**Université de Bordeaux**

**Collège Sciences et Technologie**


# Développement d'un outil de classification automatique de signaux neuronaux biologiques

## Rapport

**Auteurs:**

BLAIS Benjamin

COTTAIS Déborah

DE OLIVEIRA Lila

JOUAN Clément

THOUVENIN Arthur

Bordeaux
14-05-2018

# Sommaire

# Remerciements

Nous tenons à adresser nos remerciements aux personnes qui nous ont aidées dans la réalisation de ce projet.

En premier lieu, nous remercions André Garenne, maître de conférence à l'université de Bordeaux. En tant que client de ce projet, il nous a guidé dans notre travail et aidé à trouver des solutions concernant l'avancement du projet. Il nous a également fait découvrir et apprendre de nouvelles pratiques dans la conception et la réalisation d'un projet en bioinformatique ce qui était nouveau pour nous.

Nous remercions aussi, Marie Beurton-Aimar pour sa bienveillance dont elle a fait preuve en prenant le temps de suivre et d'orienter dans la bonne direction notre projet et d'avoir ainsi veillé au bon déroulement de ce dernier.

# Lexique

Description des différents paramètres donnés, leurs unités et acronymes, voir Figure 1 :

- *nClass* : Neuron Class, il existe des centaines de classes de neurones mais ici on utilisera seulement deux types identifiés comme 1 et 2.
- *IR* : Input Resistance (mOhm), cela correpond à une des lois d'Ohm : V = IR où V,I et R correspondent respectivement à une tension, un courant et une résistance.
- *RMP* : Resting Membrane Potential (mV), c'est le potentiel de la membrane mesuré au repos.
- *RH* : Rheobase (pA), l'intensité minimale de courant excitant qui permet de déclencher un potentiel d'action.
- *ST* : Spike threshold (mV), tension à laquelle le potentiel d'action est déclenché.
- *DTFS* : Delay to first spike (ms), il correspond au délai entre lequel on émet un courant dans un neurone et le moment où l'on observe le premier pic de réaction.
- *SA* : Spike ampli (mV), amplitude du pic de réaction.
- *SD* : Spike duration (ms), durée du pic
- *fAHP* : dV/dt (mV/ms), correspond à une variation de tension dans le neurone en fonction du temps.

Description des hyperparamètres pour la méthode des Réseaux de neurones [5] :

- *alpha* : sert à la régularisation, ce qui permet d'éviter les surajustements en pénalisant les poids de grandes amplitudes.
- *tol* : sert à l'optimisation, Lorsque la perte ou le score ne s'améliore pas d'au moins tol pour deux itérations consécutives, la convergence est considérée comme atteinte et l'entraînement s'arrête.

Description des hyperparamètres pour la méthode des SVM [6] :

- $C$ : paramètre de pénalité C du terme d'erreur.
- *degree* : représente le degré de la fonction du noyau polynomial ('poly'). Il est ignoré par tous les autres noyaux.
- *gamma* : c'est le coefficient du noyau pour 'rbf', 'poly' et 'sigmoid'.
- *Nu* : une limite supérieure sur la fraction d'erreurs d'entraînement et une limite inférieure de la fraction des vecteurs de support. Devrait être dans l'intervalle (0, 1).

# Introduction

Le but premier de ce projet est de proposer un programme permettant de classer les neurones selon leurs types, qu'ils soient de type I ou de type II. La proposition d'une interface graphique complémentaire étant suggérée.

Il s'agit donc de mettre en place un programme utilisant le Machine Learning avec les spécificités des SVM et des Réseaux de neurones [1] . Ce sont deux méthodes de classification qui ont pour but d'identifier les classes auxquelles appartiennent les éléments de l'étude, à partir d'entités communes. Ici, les éléments étudiés sont les neurones et les entités permettant de les classer ainsi que de les différencier sont des paramètres électrophysiologiques.
De plus, les SVM et les Réseaux de neurones s'appliquent au problème de la prise de décision automatisée. Il doit donc y avoir un système d'apprentissage à partir d'exemple, d'entraînement, que ce soit pour les SVM ou pour les Réseaux de neurones qui permettra de prédire au mieux les résultats attendus. Ce sont des méthodes d'apprentissage supervisé.

Pour ce faire, une analyse des données sera préalablement réalisée. En effet, cette phase du projet est nécessaire et obligatoire afin d'avoir un rendu exploitable pour la création du programme. Cette analyse passera par l'étude des différents échantillons à disposition afin de trouver les meilleurs combinaisons de paramètres possibles et ainsi proposer un programme fonctionnel et utilisable pour notamment l'équipe MNEMOSYNE appartenant au laboratoire IMN (Institut des Maladies Neurodégénératives) à Bordeaux qui est à l'origine de ce projet.

[1] Il s'agira d'utiliser des neurones formels pour analyser des neurones biologiques

## 0.1 Spécification

### 0.1.1 Etat de l'art

Le sujet étant un sujet de recherche et de programmation, il est important de s'intéresser à l'avancée des travaux concernant le domaine neurophysiologique. Il est possible d'affirmer qu'aujourd'hui, il n'existe pas de méthode systématique permettant la classification neuronale grâce à ce type de données (données électrophysiologiques).

*"Le Machine Learning est le champ d'étude qui donne aux ordinateurs la capacité d'apprendre sans être explicitement programmés"* –Arthur Samuel.

Il est intéressant de se pencher sur les méthodes de classification utilisées dans d'autres domaines que dans les neurosciences. En effet, de nombreux systèmes et entreprises utilisent des algorithmes de classification, et, l'un des leaders dans ce domaine est la librairie *scikit-learn*. Cette librairie est actuellement utilisée par bon nombre d'entreprises tel que AirBnB, Booking.com, Spotify ou encore d'autres entreprises qui les financent comme Google. Elle intervient dans de nombreuses problématiques impliquant les méthodes de Machine Learning.

Le Machine Learning est une méthode d'apprentissage permettant d'enseigner une méthode à une machine. Cette dernière sera alors capable de reconnaître certains schémas, par l'utilisation de multiples tests statistiques, dans un jeu de données. La machine sera ainsi capable de trier les données grâce à un apprentissage sur celles-ci. Elle intervient dans de nombreux domaines comme :

- la reconnaissance faciale
- la reconnaissance d'images
- la recommandation de produits
- la conduite autonome
- la reconnaissance vocale

Le principe du Machine Learning repose en général sur deux autres notions:

- l'apprentissage supervisé
- l'apprentissage non supervisé

**Machine Learning : SVM**

Les machines à vecteurs de support (Support Vector Machine (SVM) en anglais) sont des algorithmes d'apprentissage automatique qui peuvent être utilisés à des fins de classification et de régression. Les SVM sont plus couramment utilisés dans les problèmes de classification, et c'est en tant que tels, qu'ils seront exploités dans le cadre de ce projet.

Les vecteurs de support sont les points les plus proches de l'hyperplan, les points d'un ensemble de données qui, s'ils venaient à être supprimés, modifieraient la position de l'hyperplan. Pour cette raison, ils peuvent être considérés comme des éléments critiques d'un ensemble de données. (Figure 1)

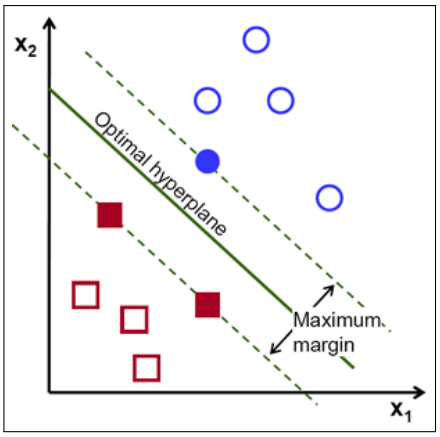


Figure 1: Représentation de l'hyperplan et de la marge. Méthode SVM linéaire.[3]

Un hyperplan peut être considéré comme une ligne qui sépare et classe un ensemble de données. Ainsi, plus les points de données sont éloignés de l'hyperplan, plus il est possible de penser que ces derniers ont été correctement classés. De ce fait, quand de nouvelles données de test sont ajoutées, une classe peut être appliquée à celles-ci. Les SVM sont donc basés sur la volonté de trouver un hyperplan qui divise avec la plus grande efficacité possible, un jeu de données en classes. La difficulté de cette méthode relève donc de la nécessité de trouver le bon hyperplan. La distance entre l'hyperplan et le point de données le plus proche de l'un ou l'autre ensemble est nommé marge. Il faut donc choisir un hyperplan avec la plus grande marge possible entre l'hyperplan et n'importe quel point des données d'apprentissage, ce qui permet d'augmenter les chances de classer correctement de nouvelles données. Cependant, les données sont rarement facilement classifiables empêchant toute séparation linéaire. Il est donc nécessaire de passer d'une vue 2D des

données à une vue 3D. C'est le kernelling.(Figure 2)

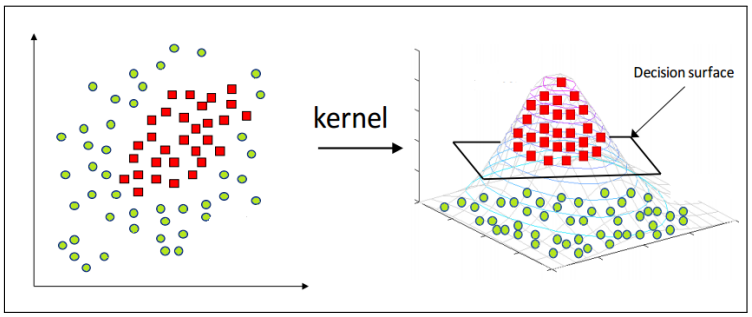


Figure 2: Kernelling d'un jeu de données.[4]

Le fait de passer en trois dimensions oblige l'hyperplan a être autre chose qu'une ligne. Il est maintenant un plan. Les données continueront donc à être représentées dans des dimensions de plus en plus élevées jusqu'à ce qu'un hyperplan puisse être formé.

Les SVM sont utilisés pour des tâches de classification de texte telles que l'attribution de catégorie, la détection de spam et l'analyse de sentiment [7] . Ils sont également couramment utilisés pour les problèmes de reconnaissance d'image, en particulier pour la reconnaissance basée sur les aspects et la classification basée sur les couleurs. Les SVM jouent également un rôle essentiel dans de nombreux domaines de la reconnaissance des chiffres manuscrits, tels que les services d'automatisation postale [1].

**Machine Learning : Réseau de neurones**

L'idée derrière les réseaux de neurones est de simuler (comme une copie simplifiée mais tout de même fidèle) des cellules cérébrales interconnectées. Cette simulation se fait à l'intérieur d'un ordinateur afin qu'il apprenne, reconnaisse des modèles et prenne des décisions comme le ferait un cerveau humain. Les simulations sur ordinateurs ne sont que des accumulations de variables algébriques et d'équations mathématiques qui permettent de relier ces simulations entre-elles (soit des nombres dont les valeurs changent sans cesse).

Un réseau de neurone classique a des milliers de neurones artificiels nommés 'unités' disposés en une série de couches. (Figure 3) Chacune de ces couches se connectant aux couches à ses côtés. Certaines de ces couches sont les unités d'entrées (input), elles reçoivent des informations que le réseau tentera de traiter. D'autres unités se trouvent du côté opposé du réseau (output) et indiquent comment elles réagissent aux informations qu'elles ont apprises.

Entre les unités d'entrées et de sorties, se trouvent une ou plusieurs couches d'unités 'cachées', formant la grande majorité de ce réseau artificiel.

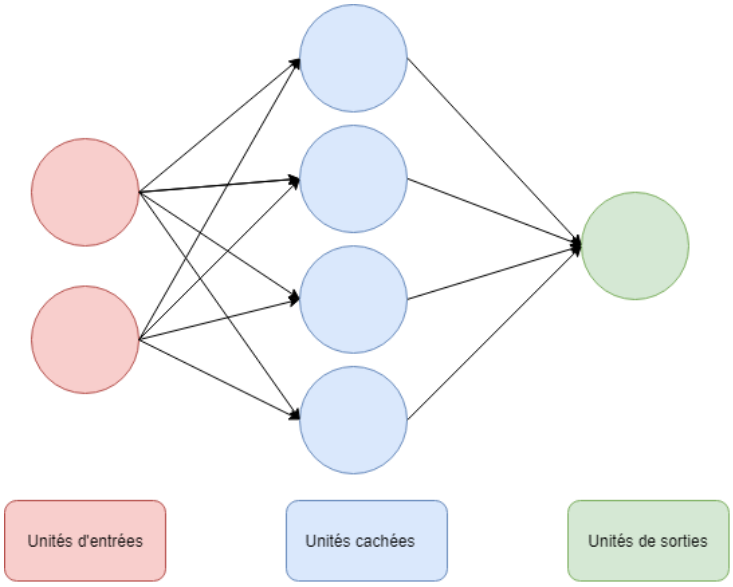


Figure 3: Schéma d'organisation du fonctionnement d'un réseau de neurones.

Leur rôle est de transformer les unités d'entrées en quelques chose que les unités de sorties peuvent utiliser. La plupart des réseaux étant entièrement interconnectés, chaque unité cachée et chaque unité de sortie est connectée à chaque unité des couches à ses côtés.

Les connections entre une unité et une autre sont représentées par un nombre nommé 'poids'. Ce poids peut-être positif si une unité en excite une autre ou au contraire négatif, si une unité en supprime ou en inhibe une autre. Ainsi, plus ce poids est élevé, plus l'influence d'une unité sur une autre est importante. Ceci correspond au même système se produisant dans le cerveau humain : une cellule du cerveau a un impact sur une autre grâce aux synapses.

Il convient à présent de se demander comment un réseau de neurones fonctionne réellement, et ainsi, de quelle façon est-il capable d'apprendre des choses.

L'information circule à travers un réseau de neurones de deux façons. Lorsqu'il apprend (donc lorsqu'il est entraîné) ou lorsqu'il fonctionne normalement (après avoir été entraîné). Les informations sont introduites dans le réseau via les unités d'entrées. Ceci déclenche les couches d'unités cachées ainsi que celles arrivant aux unités de sortie. Cependant, toutes les unités

n'envoient pas un signal en continu. En effet, chaque unité reçoit des entrées d'unités placées à sa gauche, et les entrées sont multipliées par le poids des connexions qu'elles parcourent. Chaque unité additionne toutes les entrées qu'elle reçoit de cette manière. Si la somme est supérieure à une certaine valeur seuil, l'unité envoie un signal et déclenche les unités auxquelles elle est connectée (donc celles situées à sa droite).

Pour qu'un réseau de neurones apprenne, il doit y avoir un élément de rétroaction. De la même manière qu'il est dit à un enfant si ce qu'il fait est bien ou mal. De cette façon, plus la différence en ce qui a été fait et le résultat est grande, plus la réponse va être modifiée de manière radicale. Ce processus de rétroaction est appelé rétro-propagation. Cela implique de comparer la sortie qu'un réseau produit avec la sortie qu'il était censé produire.

La différence entre les deux peut ensuite être utilisée afin de modifier les poids des connexions entre les unités du réseau, des unités de sorties aux unités d'entrées tout en passant par les unités cachées et ceci en 'marche arrière'. Ainsi, la rétro-propagation va provoquer l'apprentissage du réseau, réduisant la différence entre la sortie réelle et la sortie prévue au point où les deux coïncident. Par conséquent, le réseau trouvera la 'bonne' réponse comme il le devrait.

Une fois que le réseau été instruit avec suffisamment d'exemples d'apprentissage, il arrive à un point où il est possible de lui présenter un tout nouvel ensemble d'entrées qu'il n'a jamais rencontré auparavant.[2]

### 0.1.2 Objectif et Analyse des besoins fonctionnels

Ce projet s'appuie sur des données métriques électrophysiologiques (au nombre de 8) recueillies de la même manière par plusieurs équipes de recherche. Ces données concernent deux populations de neurones distinctes mais portant sur les mêmes paramètres. L'objectif est de créer un programme qui saura classer automatiquement les neurones, avec le meilleur taux de prédiction possible, à partir des données disponibles. Bien que ce soit un sujet de recherche, s'il advenait que ce soit possible de classer des neurones selon les valeurs de leurs paramètres électrophysiologiques, il ne serait alors plus nécessaire de réaliser une classification *post-mortem*. De plus, cela limiterait le recours à des animaux transgéniques.

Pour ce projet, le client demande un logiciel, permettant d'ouvrir aisément des fichiers de données ainsi que de visualiser les résultats de l'analyse sous-jacente. De ce fait, le but de ce projet est de créer un logiciel qui permettra de distinguer les différents types de neurones grâce à des enregistrements effectués *in vitro* avec le plus d'efficacité possible.

Afin de créer un programme clair et facile d'utilisation, il sera nécessaire de créer une interface graphique dans le but d'éviter à l'utilisateur d'avoir recours au terminal. Aussi, il est important de définir le type de fichier d'entrée à étudier et de sortie en réponse. De plus, sachant qu'il y a plusieurs méthodes possibles pour l'analyse de leur fichier, il sera nécessaire de proposer à l'utilisateur un choix des paramètres souhaités ainsi que le choix du fichier pour l'entraînement. En outre, si ce dernier ne désire pas exprimer de choix, les paramètres seront mis par défaut et un fichier d'entraînement lui sera proposé afin de faciliter l'utilisation du programme. Cette configuration permettra d'avoir un programme qui sera utilisable tant pour un utilisateur ne connaissant ni les SVM ni les Réseaux de neurones, tant pour un utilisateur qui connaît le fonctionnement, et qui souhaite avoir son mot à dire sur le choix des paramètres.

En plus du fichier de sortie, un visuel présent sur l'interface graphique des résultats sera présent. Celui-ci sera composé d'un graphique en secteur avec le pourcentage de neurones de type I et de type II, ainsi qu'un graphique 3D de la répartition des types I et II puis enfin, d'un tableau modulable des résultats. Ce tableau, pourra donc être corrigé par l'utilisateur et être utilisé pour l'entraînement de la méthode. Il sera aussi primordial de créer un programme de façon à ce qu'il puisse être exécutable sur plusieurs systèmes d'exploitation telles que *MacOs*, *Windows* et *Linux*. Ceci a pour but d'éviter tout conflit pour l'utilisateur et de lui faciliter la prise en main du logiciel. Ainsi, le choix des langages, des librairies et des logiciels devra être fait en fonction de ce point.

Dans le but de pouvoir créer tout cela, il sera nécessaire d'étudier et de comparer les meilleurs méthodes et classes que ce soit pour les SVM que pour les Réseaux de neurones ainsi que les meilleurs échantillonnages. Cela permettra donc de proposer à l'utilisateur les meilleurs combinaisons possibles. En outre, il sera indispensable alors de créer des programmes, scripts, pour cette étude.

### 0.1.3 Choix de Programmation et Analyse des besoins non fonctionnels

Il existe de nombreuses méthodes de classification supervisée. Ici, seuls les SVM et les Réseaux de neurones seront utilisés.
Il est donc important d'implémenter des scripts faisant fonctionner les SVM et les Réseaux de neurones avant de pouvoir sortir un programme fonctionnel avec interface graphique.

Pour cela et afin de répondre aux attentes du programme demandé, tous les algorithmes utiliseront le langage Python. En effet ce langage est multi-plateformes, puissant, de haut niveau, structuré et open source. Par ailleurs, ce langage est adapté à la manipulation de données quantitatives, telles que les données fournies. De plus, il a l'avantage d'avoir des bibliothèques qui sont intéressantes pour le programme. Ces librairies sont principalement Scikit-Learn et MatPlotLib. La librairie Scikit-Learn est dédiée à l'apprentissage automatique et est conçue pour s'harmoniser avec des autres bibliothèques libre comme Python. Elle comprend des algorithmes de classification.
La librairie Matplotlib quant à elle sert à tracer et visualiser des données sous forme de graphiques, elle peut être aussi combinée avec Python. De plus, elle possède une forte communauté très active. Cela permet d'utiliser une librairie bien "rodée" avec une forte documentation en ligne. Elle est aussi une bibliothèque de haut-niveau. Ceci permet de faire des calculs interactifs et des représentations 3D.
Les avantages de ces deux librairies sont donc la raison de leur sélection.

Pour l'interface graphique, Tkinter [8] est le choix le plus logique. Effectivement, cette librairie graphique libre est créée pour le langage Python. WxWidget aurait pu être utilisé mais par soucis de portabilité et voulant faire fonctionner le programme sur *MacOs* sans rien devoir configurer sur la machine, le choix s'est donc porté sur Tkinter. De plus, Kivy, qui était l'idée de départ, aurait pu être une solution, mais cette dernière ayant des conflits avec la dernière version d'*Ubuntu*, il a été donc préférable d'utiliser Tkinter. D'autre part, Tkinter est inclus dans la bibliothèque Python standard ce qui induit moins de problème de compatibilité.

## 0.2 Conception

Au départ l'élément d'intérêt est un neurone qui est représenté par un objet structuré dans un fichier de type CSV. Chaque ligne correspond à un neurone, qui est lui même caractérisé par 8 colonnes. Ces dernières représentent chacune un paramètre électrophysiologique récolté sur des tissus vivants. Le fichier se présente donc sous la forme de la table 1. Les unités sont des variables quantitatives mais ayant des unités différentes.

Table 1: Aperçu des données (sélectionnées aléatoirement)

| nClass | IR | RMP | RH | ST | DTFS | SA | SD | fAHP |
|---|---|---|---|---|---|---|---|---|
| 1 | 180.3 | -76.32 | 95 | -36.75 | 357 | 68.51 | 3 | -0.73 |
| 2 | 67.89 | -72.25 | 205 | -40.67 | 432.7 | 82.92 | 2.3 | -1.07 |

Finalement, il faudra récupérer un fichier avec un format et séparateur pour ainsi définir ce type de fichier et le type de séparateur. Laisser le choix du séparateur à l'utilisateur peut lui permettre de ne pas avoir à retoucher son fichier de données.

### 0.2.1 Architecture pour méthodes supervisées de Machine Learning

#### SVM

Pour l'analyse des paramètres optimaux et hyperparamètres, divers scripts devront être réalisés. Tout d'abord il faudra déterminer les classes de SVM dans Scikit-learn (*sklearn.svm*) qui seront utilisées parmi :

- LinearSVC (*sklearn.svm.LinearSVC*)
- NuSVC (*sklearn.svm.NuSVC*)
- SVC ((*sklearn.svm.SVC*)

Ces différentes classes de SVM ont été mises en place car elles utilisent des librairies spécifiques. *Liblinear* pour *LinearSVC* et *Libsvm* pour les deux autres. Ce qui différencie ces deux dernières ce sont des hyperparamètres.

Effectivement, la classe LinearSVC, ne possède pas de noyau/méthode, celle-ci est focalisée sur la méthode linear, en effet elle est la méthode 'linear' la plus efficace et fournissant les meilleurs résultats. Pour les deux autres classes, il faudra étudier les cinq noyaux ou méthodes différent(e)s. En effet il existe cinq noyaux ou méthodes en fonction des classes, ceux-ci se basent sur différentes façon de classer les données. Ces méthodes de classification sont elles-mêmes basées sur des principes mathématiques et statistiques, ainsi il sera possible d'utiliser les méthodes suivantes (cf.Annexe, figure 14):

- 'linear'[1]
- 'poly'[2]
- 'rbf' [3]
- 'sigmoid'[4]
- 'precomputed'[5]

La méthode '*precomputed*' ne sera pas intéressante, en effet c'est une méthode qui requièrent une mise au point de sa propre méthode de classification mathématique et statistique. De plus, comme indiqué, plusieurs hyperparamètres influent sur les résultats. Il existe donc différentes combinaisons possibles à étudier.
En effet, *NuSVC* est une classe qui dispose d'un hyperparamètre (Nu) capable de contrôler le nombre de vecteurs supports (support vectors) alors que la classe *SVC* se base sur des comparaisons "un à un" avec l'hyperparamètre changeant(C). De plus, l'hyperparamètre 'gamma' est commun aux méthodes rbf et sigmoid puis l'hyperparamètre 'degree' est utilisé pour la méthode 'poly'.
Des scripts seront réalisés afin d'observer la meilleure optimisation des paramètres. Des boucles imbriquées permettront de comparer les différentes combinaisons de paramètres et hyperparamètres entre eux ainsi que leurs variations propres.

Il existe également un paramètre coef0 commun à la méthode '*sigmoid*' et '*poly*' mais celui ci est un paramètre non essentiel fixé par défaut.

[1]Cette méthode rend un modèle de type linéaire : $k(x, y) = x^T y$

[2]Cette méthode rend un modèle de type polynomial : $k(x, y) = (x^T y + c)^d$

[3]Cette méthode utilise des courbes normales autour des points de données, c'est l'une des meilleurs pour établir une analyse multidimensionnelle : $\mathrm{x(x,y)} = \exp(-\frac{|x-y|}{a})$

[4]Cette méthode renvoie un modèle de type sigmoïde, en forme de S : $k(x, y) = tanh((ax^T)y + c)$

[5]Cette méthode n'est pas définissable à l'avance car il est nécessaire de créer son propre modèle mathématique de classification

Idéalement, le programme pouvant faire tourner tous les différents types de SVM devra se présenter de la manière suivante (figure 4):

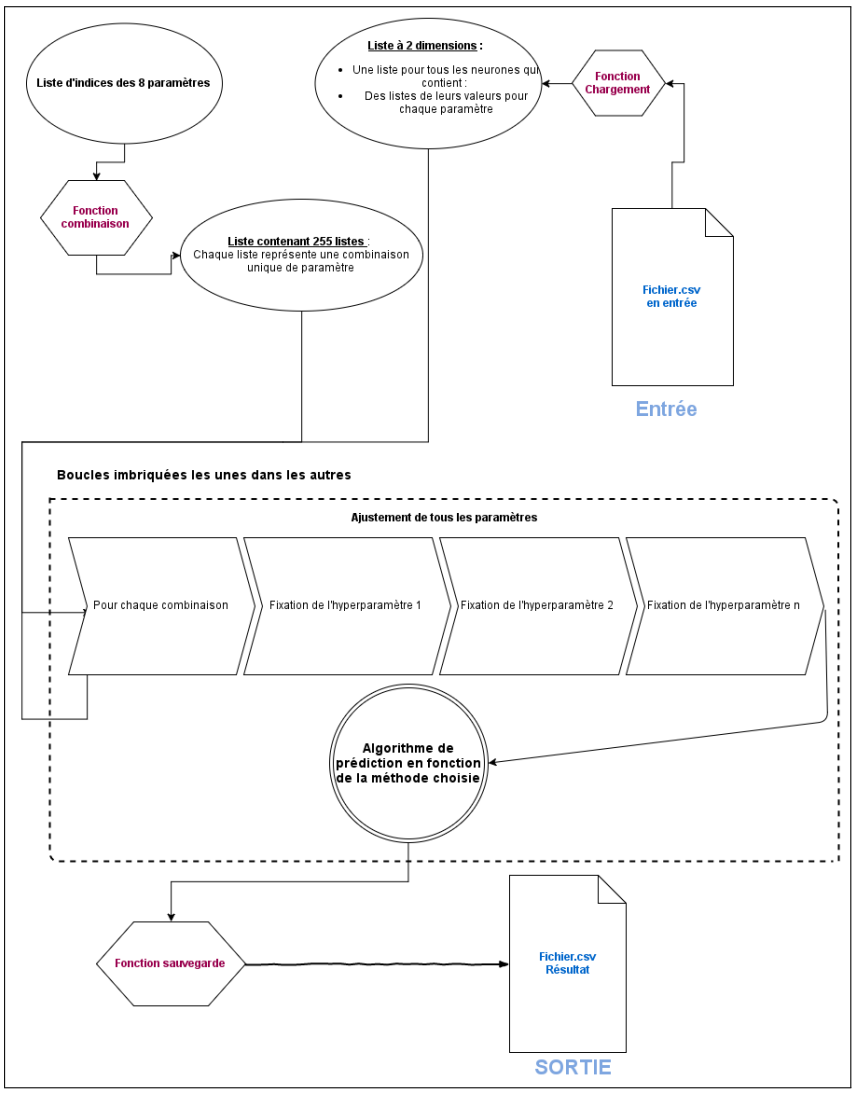


Figure 4: Schéma de conception des programmes pour établir les meilleurs paramètres afin de classer les neurones

Il est donc prévu d'avoir un fichier en entrée. Il est indispensable de mettre en place une *fonction* permettant le chargement et l'utilisation du fichier. Ainsi, en découlera une *liste à deux dimensions*. Pour chaque neurone une liste devra être créée contenant le type du neurone ainsi que ses données électrophysiologiques. En outre, grâce à une *fonction combinaison*, il sera possible de récolter une liste contenant toutes les combinaisons de paramètres

possibles, par exemple, IR avec RMP, puis IR avec RMP et RH etc...

A partir de cela, l'utilisation de *boucles* imbriquées est nécessaire afin d'avoir un ajustement de tous les paramètres. Pour chaque combinaison trouvée par la *fonction combinaison*, il devra y avoir fixation du premier hyperparamètre, puis du second, jusqu'à l'hyperparamètre n (en fonction des différentes méthodes). Ainsi, avec ce système tous les assemblages possibles avec les différentes combinaisons, les différents hyperparamètres seront étudiés. De ce fait, il sera plus aisé de choisir et de trouver les meilleurs combinaisons entre toutes celles possibles.

De plus, il sera nécessaire, par souci de clarté et d'organisation, de faire varier les différentes méthodes et classes énumérées en appliquant un script différent pour chacune d'elles. Effectivement, cela permettra de ne pas avoir des fichiers de sortie trop imposants ou un script trop long à exécuter.

### Réseau de neurones

Pour l'analyse des paramètres optimaux et hyperparamètres des Réseaux de neurones, la librairie *sklearn.neural_network* devra être utilisée. Elle provient elle-même de Scikit-Learn. Dans cette librairie, les classes suivantes sont disponibles :

- 'Regressor'
- 'Classifier'

Il existe quatre noyaux (ou méthodes) différents pour ces classes qui sont les suivants :

- 'identity' [1]
- 'logistic' [2]
- 'tanh' [3]
- 'relu' [4]

[1] L'activation du neurone est transmise directement en sortie, retourne $f(x) = x$

[2] Fonction sigmoïde logistique : retourne $f(x) = 1/(1 + exp(-x))$ : courbe en "S"

[3] Courbe sigmoïdale similaire à la fonction logistique mais qui produit généralement de meilleurs résultats en raison de sa symétrie : retourne $tan(x) = (2/(1 + exp(-2x))) - 1$

[4] Fonction linéaire rectifiée : retourne $f(x) = 0$ for $x < 0$ et x for $x >= 0$

Ces derniers peuvent aussi se nommer fonction d'activation qui représente le potentiel d'activation du neurone. "Les neurones d'un réseau possèdent des fonctions d'activation qui vont transformer les signaux émis par les neurones de la couche précédente à l'aide d'une fonction mathématique."[10]

Par défaut c'est la méthode '*relu*' qui est initialisée dans la librairie. Ces fonctions d'activations diffèrent selon leur manière d'effectuer leurs calculs. Par exemple, '*tanh*' est une fonction hyperbolique alors que '*logistic*' est une fonction sigmoïde logistique.

Idéalement, le programme pouvant faire tourner tous les différents types de Réseaux de neurones devra se présenter de la même manière que celui pour les SVM.

Effectivement, seulement l'algorithme de prédiction changera car ce dernier prendra en compte des classes, des méthodes ainsi que des hyperparamètres différents de ceux des SVM comme vu au-dessus.

En effet, 3 *solver* permettent d'étudier dans la classe 'Classifier' :

- lbfgs
- sgd
- adam

Le choix se portera sur le *solver 'lbfgs'* car ce dernier correspond à notre taille d'échantillons. En effet, dans la librairie de Scikit learn, il est décrit que le *solver 'lbfgs'* est plus performant et rapide pour des petites tailles d'échantillons alors que le *solver 'adam'* sera plus efficace sur des gros échantillons correspondant à des milliers de données d'entraînements voir plus. Or nous notre taille d'échantillonnage est de l'ordre de la centaine (117 données).

De même que pour les SVM, un script différent sera crée pour chaque noyau et chaque classe. Ainsi, chaque script correspondra à un algorithme de prédiction différent.

Finalement, le temps passé pour ces analyses sera très chronophage sur le temps imparti total au vu de l'exploration des paramètres optimaux prévue.

### 0.2.2 Architecture du programme et son interface graphique

Afin que le logiciel soit le plus simple d'utilisation possible, il est important de mettre en place une interface graphique. Ainsi, l'utilisateur pourra utiliser le logiciel sans avoir à recourir à l'utilisation d'un terminal de commandes. Ce dernier pouvant s'avérer repoussant pour une personne n'étant pas familière de son utilisation.

L'interface graphique permet à l'aide de fonctionnalités et de boutons de simplifier l'utilisation du logiciel. Il est important qu'une interface soit ergonomique et facile d'utilisation afin d'être optimale pour tout types d'utilisateurs. Le choix de la librairie pour la création de cette interface graphique semblera donc se porter sur Tkinter.
A ce stade, l'interface graphique envisagée propose plusieurs options :

- Un bouton "Entraînement" permettant de charger un fichier d'entraînement nécessaire à l'analyse et au bon fonctionnement de l'apprentissage des SVM.
- Un bouton "Test" permettant de charger un fichier test sur lequel l'analyse sera réalisée.
- Un bouton "Paramètres" permettant à l'utilisateur d'ajuster les paramètres de la méthode.
- Un bouton "Analyse" permettant d'exécuter l'analyse sur le fichier test de l'utilisateur.
- Un bouton "Quitter" afin de quitter l'interface graphique.

Chacun de ces boutons devra ouvrir une nouvelle fenêtre contenant les indications à suivre, considérées comme prérequis afin d'exécuter l'analyse. Chaque fichier chargé devra être au format csv ou txt. Un visuel pour illustrer cela est disponible avec la figure 5.

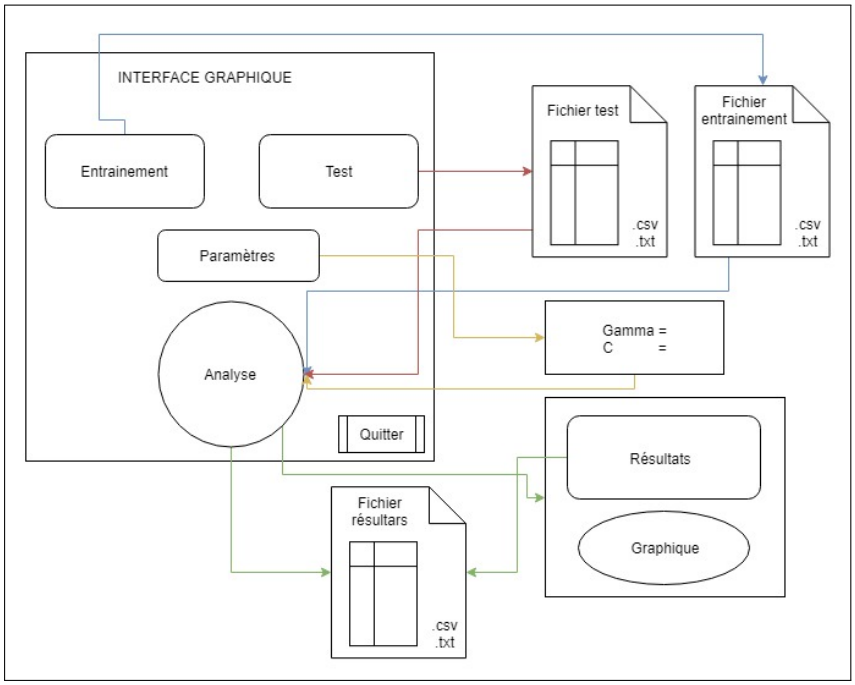


Figure 5: Schéma de conception du programme et son interface graphique

Il est envisagé que les résultats apparaissent dans un nouveau fichier CSV ou texte en sortie ainsi que sur l'interface graphique où un tableau avec les résultats finaux sera représenté. Il est également envisagé d'afficher un graphique permettant de visualiser rapidement les résultats obtenus grâce à l'analyse.

## 0.3 Réalisation

Finalement, il a été choisi concernant le chargement des données l'utilisation de fichiers au format CSV ou texte qui sont au final, les mêmes formats. Ceci a été choisi afin que le client ne s'y perde pas. En outre le choix du séparateur peut varier à la convenance de l'utilisateur. Comme prévu, une importante partie du projet a été consacrée à l'implémentation et à la mise au point des paramètres optimaux et à la méthode de recherche pour ces derniers. Cette partie de développement a empiété sur le développement du produit final. Cependant, celui-ci ne pouvait exister qu'après la phase d'exploration des paramètres optimaux.

### 0.3.1 Identification des (hyper)paramètres optimaux

#### Agencement des scripts

Dans le but de trouver la combinaison de paramètres ainsi que des hyperparamètres optimaux, il convient de tester chaque combinaison en fonction de chaque valeur possible des hyperparamètres. Pour ce faire, un script a été mis en place. Celui-ci débute par une fonction qui permet de charger le fichier dans le script. Ce fichier est converti en un tableau qui est retourné afin de pouvoir être réutilisé. Il convient cependant de prendre en compte que la première colonne du fichier csv contient la classe des neurones. Cette fonction prend donc en argument le nom du fichier qui est donné par l'utilisateur du script. Puis, elle retourne une liste à deux dimensions indispensable pour la conversion en *numpy.array*. En effet, cette forme *numpy.array* est primordiale pour l'utilisation de la librairie scikit learn.

Une liste comportant les indices pour chaque paramètre électrophysiologique (*listecombin=[1,2,3,4,5,6,7,8]*) est créée. Cette dernière est prise en argument dans une fonction *combinaisons* permettant de retourner toutes les combinaisons possibles à partir des huit paramètres, soit 255 combinaisons possibles (équivaut à 2 puissance 8 -1).

A ce stade, toutes les combinaisons de paramètres sont prêtes à être testées. Il reste alors à faire varier les hyperparamètres afin de recueillir le plus haut pourcentage de classification réussie.

Pour faire varier les hyperparamètres, des boucles imbriquées ont été mises en place. Celles-ci permettent de faire varier un hyperparamètre tout en fixant les autres, et ce, pour chaque hyperparamètre et pour chaque combinaison de paramètres électrophysiologiques.

Effectivement, il est possible de constater sur la figure 6, qu'*Alpha* est fixé alors que *Tol* varie et ce, pour une même combinaison de paramètres.
C'est ce qui permet l'exploration complète de l'ensemble des paramètres étudiés.

Une fois que toutes les boucles imbriquées sont en place, il faut s'occuper des différentes méthodes de classification. Pour ce faire, plusieurs scripts ont été mis en place, chacun comprenant une classe différente pour la méthode choisie comme il était prévu de le faire, pour un soucis d'organisation.

| Résultat en % | Alpha | Tol | Combinaison de paramètres | | |
|---|---|---|---|---|---|
| 58.620 | 1E-05 | 1 | IR | RH | SA |
| 44.827 | 1E-05 | 0.1 | IR | RH | SA |
| 48.275 | 1E-05 | 0.01 | IR | RH | SA |
| 41.379 | 1E-05 | 0.001 | IR | RH | SA |
| 58.620 | 1E-05 | 0.0001 | IR | RH | SA |
| 41.379 | 1E-05 | 1E-05 | IR | RH | SA |
| 62.068 | 1E-05 | 1E-06 | IR | RH | SA |
| 41.379 | 1E-05 | 1E-07 | IR | RH | SA |
| 44.827 | 1E-05 | 1E-08 | IR | RH | SA |
| 41.379 | 1E-05 | 1E-09 | IR | RH | SA |
| 44.827 | 0.0001 | 1 | IR | RH | SA |
| 41.379 | 0.0001 | 0.1 | IR | RH | SA |
| 41.379 | 0.0001 | 0.01 | IR | RH | SA |

Figure 6: Exemple de résultats des boucles imbriquées pour les hyper-paramètres pour une même combinaison de paramètres pour la méthode réseau de neurones.

**Module SVM** Pour la méthode SVM, les noyaux LinearSVC, NuSVC et SVC définiront des classes qui seront implémentées dans différents scripts.

```
svm.NuSVC(kernel='linear',nu = t)
```

A partir de cette ligne de code, il est possible de la modifier afin d'adapter chaque script à chaque méthode.

```
svm.SVC(kernel='rbf', gamma=h, C=t)
```

En effet, pour modifier la méthode, il suffit d'ajouter SVC, NuSVC ou LinearSVC avant la parenthèse et après le svm. Pour modifier le noyau (ou kernel), il faut indiquer le nom de celui-ci entre guillemets. Comme il est possible de le voir avec 'linear' et 'rbf'. Enfin, cette ligne de code intègre également un nombre variable d'hyperparamètres influençant le noyau. Cette ligne est disponible grâce à la librairie scikit-learn.

**Réseaux de neurones** Pour la méthode réseau de neurones, la classe 'Regressor' ne sera pas utilisée car après plusieurs essais, il s'avère qu'elle renvoie des floats au lieu d'integer (1 ou 2) pour les résultats de classes des types de neurones. Ce qui induit donc un résultat de 0% de réussite pour cette classe. En effet cela est dû au fait que cette classe ne permet pas de classifier des données mais seulement d'y appliquer un régression.

La classe choisie est donc 'MLPClassifier'. Elle prendra en argument le solver 'lbfgs' qui n'est jamais modifié pour les raisons énumérées dans la partie

conception. L'argument activation est ensuite modifié pour chaque script de manière à ce qu'ils prennent la valeur 'tanh', 'relu', 'identity', 'logistic'. La méthode prend également en arguments deux hyperparamètres alpha et tol ainsi que hidden_layer_sizes.
L'hidden layer sizes qui correspond à la taille de la couche cachée. Le travail de la couche cachée est de transformer les données d'entrées en quelque chose que la couche de sortie peut utiliser. Il est généralement calculé de la façon suivante : le nombre de neurones dans la couche est la moyenne des neurones entre la couche d'entrée et la couche de sortie.

```
1     MLPClassifier(solver='lbfgs', activation='tanh',
2 alpha=t, hidden layer sizes=(4,), tol=h)
```

Il est possible d'utiliser cette ligne de code grâce à la librairie scikit-learn. Cette ligne est finalement le cœur du script, c'est elle qui a pour rôle de faire varier les méthodes et les classes.

### Résultats des analyses

### SVM

Il a été trouvé que C (qui est un hyperparamètre) pour des valeurs très petites entre 10e-5 et 10e-12 il y avait toujours le même pourcentage. Ce qui nous fait la limite basse du paramètre. En revanche pour la limite haute c'est la charge de calcul qui devient limitante, en effet plus la valeur de C augmente plus le temps de calcul est long jusqu'à, pour les valeurs les plus hautes, atteindre un freeze du programme (aux alentours de 10e+3 10e+8).

Ici, il est supposé que la classe qui convient le mieux (visible après analyse des premiers scripts) est la classe SVC. Effectivement, c'est une classe complète qui fournie les résultats les plus satisfaisants (une classification des neurones correcte), le plus haut score (cf figure 7). Cependant au niveau de la moyenne et du minimum, il est observé que celles-ci ne font pas parties des plus élevées. Ceci peut certainement s'expliquer en raison du grand nombre de données générées comme il est possible de l'observer sur la figure 8.

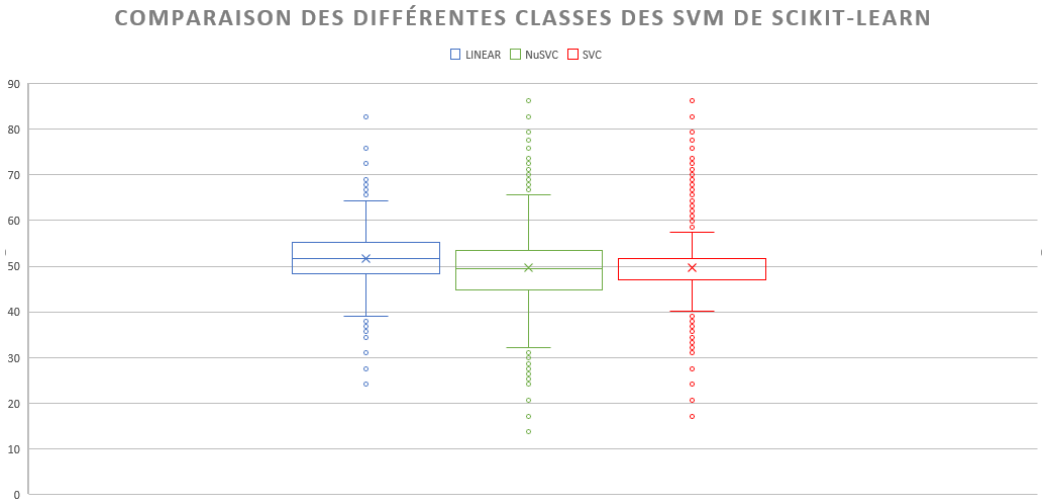


Figure 7: Boxplot comparant les différentes classes de la méthode des SVM

| | Taille du jeu de données | MAX | MIN | MOYENNE | Classement de la vitesse |
|---|---|---|---|---|---|
| LINEAR | 9937 | 82,75 | 24,13 | 51,70 | 1 |
| NuSVC | 166995 | 86,20 | 13,79 | 49,71 | 3 |
| SVC | 337609 | 86,20 | 17,24 | 49,71 | 2 |

Figure 8: Tableau récapitulatif pour chaque méthode de SVM

On se penchera donc sur cette classe elle même divisée en 5 méthodes.

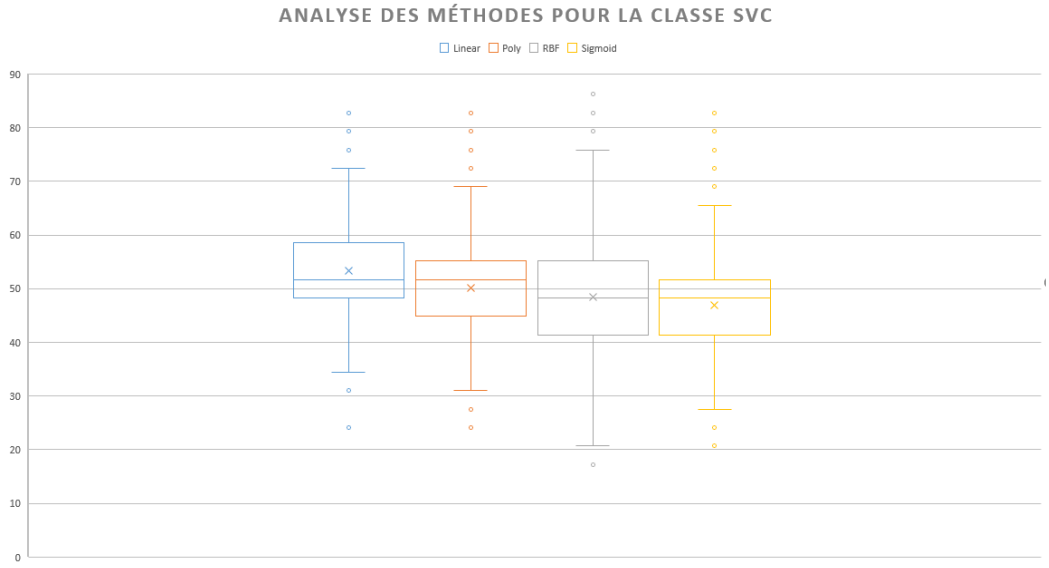


Figure 9: Boxplot comparant les méthodes de classe SVC (Linear, Poly, RBF et Sigmoid)

Ici, grâce à la figure 9, il est possible de remarquer que le plus grand nombre de haut score est fourni par la méthode RBF parmi les méthodes analysées, c'est donc elle qui a été retenue pour les paramètres par défaut du programme.
Finalement, une dernière analyse doit être effectuée, celle de l'échantillonnage. Comme il est possible de le voir sur la figure 10, il y a un net effet de l'échantillonnage, en effet, même si cela n'est pas significatif on remarque qu'entraîner avec 75% des données et tester avec 25% permet d'obtenir les meilleurs résultats.

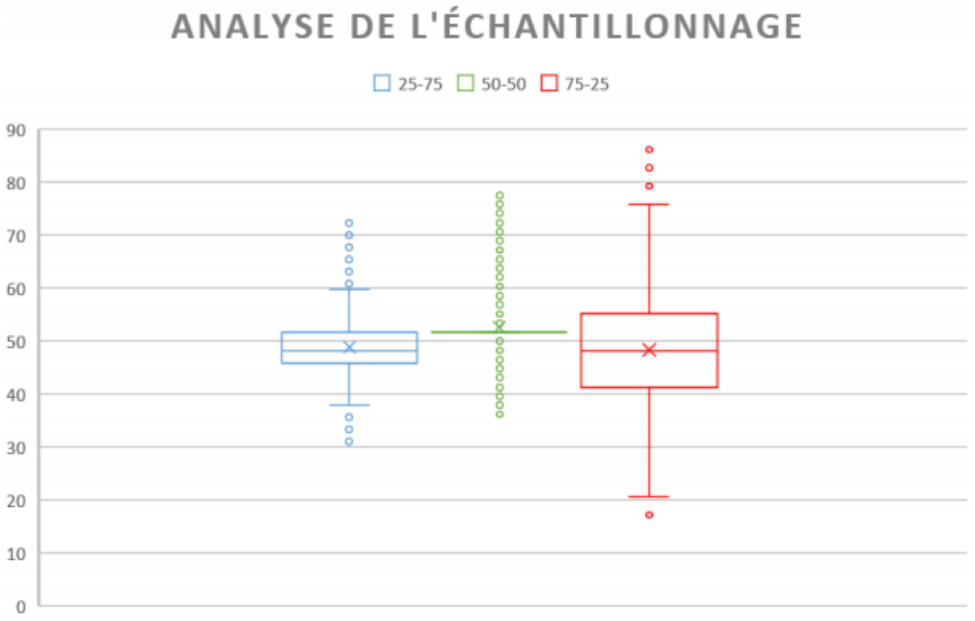


Figure 10: Boxplot de comparaison des différents échantillonnages possibles

Finalement, la méthode des SVM se verra attribuer dans le programme proposé des la classe SVC, la méthode RBF et un échantillonnage de 75% d'entraînement et 25% de test, par défaut.

### Réseau de neurones

Ici, pour les réseaux de neurones, on va préférer la méthode Identity. En effet, elle est celle qui est la plus lente mais elle fournit le plus fort taux de réussite parmi les nombreuses méthodes comme il est possible de voir sur la figure 11.

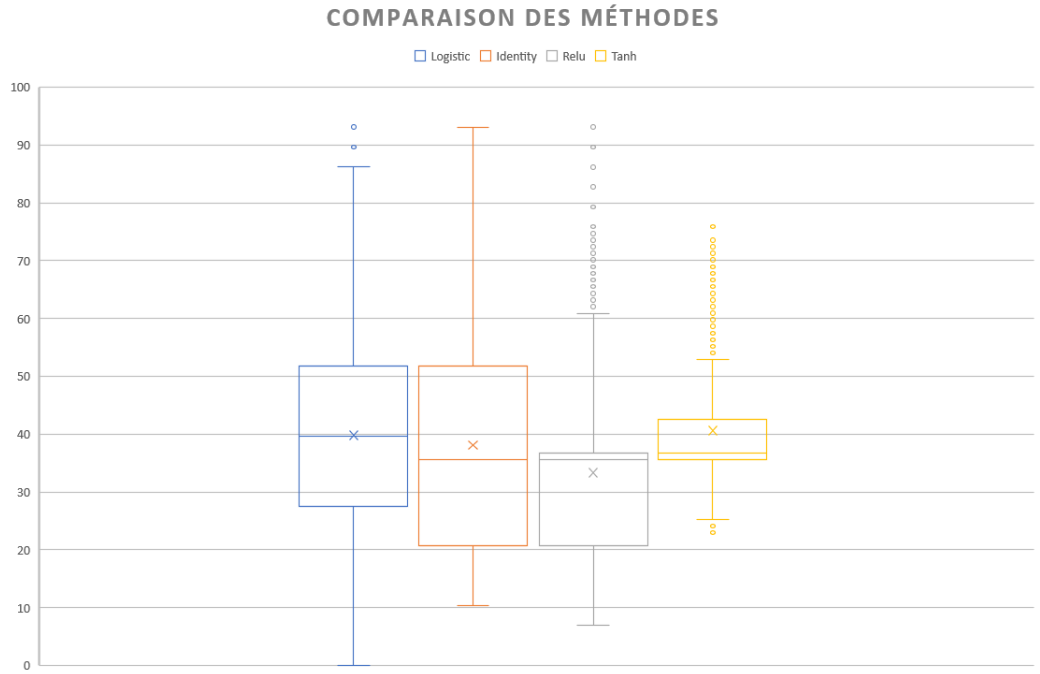


Figure 11: Boxplot de comparaison des différentes méthodes (Logistic, Identity, Relu, Tanh) pour la classe 'Classifier' de la méthode Réseau de neurones

Avec le même raisonnement que celui établi pour la méthode des SVM, il est intéressant de comparer les différents échantillonnages possibles avec la méthode 'identity'. En résultat de cela, la figure 12 est proposée.

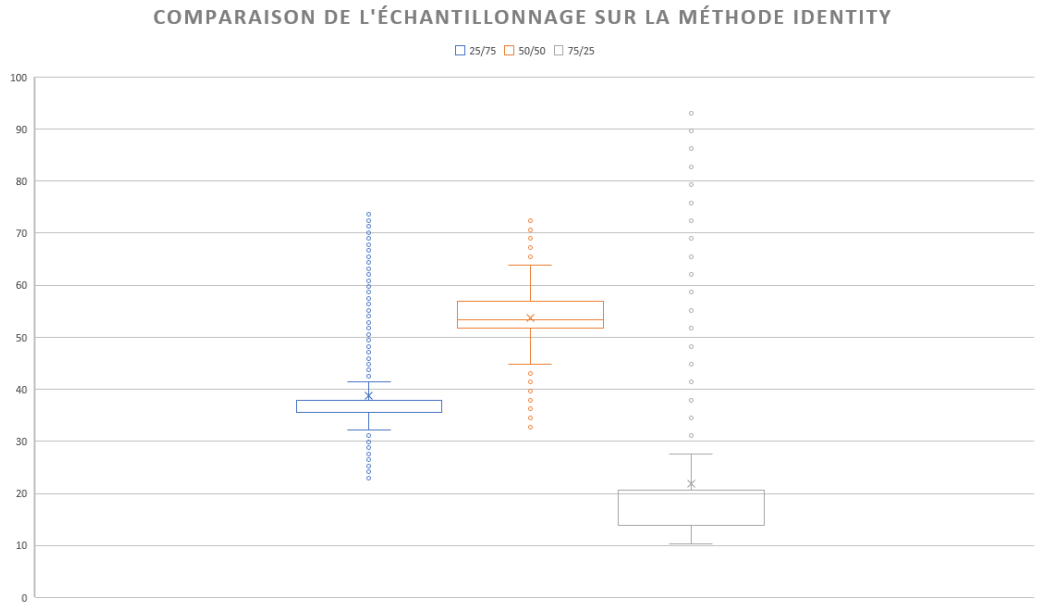


Figure 12: Boxplot comparant trois échantillonnages différents pour la méthode 'identity'

Sur cette figure, nous constatons qu'il y a une structure particulière. En effet, les résultats les plus hauts ont pour échantillonnage 75% entraînement-25% test. Or, la plupart des résultats pour cet échantillonnage se retrouvent être des taux de réussite relativement faible. Cependant, le fait qu'il fournisse de très hauts taux de réussite oriente le choix vers cette configuration.
Finalement, pour le programme, la méthode des réseaux de neurones se verra attribuée par défaut la classe 'MLPClassifier', la méthode 'identity' et un échantillonnage de 75% d'entraînement et 25% de test.

**Conclusion des paramètres optimaux** Une analyse plus poussée des meilleurs résultats obtenus a été réalisée en réitérant les entraînement de 75% des données et en testant les modèles sur les 25% restant. Il s'est alors révélé que les méthodes retenues n'étaient pas forcément les plus performantes. La méthode retenue, pour SVM et Réseau de neurones confondus, malgré ces résultats moyens correspond à la classe SVC, la méthode rbf avec un C=10e-7 et un gamma=10e7 ainsi que la combinaison de paramètres RH-ST-SA-SD-fAHP.
Il a également été remarqué un effet notoire d'un échantillon à l'autre, ainsi l'écart de valeurs entre les différents échantillons empêcherait un apprentissage efficace. Une piste à explorer serait de veiller aux conditions d'expérimentation qui semblent avoir un impact important sur le modèle.

### 0.3.2 Module du programme principal et interface graphique

Pour la mise en place de l'interface graphique, plusieurs boutons ont été créés. En effet, la fonction *Button* implémentée dans TKinter permet de modifier à souhait, l'emplacement, la taille, la police, la couleur et d'y ajouter du texte. Ces derniers permettent également d'appeler les *fonctions* souhaitées.

Le premier bouton, "Ajustement des paramètres", permet d'ouvrir en chaîne des fenêtres *TopLevel*, chacune reliée à différentes *fonctions*. La première offre à l'utilisateur le choix de choisir la classe désirée. La seconde lui permet le choix de la méthode de la classe choisie au *TopLevel* précédent. La troisième lui autorise à varier les valeurs des hyperparamètres. Enfin, la dernière lui confère la possibilité de sa combinaison de paramètres électrophysiologiques. Cependant, il peut aussi laisser les paramètres par défaut en cliquant sur le bouton par défaut lors de l'ouverture de la première fenêtre.

Le bouton suivant, "Chargement du fichier d'entraînement", propose à l'utili-sateur de charger son propre fichier, directement à partir de son répertoire. Dans le cas échéant, un fichier sera chargé par défaut. Cependant, l'utilisateur doit indiquer le séparateur de son fichier .csv ou .txt.

De ces deux derniers découle le troisième bouton : "Lancer la simulation". Effectivement, avant de cliquer sur ce bouton, il est nécessaire d'avoir choisi les paramètres ainsi que d'avoir chargé un fichier d'entraînement auquel cas, les paramètres par défaut seront chargés. Une fois ces critères réunis, ce bouton permet de choisir la méthode d'échantillonnage puis de lancer une simulation de classification selon l'échantillonnage et les paramètres choisis. Le résultat de cette simulation sera affiché, dans une fenêtre *TopLevel*, sous forme de pourcentage.

A présent, l'utilisateur peut charger son fichier .csv ou .txt à tester. Le logiciel demande à l'utilisateur le séparateur de son fichier. De plus, le logiciel est fait de manière à ce que l'usager doive choisir le nom du fichier de sortie à travers le bouton du même nom. Effectivement, cela permet à l'utilisateur de retrouver ses résultats plus facilement.
Une fois que tous les fichiers sont chargés et que les paramètres et hyperparamètres sont sélectionnés, il est alors possible de lancer l'analyse en cliquant sur le bouton à cet effet "Lancer la classification". Celui-ci permet d'afficher une fenêtre de résultats sous forme d'un tableau avec les différents neurones classés selon qu'ils soient de type I ou de type II.

A partir de cette fenêtre, il est également possible d'afficher un diagramme représentant le pourcentage de neurone de chaque type. Ce diagramme peut être enregistré en cliquant simplement sur l'icône de la disquette se situant en bas du graphique. L'utilisateur peut également avoir une représentation de trois de ses paramètres sur un graphique 3D grâce à *MatPlotLib*. Ce dernier propose un angle de vue pouvant être modifié à la guise de l'utilisateur.
Ce graphique peut également être enregistré.

De plus, il a été prévu de prendre en compte que l'usager puisse connaître les résultats du fichier à tester ou les vérifier à posteriori. Ainsi, le programme propose de corriger les résultats erronés et de ce fait de se servir des données corrigées afin d'entraîner les modèles qui serviront pour les analyses ultérieures. En effet, les données s'ajoutent au fichier d'entrée pour l'entraînement afin que les analyses à venir soient plus fiables.
Par ailleurs, si l'utilisateur se rend compte qu'il y a des erreurs dans les résultats modifiés ou non et qu'il a choisi d'entraîner les modèles, un fichier Backup est prévu afin de corriger le tir.
Tous les boutons ainsi que les actions qu'ils engendrent sont résumés et illustrés sur la figure 13.
Une fois que toutes les fonctions ont été implémentées, il est possible par souci de design de modifier les couleurs et la taille des fenêtres et de pouvoir les centrer. Il est également permis d'utiliser la fonction *Canvas* grâce à TKinter qui permet d'ajouter des images en fond.

De plus, une gestion d'erreur a été élaborée. Elle se traduit, par exemple, par la mise en place de quelques lignes de code permettant de distinguer le fichier d'entraînement du fichier test. En effet, le fichier entraînement contient une colonne de plus contenant les types de neurones pour chaque ligne. De fait, si l'usager choisit un fichier test à la place d'un fichier d'entraînement, une fenêtre apparaîtra lui indiquant une erreur. De même, si l'utilisateur ne rentre pas un nom de fichier de sortie, le programme est codé de manière à ce qu'il ne puisse pas continuer sans cela.

Finalement un manuel d'utilisation et un README sont fournis avec le logiciel. Le manuel d'utilisation a pour fonction de faciliter la prise en main du logiciel pour tout niveau possible d'utilisateur. Le README, quant à lui, comporte des informations nécessaires pour pouvoir installer le logiciel sur les différents systèmes d'exploitation comme cela été prévu : Windows et Linux. Cependant en raison du temps imparti et du manque de matériel sous MacOS à notre disposition, la version MacOS ne sera pas fournie.

Cependant il est possible de faire tourner le programme sur MacOS sans executable grâce aux lignes de commandes présent dans le README. Le script nécessaire pour faire fonctionner la compilation de l'installation (avec py2exe) sont disponibles en annexes (setup.py).
De plus, sera fournie un dossier Backup comportant deux fichiers .csv : Backup_G.csv qui permet de revenir à l'état initial de la base de données, au cas où l'utilisateur aurait effectué de mauvaises manipulations et le fichier ModelG.csv qui est la base de données entraînée avec les résultats que l'utilisateur a obtenu et validé avec la classification.

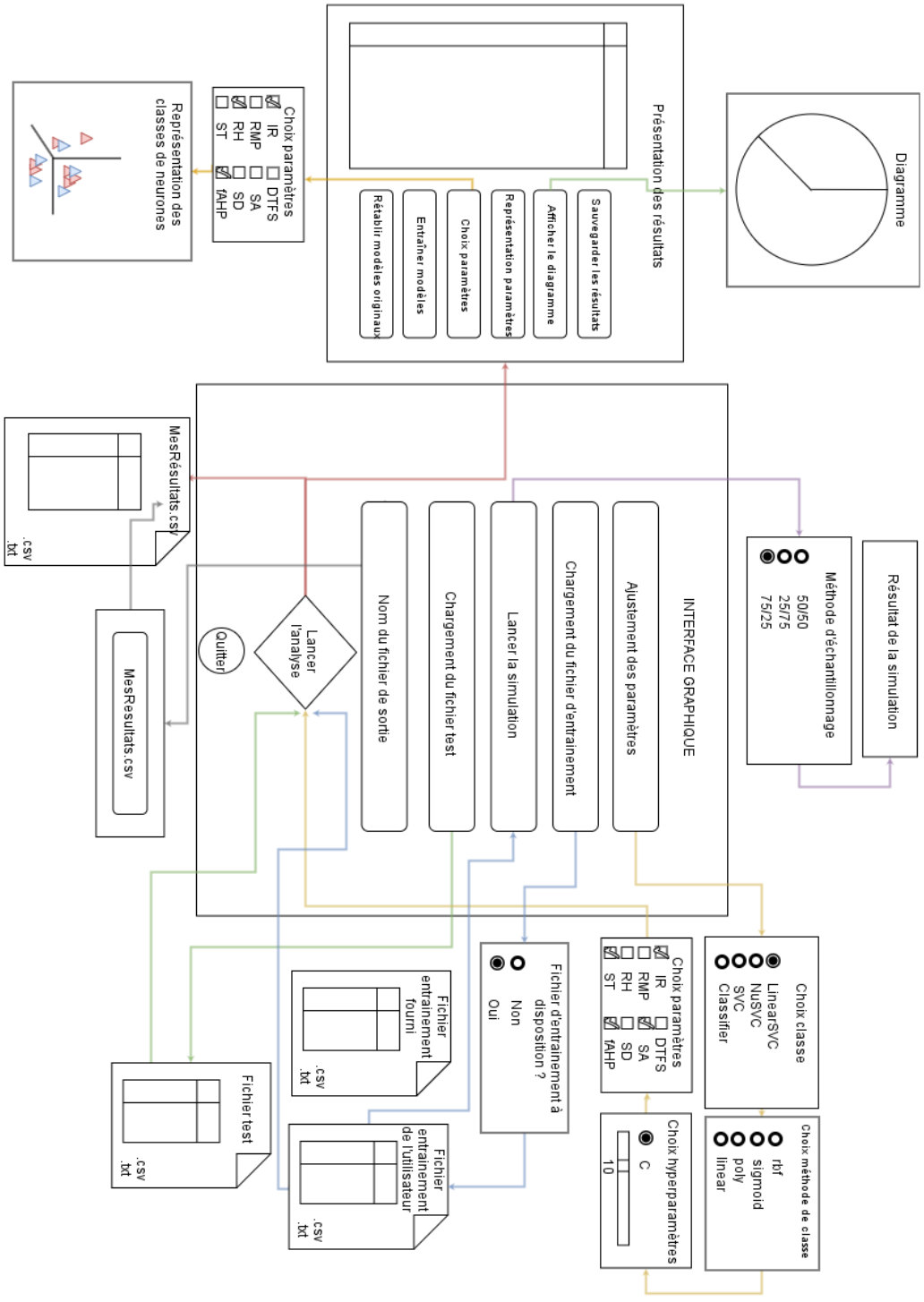


Figure 13: Figure représentant les différentes activations

### 0.3.3 Limites du programme

**Problèmes rencontrés** Le temps de calcul pour les analyses a été chronophage à cause des nombreuses boucles imbriquées pour tester les meilleures combinaisons possibles. Cependant, il aurait pu être envisagé de réduire le pas entre chaque variation des hyperparamètres, mais cela aurait demandé des temps de calculs beaucoup trop long par rapport au temps imparti.
De plus lors de l'exploration des paramètres le temps de calcul et la mémoire utilisée on conduit à des freezes des analyses, celles-ci indiquant les limites du paramètres concernés.
Un des problèmes majeur de la mise en place du programme a été la récupération des variables depuis Tkinter vers des variables plus communes. En effet Tkinter les retourne sous la forme PYVARx.
La compatibilité a également été complexe, en effet plusieurs programmes ont permis la mise en place d'un .exe pour Windows par exemple py2exe et pyinstaller. L'élaboration de ces deux programmes a rencontré bon nombre de difficultés pour faire du programme un exécutable autonome. Il a fallu mettre en place un fichier setup.py contenant le nom de toutes les sous librairies qu'ils n'arrivaient pas à atteindre.

**Freins et restrictions du programme** La principale amélioration concerne l'ajout des paramètres. En effet, il serait intéressant de permettre à un utilisateur non expérimenté de pouvoir rajouter une nouvelle méthode. Il aurait donc été judicieux avec plus de temps, de créer un fichier permettant à l'utilisateur de rentrer sa ligne contenant ses nouveaux paramètres sans avoir à rentrer dans le code, et ainsi de faire appel à ce fichier.

De plus, la partie sur l'entraînement des modèles grâce aux fichiers Backup_G et ModelG aurait pu être plus poussée et avancée avec le temps. Par exemple, ici, si l'utilisateur veut réinitialiser, il devra remplacer les deux fichiers cités précédemment par ceux disponibles dans le dossier Backup.

Lors de l'affichage des résultats sous Tkinter en forme de tableau, il a également été constaté un ralentissement au niveau de l'affichage, ce qui suggère que Tkinter n'est peut être pas adapté à l'affichage de tableau interactifs comme celui proposé.

Lors de l'appel d'une visualisation sous matplotlib la perception humaine devient limitante. En effet l'humain ne peut dépasser la visualisation en 3 dimensions ce qui limite la visualisation des résultats. On ne peut donc pas avoir de visualisation à 8 dimensions, correspondant à l'ensemble des paramètres étudiés.

# Conclusion

Ce projet consistant à développer un outil de classification automatique de signaux neuronaux biologiques a été mené à son terme. Un programme avec une gestion d'erreur intégrée a été mis en place. De plus, une interface graphique fonctionnelle permettant d'utiliser le programme à bon escient est incluse. Ce programme intègre deux méthodes de classification supervisée : les SVM et les Réseaux de Neurones.

Les résultats obtenus à la suite de l'analyse d'un fichier d'entrée par le programme donne des valeurs avoisinant les 80% de réussite. Ainsi, les prédictions proposées ne sont pas parfaites.

Il a cependant été possible de refaire ressortir une combinaison optimale (80%) avec la classe SVC, la méthode RBF, les paramètres RH,ST, SA, SD, fAHP ainsi que les hyperparamètres gamma = 10e-7 et C = 10e-7 et un échantillonnage à 75% d'entraînement et 25% de test.

Il convient cependant de noter que c'est un sujet de recherche, et qu'il n'y a, à ce jour, aucune étude permettant d'affirmer qu'il est possible de classer les neurones selon leurs paramètres électrophysiologiques.

Il peut cependant être envisagé de tester d'autres méthodes de classification supervisée afin d'améliorer les résultats ou de tester les analyses sur de plus grands jeux de données.

# Références

Tutoriels sous forme de vidéos sur la chaîne de sentdex, Youtube :

- Scikit Learn Linear SVC Example Machine Learning Tutorial with Python p. 11
- Matplotlib Tutorial 4 - Scatter Plots
- Scikit Learn Machine Learning SVM Tutorial with Python p. 2 - Example
- Python Scripts to Executables with Py2exe tutorial

[1] https://monkeylearn.com/blog/introduction-to-support-vector-machines-svm/
[2] http://www.explainthatstuff.com/introduction-to-neural-networks.html
[3] https://medium.com/@neilaamy/wtf-is-a-support-vector-machine-a45281e3f915
[4] https://www.datasciencecentral.com/profiles/blogs/some-deep-learning-with-python-tensorflow-and-keras-1?xg_sourcea=activity
[5] http://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLP Classifier.html#sklearn.neural_network.MLPClassifier
[6] http://scikit-learn.org/stable/modules/svm.html
[7] Ms. Gaurangi Patil, Ms. Varsha Galande, Mr. Vedant Kekan, Ms. Kalpana Dange, "Sentiment Analysis Using Support Vector Machine", Vol. 2, Issue 1, January 2014
[8] https://docs.python.org/2/library/tkinter.html
[9] http://scikit-learn.org/stable/modules/svm.html
[10] http://www.statsoft.fr/concepts-statistiques/reseaux-de-neurones-automatises/reseaux-de-neurones-automatises.htm#.WvlJ9K09a3c

# Annexes

Ci-dessous, la figure 14 tirée d'une étude sur la base de données Iris afin de pouvoir visualiser les modèles des différentes méthodes énumérées 2.1.1

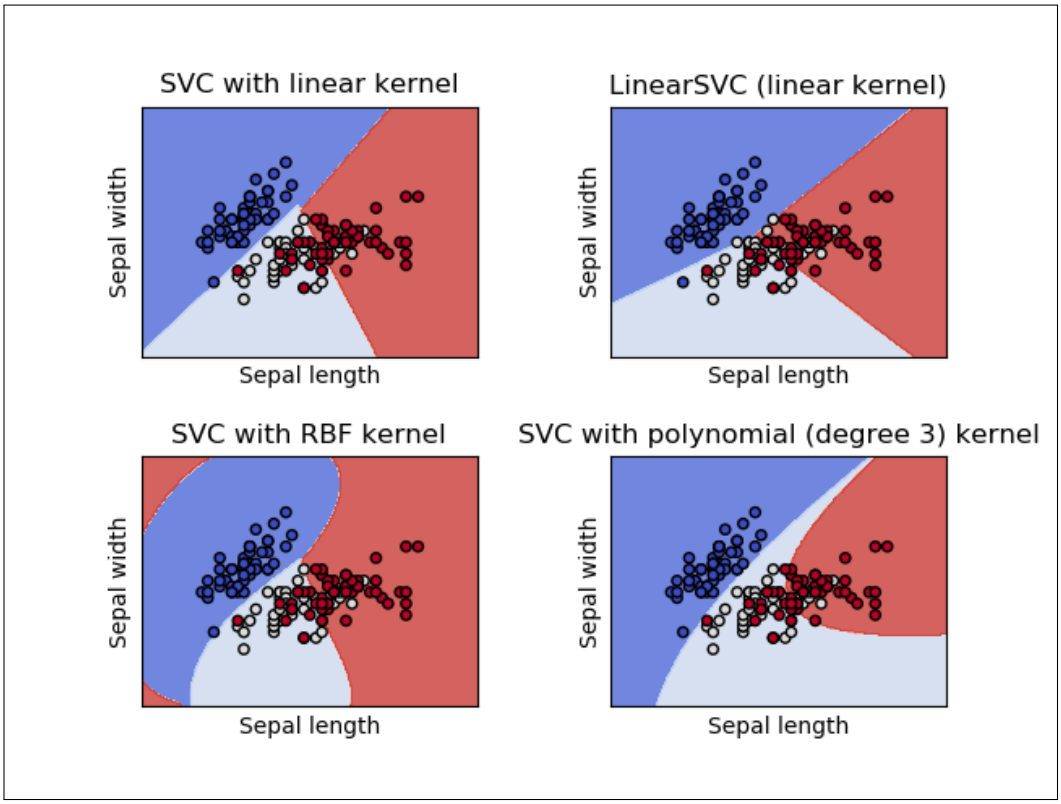


Figure 14: Schéma représentant les modélisations graphique des résultats d'une classification lambda en fonction des différentes méthodes.[9]