UCT approach (0.9966). This can be explain by the fact that FUSE-2 does not explore at higher depth because the (linear) reward improvement is not considered important enough by the search ($0.999963 - 0.9966 = 0.0034$).

Even though all methods selected different features, it looks like features tends more to be selected in the $[900\,\mathrm{cm}^{-1}, 1600\,\mathrm{cm}^{-1}]$ wavenumber area. One can also note that the first two features selected by FUSE-2 are $1417\,\mathrm{cm}^{-1}$ and $1484\,\mathrm{cm}^{-1}$, which are the one previously found by assistant Professor Koji TABATA (III.3).

Regarding the mutual information, the results are relatively surprising. Figure.28(b) shows that the features selected by FUSE are strongly correlated (they have a normalized mutual information higher than 0.7). Thus, even though a major part of the information contained in these features are redundant, it seems like the remaining information, which is specific to each feature, is relatively useful for cancer diagnosis. Similarly, the feature set selected by greedy strategy also contains non negligible redundancy.

### III.3.4 Comparison of the algorithm's performance

To compare the performance of the feature selection methods described above, we train a 5-NN and a Random Forest classifier with the obtained feature sets (the `python` implementation of the `sklearn` library is used). For the training, the same 5000 examples as the one used for feature selection are used, and predictions is then performed on a test set containing 10 000 other examples. The results are displayed Figure.32.

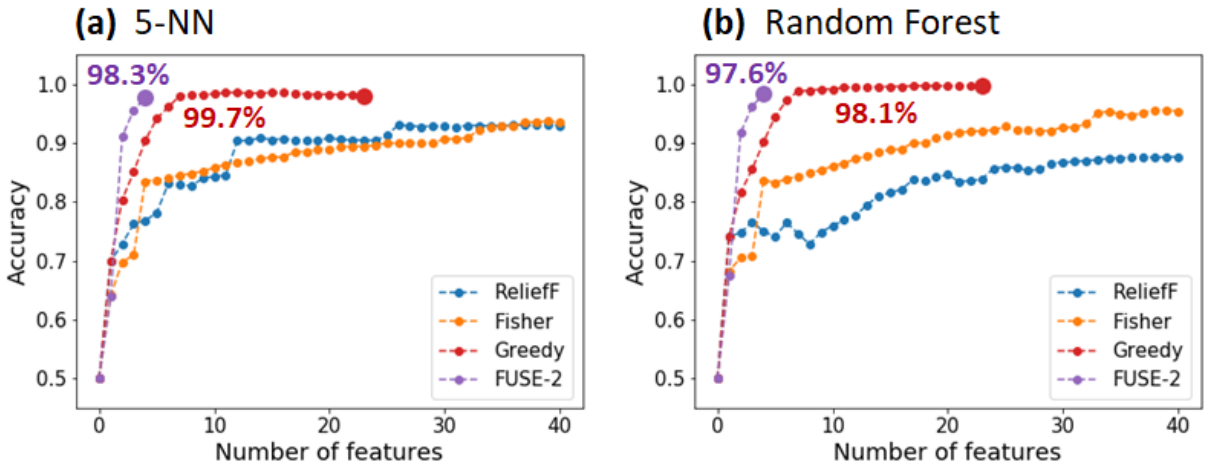


**Figure 29:** Accuracy obtained with classifiers trained with the selected features from different methods on the Raman test set (10000 examples).

As expected, the 5-NN classifier globally performed better than Random Forest (for Greedy-SR and FUSE-2), because these methods had selected features specifically to optimize a reward which is based on a 5-NN classifier. Globally, these results are quite remarkable, because they indicate that it is possible to diagnosis follicular thyroid cancer with more than 98 % accuracy by using only 5 wavenumbers from the initial 840 wavenumbers contained in the Raman spectra. As for the performance of the Greedy approach compared to the UCT approach, two major differences can be observed:

- While Greedy strategy only consider the local optimum choice at each node, the UCT approach is able to find better *combo* of features, resulting in a higher accuracy for the same number of feature (for 5 selected features, FUSE-2 perform 98 % accuracy while Greedy-SR only achieve 90 %).
- FUSE-2 is stopping after 5 features because it is limited by the linear overview of the feature subset evaluation (for example, it consider that going from 99 % to 99.5 % is a small improvement whereas it is actually considerably significant because it improves the error rate by a factor of two). To solve this problem, one could think of applying a function (like the logit function) to the reward so that such a small difference will be enhanced when the reward becomes close to 1.

# IV Conclusion & Perspectives

### Feature selection

We formalized feature selection as a reinforcement learning problem, and presented different ways to tackle the exploration/exploitation trade-off in the feature lattice. An improved version of the original FUSE algorithm [10], which rely on the UCT strategy, has been introduced as FUSE-2. We also presented a greedy strategy with successive elimination of irrelevant child as Greedy-SR. Both algorithms has been implemented in `C++` and tested on various dataset, they have proven remarkably more efficient than major filtering approaches.

The strategies presented in this thesis currently relies on a feature set evaluation metric based on naive $k$ nearest neighbors search. As for further research, one can consider using alternative $k$-NN search [19, 20] to gain computational time, or using a different evaluation metrics such as the Correlation-based Feature Selection (CFS) [8, 9] as a different approach for comparisons with the current evaluation method. One can also implement the original FUSE algorithm [10] to see how it performs compared to FUSE-2 on benchmark datasets.

### Raman spectroscopy

A detailed study of the relevant wavenumbers in Raman spectra for follicular thyroid cancer diagnosis was also presented. Redundancy analysis within the wavenumbers and information clustering was performed using Information theory, and high correlations between many wavenumbers have been observed. Wavenumbers selection was achieved with FUSE-2 and Greedy-SR, and both methods have shown that less than 7 wavenumbers are enough to diagnosis cancer with 98 % accuracy, which is a remarkable result given the 840 initial wavenumbers contained in Raman spectra. This will considerably speed up the Raman measurement because it makes possible that more points are measured simultaneously by measuring only the relevant wavenumbers.

Due to computational limitations, the wavenumbers selection was performed on a subdataset of 5000 instances whereas $\approx$ 300 000 Raman spectra are available. By optimizing the algorithms computational complexity, one could improve the performance of wavenumbers selection by training the algorithms on larger dataset. It may also be possible to reach higher accuracy by applying the inverse sigmoid function to the computed reward in the FUSE-2 algorithm. Finally, changing the value of $k$ in the feature set evaluation may also significantly improve the models predictions.

### *Personal assessment*

*For 3 month, I have been able to immerse myself into the field of machine learning. Being from a physics background, most of the concepts described in the first half of this thesis were new to me, and this experience has proven to be an excellent introduction to the research in theoretical computer science. Furthermore, not only this internship taught me a lot about machine learning and programming, it also gave me the opportunity to take part into a remarkably dynamic and ambitious project. Because wavenumber selection on Raman spectra is an interdisciplinary problem, I found my physics background particularly useful, and I developed through my work valuable multi-tasking skills.*

*I would like to add that Machine Learning could be in general a remarkably efficient tool for many scientists, and I am glad to see that the intersection between Physics and Machine Learning seems to be growing considerably. I do believe that Machine Learning will become an unavoidable tool for most physicists and biologists, and I am looking forward to take part into the artificial intelligence revolution of our century.*

# A Complements on classifiers

## A.1 Confusion matrix, Area Under Curve and Accuracy

From the confusion matrix of a classifier (Figure.30), one can compute the True Positive Rate (TPR) and the False Positive Rate (FPR). Intuitively, the TPR correspond to the proportion of positive data points that are correctly considered as positive, with respect to all positive data points. On the other hand, the FPR indicates the proportion of negative data points that are mistakenly considered as positive, with respect to all negative data points.

The AUROC, commonly written as AUC, refers to the Area Under the Receiver Operating Characteristic Curve, which plot the False Positive Rate (FPR) as a function of the True Positive Rate (TPR) as we change the discrimination threshold of the binary classifier. The AUC is equal to the probability that the classifier will score a randomly chosen positive example higher than a randomly chosen negative example, making it a relevant indicator of its performance:

$$\text{AUC} = P\Big(\, s(x+) > s(x-) \,\Big)$$

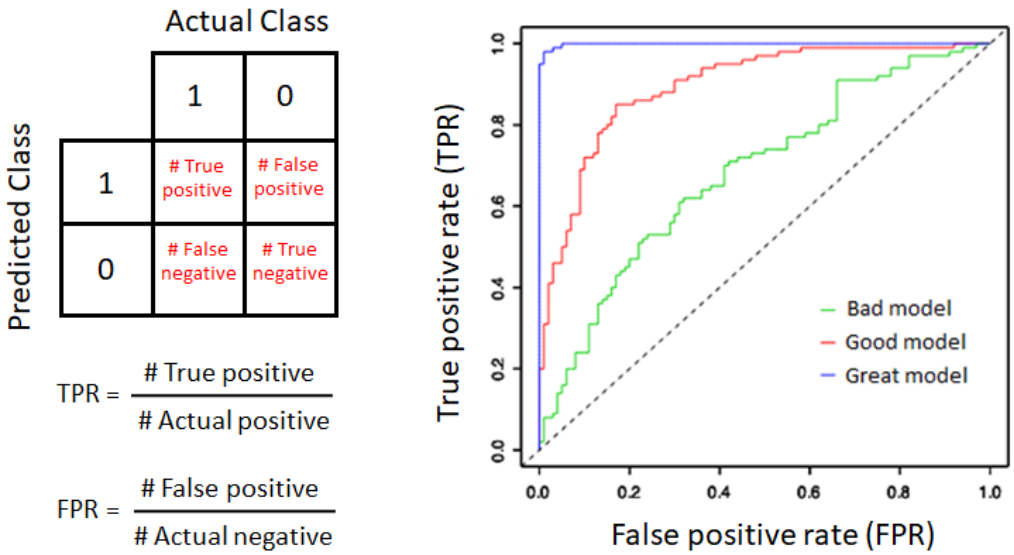


**Figure 30:** Confusion matrix and typical ROC curve of different classifiers.

The ROC curve of a random classifier is a straight line going from the lower left to upper right, which leads to AUC = 0.5. In opposition, the ROC curve of a perfect classifier will pass to the upper left point and thus gives AUC = 1.

Another good indicator of the performance of a binary classifier is given by its Accuracy (ACC), that measures the percentage of points correctly classified for a given threshold:

$$\text{ACC} = \frac{TP + TN}{TP + TN + FP + FN}$$

Although ACC and AUC measure different things and thus are complementary to each other, ACC depends on the chosen discrimination threshold, whereas the AUC considers all possible thresholds. Because of this, AUC is often preferred as it provides a broader view of the performance of classifier. One shoudl also note that their is many other ways to evaluate binary classifiers, such as Precision and Recall.

### Implementation details

Mathematically, the AUC of a binary classifier $s$ on a training set $\mathcal{L}$ containing $n$ labeled examples $(x_i, y_i)$ where $y_i \in \{0, 1\}$, is calculated as follow:

$$\text{AUC} = \frac{\left|\left\{(x, x') \ \in \mathcal{L}^2, \ \ s(x) < s(x') \ , \ \ y < y'\right\}\right|}{\left|\{(x, x') \ \in \mathcal{L}^2, \ \ y < y'\}\right|} \tag{21}$$

In practice, to compute the AUC, we use an array of pairs $\langle$example - label$\rangle$ denoted $\mathcal{L} = \langle x_i, y_i \rangle_i$ sorted by their score so that $i < j \Leftrightarrow s(x_i) < s(x_j)$. Sorting $\mathcal{L}$ has a complexity $O(n \log(n))$ (with $n = |\mathcal{L}|$), and the following algorithm, implemented in `C++` with the `std::multimap` container, then compute the AUC with complexity $O(n)$:

**Algorithm** AUC

**function** AUC $(s)$

Initialize $U = 0$, $TP = 0$

**for** Elements $\langle x, y \rangle$ in $\mathcal{L}$ from lowest to highest score **do**

**if** $y = 1$ **then**

$TP \leftarrow TP + 1$

**else**

$U \leftarrow U + TP$

$$\text{AUC} = \frac{U}{TP \times (n - TP)}$$

AUC computation with a sorted array of examples, $U$ stands for *Under curve* and $TP$ for *True Positive*.

The variable $U$ is counting the number of examples under the curve, while $TP$ indicates the number of true positives for a given discrimination threshold, which is increasing as we ramp higher in the sorted $\mathcal{L}$ container. The discrimination threshold is maximal when we reach the last element in the container, making the number of true positive (TP) equals to the number of actual positives, and thus leading to the denominator in the AUC final formula.

## A.2 The choice of $k$ in the $k$-NN Classifier

In general, a larger $k$ suppresses the effects of noise, but makes the classification boundaries less distinct, therefore the best value of $k$ for a $k$NN classifier is highly data-dependent. One can also note that $k$ is often chosen as an odd number to avoid ambiguity in the model predictions. The study in this report is conducted with $k = 5$, but a rigorous test on a cross validation set with different values of $k$ shall be performed to obtain optimal results on the model predictions.

# B Practical details

## B.1 The C++ implementation

We want to perform feature selection on a feature set $\mathcal{F}$ containing $n_f$ features. This section describes the implementation choices to deal with the large feature lattice and the high branching factor.

**Feature subset $F$:**
We have chosen to represent each feature subset with a boolean array $F$ of length $n_f$, where $F[f] = 1$ if $f \in F$ and 0 otherwise ; the `boost::dynamic_bitset` container is used. We then implement a `class Node` to store all the important information about the node $N_F$ (part II.2.4).

**Feature lattice $\mathcal{T}$:**
Since dataset regularly contains about $n_f = 1000$ features, $2^{1000}$ nodes are involved in the feature lattice. It is practically impossible to use an array of that length for our implementation because it would require a memory of at least $2^{1000}$ bits $\approx 10^{300}$ bits. One can note that it is even orders of magnitude higher than the maximum theoretical information than can be contained in one $m^3$, which is currently estimated as $10^{70}$ bits.

The lattice $\mathcal{T}$ is therefore implemented as a dynamic container `std::vector`, which contains only the root node $N_\varnothing$ at the beginning of the search but then add a new node when it is visited for the first time. The complexity of finding an element in a `std::vector` container scales as $O(n)$, thus it becomes longer to find a node in $\mathcal{T}$ as its size increase. Because we need to know the position of the child nodes at each step in the UCT phase, it becomes computationally heavy for big containers.

We then introduce a hash function which allows us to find an element in $O(1)$ (Figure.31) (the hash function is deterministic and always return the same hash value for the same input). The std `std::unordered_map` container is used. The input of the hash function is the boolean array $F$, and the output is a bucket containing the address of $F$ in the tree $\mathcal{T}$. The node $N_F$ can then be accessed at $T[hash(F)]$.

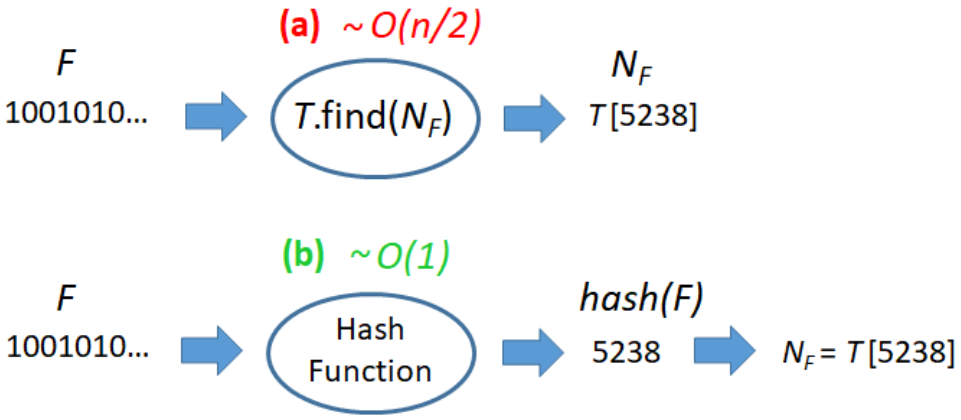


**Figure 31:** Naive way (av. complexity $O(n/2)$) to find a node $N_F$ in the lattice $\mathcal{T}$ vs. implemented method using a hash function (av. complexity $O(1)$).

## B.2 Complexity analysis

This section provides a detailed complexity analysis of the Greedy-SR and FUSE-2 algorithm when performing feature selection on a feature set $\mathcal{F}$ containing $f$ features (the notation $f$ is used instead of $n_f$ for better readability). The training set size is denoted $|\mathcal{L}| = n$ and the subsample size $|\mathcal{V}| = m$

### Reward

Both algorithms rely on the reward computation as it is calculated at each iteration. For a feature set evaluation at depth $d$ in the feature lattice, the complexity is the following:

- **Subsampling**: This step simply uniformly pick $m$ random example from the original training set $\mathcal{L}$, the complexity is then $O(m)$.
- **$k$-NN predictions**: To make a prediction, $n$ distances has to be computed, and since $d$ dimensions are involved, the complexity is $O(nd)$. After that, the $k$ nearest neighbors are found with the quickselect algorithm in $O(n)$. The process needs to be repeated for $m$ examples, leading to the final complexity of $O(nm(d+1))$.
- **AUC computation**: The AUC is then computed with the method described in part A.1, which has a complexity of $O(mlog(m) + m)$.

### FUSE-2

For one iteration in the FUSE-2 algorithm searching at depth $d$ in the feature lattice:

- **UCT phase**: At each step in the UCT phase, the UCB score of $f$ features is computed, complexity $O(fd)$.
- **Adding a new node**: When a new node is added, we check for its parent and children to add their addresses if they already exist in the tree, complexity $O(f)$.
- **Random phase**: On average, the same number of feature is added inr each random phase (defined by parameter $q$), complexity $O(1)$
- **Reward computation**: Overall dominated by the $k$-NN prediction so complexity is $O(nm(d+1))$
- **Backpropagation**: We update all the ancestors as described in part II.2.4, The number of nodes to update scales as $O(2^d)$, each containing a $\ell-$RAVE score array of $f$ features. The complexity is then $O(2^d f)$.

### Greedy-SR

At each stage in the greedy search, the reward is computed $N$ times, and the depth of the search $d_i$ is increased by one at each stage, until the final feature set is selected at depth $d$. Thus the Greedy-SR algorithm has complexity:

$$\mathcal{C}_{\text{Greedy}} = \sum_{d_i=1}^{d} N \times O(nm(d_i+1)) = O\left(Nnm\,\frac{(d+1)(d+2)}{2}\right)$$

# C Complements on Raman

## C.1 Raman wavenumbers in living cells

The following table provides the correspondences between the peaks observed in the Raman spectra and the molecules contained in the living cells. It may give us some insight about why and how some wavenumbers in the Raman spectra could be related, as highlighted by the wavenumber information clusters found by HAC.

| $\Delta\omega$ [cm$^{-1}$] | Protein | Lipid | Nucleic acid |
|---|---|---|---|
| 666-684 | | | グアニン 呼吸振動 |
| 720-1060 | | $CH_2$ 横揺れ振動 | |
| 729, 731 | | | アデニン 呼吸振動 |
| 750, 753 | | | チミン 呼吸振動 |
| 784 | | | シトシン 呼吸振動 |
| 785 | | | ウラシル 呼吸振動 |
| 807, 836 | | | O-P-O 伸縮振動 |
| 830 | チロシン | | |
| 850 | チロシン | | |
| 1000 | フェニルアラニン | | |
| 1093, 1101 | | | $PO_2^-$ 伸縮振動 |
| 1175-1415 | | $CH_2$ 縦揺れ振動 | |
| 1230-1240 | アミド-III $\beta$ シート | | |
| 1240-1260 | アミド-III ランダムコイル | | |
| 1260-1300 | アミド-III $\alpha$ ヘリックス | | |
| 1257 | | | アデニン, グアニン |
| 1295-1305 | | $CH_2$ ひねり振動 | |
| 1339 | | | アデニン, グアニン |
| 1363 | トリプトファン | | |
| 1370-1385 | | $CH_3$ 対称変角振動 | |
| 1445-1475 | | $CH_2$ はさみ振動 | |
| 1449 | CH 変角振動 | CH 変角振動 | |
| 1460-1470 | | $CH_3$ 非対称変角振動 | |
| 1486 | | | アデニン, グアニン |
| 1577 | | | アデニン, グアニン |
| 1645-1660 | アミド-I $\alpha$ ヘリックス | | |
| 1660-1665 | アミド-I ランダムコイル | | |
| 1665-1680 | アミド-I $\beta$ シート | | |
| 1650-1665 | | C=C 結合 シス型 | |
| 1670-1680 | | C=C 結合 トランス型 | |
| 1669 | | | チミン C=O 結合 |
| | | | |
| 2845-2865 | | $CH_2$ 対称伸縮振動 | |
| 2860-2885 | | $CH_3$ 対称伸縮振動 | |
| 2915-2945 | | $CH_2$ 反対称伸縮振動 | |
| 2950-2970 | | $CH_3$ 非対称伸縮振動 | |

**Figure 32:** Raman shift measured in living cells with their corresponding molecule, from Osaka group.

## C.2 Further study of Hierarchical Agglomerative Clustering results

The Video.33 below emphasizes the clusters found by the HAC algorithm as the threshold distance $d_{\max}$ is changed. In addition to the analysis of part III.3.1, we observe the following:

- At distance $d_{\max} = 0.13$, we observe that some wavenumbers which are far away from each others in the spectrum already belong to the same cluster (cf. wavenumbers $500\,\mathrm{cm}^{-1}$ and $1500\,\mathrm{cm}^{-1}$). This indicates that these wavenumbers has a mutual information of at least 0.87.
- At distance $d_{\max} = 0.99$, some wavenumbers still belong to very small clusters (containing 2 or 3 wavenumbers) (cf. right part in the dendogram), meaning that they are completely uncorrelated with the whole spectrum.

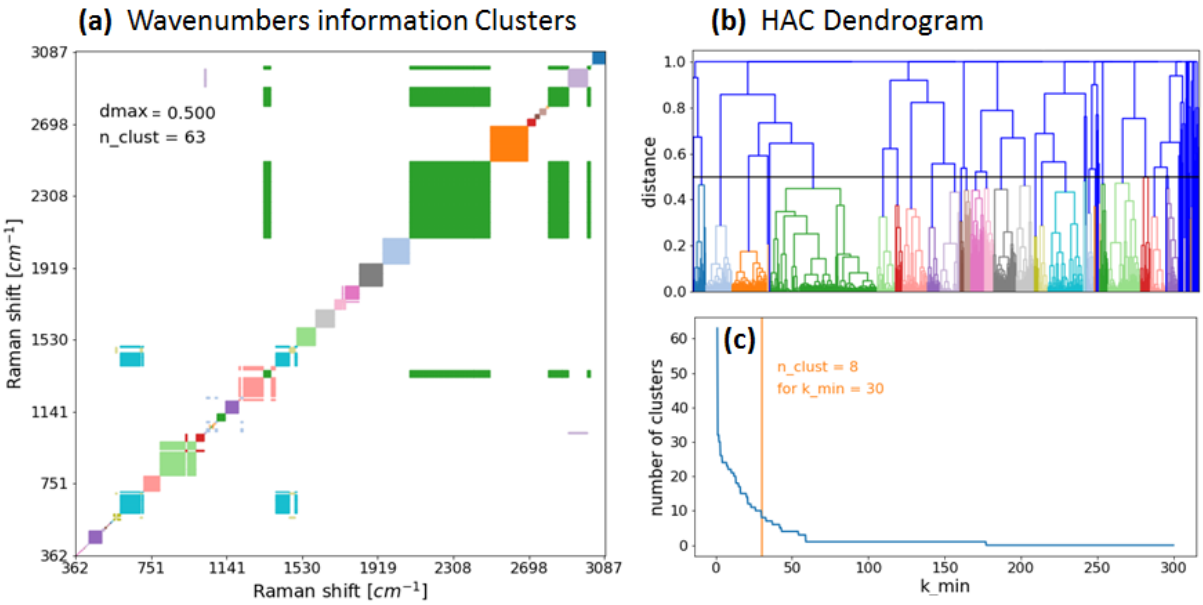


**Figure 33:** HAC results (part III.3.1) for different threshold distance $d_{\max}$.
*To watch the video, the last version of Adobe Reader (Mac or Windows) is needed, it is not possible on Linux.*

Figure.34 plots the number of obtained clusters as a function of the distance threshold $d_{\max}$. As expected, the number of clusters decreases when the distance threshold is increased. On the other hand, the number of clusters containing more than 30 wavenumbers reach a maximum of 11 clusters for distance $d_{\max} = 0.6$

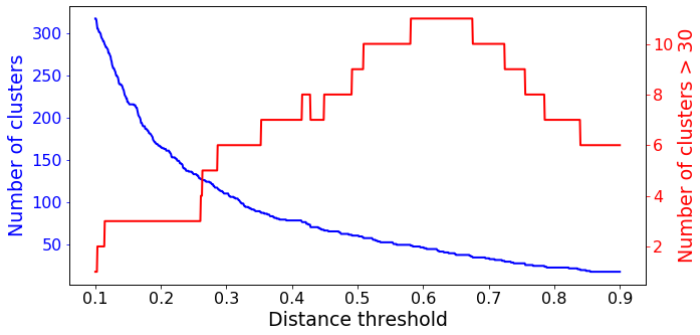


**Figure 34:** Number of clusters selected by HAC as a function of the threshold distance.

## Calendar

| Month(s) | Accomplished work |
|---|---|
| March | Paper review, FUSE implementation |
| April | Test on Benchmark dataset, Greedy-SR implementation |
| May | Feature selection on Raman, Mutual Information, writing Report |
| June | Information Clustering, writing Report, research Poster, Defense |

## Bibliography :


[1] Inês P Santos, Elisa M Barroso, Tom C Bakker Schut, Peter J Caspers, Cornelia GF van Lanschot, Da-Hye Choi, Martine F van der Kamp, Roeland WH Smits, Remco van Doorn, Rob M Verdijk, et al. Raman spectroscopy for cancer detection and cancer surgery guidance: translation to the clinics. *Analyst*, 142(17):3025–3047, 2017.

[2] Harvey Lui, Jianhua Zhao, David McLean, and Haishan Zeng. Real-time raman spectroscopy for in vivo skin cancer diagnosis. *Cancer research*, 72(10):2491–2500, 2012.

[3] Julia Powles and Hal Hodson. Google deepmind and healthcare in an age of algorithms. *Health and technology*, 7(4):351–367, 2017.

[4] Almar F Palonpon, Jun Ando, Hiroyuki Yamakoshi, Kosuke Dodo, Mikiko Sodeoka, Satoshi Kawata, and Katsumasa Fujita. Raman and sers microscopy for molecular imaging of live cells. *Nature protocols*, 8(4):677, 2013.

[5] Peter W Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM review*, 41(2):303–332, 1999.

[6] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436, 2015.

[7] Maria CERVERA. Exploring the biological plausibility of unsupervised machine learning algorithms. Master's thesis, EPFL, 2018.

[8] Mark A Hall. Correlation-based feature selection of discrete and numeric class machine learning. 2000.

[9] Mark Andrew Hall. Correlation-based feature selection for machine learning. 1999.

[10] Romaric Gaudel and Michele Sebag. Feature selection as a one-player game. In *International Conference on Machine Learning*, pages 359–366, 2010.

[11] Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2-3):235–256, 2002.

[12] William R Thompson. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3/4):285–294, 1933.

[13] Neha Gupta, Ole-Christoffer Granmo, and Ashok Agrawala. Successive reduction of arms in multi-armed bandits. In *Research and Development in Intelligent Systems XXVIII*, pages 181–194. Springer, 2011.

[14] Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *European conference on machine learning*, pages 282–293. Springer, 2006.

[15] Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1):1–43, 2012.

[16] Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2-3):235–256, 2002.

[17] Rémi Coulom. Efficient selectivity and backup operators in monte-carlo tree search. In *International conference on computers and games*, pages 72–83. Springer, 2006.

[18] Sylvain Gelly and David Silver. Combining online and offline knowledge in uct. In *Proceedings of the 24th international conference on Machine learning*, pages 273–280. ACM, 2007.

[19] V Ramasubramanian and Kuldip K Paliwal. Fast k-dimensional tree algorithms for nearest neighbor search with application to vector quantization encoding. *IEEE Transactions on Signal Processing*, 40(3):518–531, 1992.

[20] Malcolm Slaney and Michael Casey. Locality-sensitive hashing for finding nearest neighbors [lecture notes]. *IEEE Signal processing magazine*, 25(2):128–131, 2008.

[21] Isabelle Guyon. Design of experiments of the nips 2003 variable selection benchmark, 2003. URL = .

[22] Isabelle Guyon, Steve Gunn, Asa Ben-Hur, and Gideon Dror. Result analysis of the nips 2003 feature selection challenge. In *Advances in neural information processing systems*, pages 545–552, 2005.

[23] Robert I Jennrich and PF Sampson. Newton-raphson and related algorithms for maximum likelihood variance component estimation. *Technometrics*, 18(1):11–17, 1976.

[24] Tamiki Komatsuzaki. Development of accelerated measurement techonology for cell diagnosisby bridging single-cell raman imaging and information science. *CREST*
URL = `https://www.jst.go.jp/kisoken/crest/en/project/1111092/16815537.html`.

[25] Krzysztof Czamara, Katarzyna Majzner, Marta Z Pacia, K Kochan, A Kaczor, and M Baranska. Raman spectroscopy of lipids: a review. *Journal of Raman Spectroscopy*, 46(1):4–20, 2015.

[26] A Rygula, Katarzyna Majzner, Katarzyna M Marzec, Agnieszka Kaczor, Marta Pilarczyk, and M Baranska. Raman spectroscopy of proteins: a review. *Journal of Raman Spectroscopy*, 44(8):1061–1076, 2013.

[27] Chad A Lieber and Anita Mahadevan-Jansen. Automated method for subtraction of fluorescence from biological raman spectra. *Applied spectroscopy*, 57(11):1363–1367, 2003.

[28] Isabelle Guyon, Jason Weston, Stephen Barnhill, and Vladimir Vapnik. Gene selection for cancer classification using support vector machines. *Machine learning*, 46(1-3):389–422, 2002.

[29] Ivan Kojadinovic. On the use of mutual information in data analysis: an overview. In *Proc Int Symp Appl Stochastic Models Data Anal*, pages 738–47, 2005.

[30] Alexander Kraskov, Harald Stögbauer, and Peter Grassberger. Estimating mutual information. *Physical review E*, 69(6):066138, 2004.

[31] Daniel Müllner. Modern hierarchical, agglomerative clustering algorithms. *arXiv preprint arXiv:1109.2378*, 2011.

[32] Kenji Kira and Larry A Rendell. A practical approach to feature selection. In *Machine Learning Proceedings 1992*, pages 249–256. Elsevier, 1992.

[33] Jundong Li, Kewei Cheng, Suhang Wang, Fred Morstatter, Trevino Robert, Jiliang Tang, and Huan Liu. Feature selection: A data perspective. *arXiv:1601.07996*, 2016. URL = `http://featureselection.asu.edu/`.