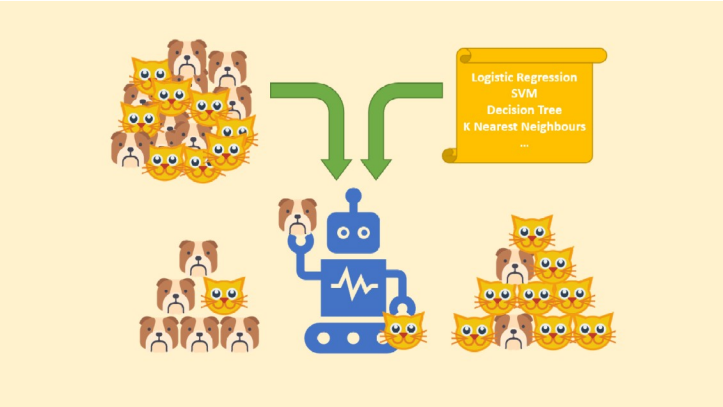
Apprentissage automatique pour la classification d'images



https://miro.medium.com/max/2886/1*xpRoFLy0BHsCr62Uf1FfTQ.png

Rapport de stage

Du 16/03/20 au 29/05/20

Étudiant: Boulli Marouan

Tuteur: Dr. Ali HAJJ HASSAN

Établissement: Université Grenoble-Alpes - UFR Sciences de l'Homme et de la société
UGA - UFR SHS 1251 Avenue Centrale CS 40700
38058 GRENOBLE CEDEX 9

Remerciements

Je tenais avant tout à remercier mon professeur de mathématiques Ali HAJJ HASSAN pour m'avoir offert cette opportunité de stage ainsi que pour son suivi et ses conseils tout au long du stage.

Sommaire

1 Introduction

1.1 Contexte

Ce stage facultatif a été réalisé au cours du second semestre de ma deuxième année en licence Miashs, au sein de l'université Grenoble-Alpes (UFR SHS). Il s'est étendu du 16/03/20 au 29/05/20.

Il s'agit d'une initiative personnelle visant à compléter ma formation, le stage ne faisant pas partie du programme de la deuxième année.

1.2 Objectifs

Ce stage avait pour objectifs :

- une introduction aux différentes techniques d'apprentissage automatique pour la classification d'images
- l'application de ces techniques sur des jeux de données réelles
- la comparaison de la performance de ces différentes techniques

1.3 Missions

Plusieurs missions ont été réalisées :

→ Réaliser une synthèse sur les techniques de pré-traitement des images et sur les différentes techniques d'apprentissage automatique

→ Tester ces techniques pour différents types de classification (mono-classe vs bi-classe vs multi-classe) sur un jeu de données réelles

→ Comparer la performance de ces techniques en optimisant les paramètres des fonctions utilisées et présenter les résultats

Je présenterai dans un premier temps une synthèse de certaines techniques de pré-traitement des données et d'apprentissage automatique utilisées pour la classification d'images; puis, j'expliquerai dans un second temps en détails la mise en application de ces techniques que j'ai réalisée à l'aide du logiciel R Studio.

2 Machine learning pour la classification d'images

2.1 Qu'est-ce que le machine learning (apprentissage automatique)?

L'objectif d'un algorithme de machine learning est de faire des prédictions sans être explicitement programmé pour cela. Il se base sur un modèle mathématique qui lui permet d'apprendre à partir de données qu'on lui fournit (training data).

« Learning is any process by which a system improves performance from experience.» (Herbert Simon)
« A computer program is said to learn from experience *E* with respect to some class of tasks *T* and performance measure *F* if its performance at tasks in *T*, as measured by *F*, improves with experience *E*.» (Tom M.Mitchell)

Ainsi, plus on fournit à l'algorithme des données, plus il apprend et plus ses prédictions sont précises.

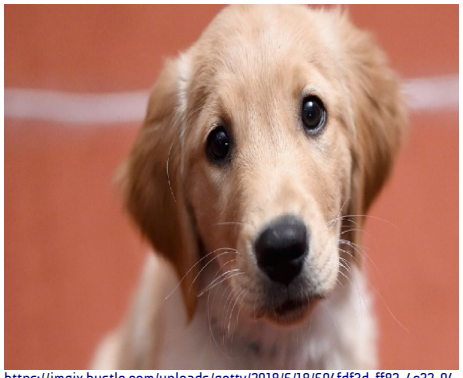
2.2 Les différents types d'apprentissage automatique

2.2.1 Apprentissage supervisé

Il s'agit d'un type d'apprentissage automatique où les données fournies sont étiquetées; c'est-à-dire qu'on précise pour chaque observation l'étiquette à laquelle elle appartient.

L'algorithme doit ensuite prédire les étiquettes de nouvelles observations.

Par exemple, admettons que l'on souhaite classifier automatiquement des images de chiens et de chats. On fournit à l'algorithme des images de chiens et de chats en lui précisant pour chaque image à quelle catégorie elle appartient, puis on utilise cet algorithme pour déterminer la classe de nouvelles images.


https://imgix.bustle.com/uploads/getty/2018/6/18/604fdf3d-ff83-4c32-9410-46e230f75fe3-getty-511711772.jpg?w=1200&h=630&q=70&fit=crop&crop=faces&fm=jpg


https://cdnfr2.img.sputniknews.com/images/102281/59/1022815949.jpg

Techniques d'apprentissage supervisé: machines à vecteurs supports (SVM), K plus proches voisins (KNN), réseaux de neurones artificiels (ANN), réseaux de neurones convolutifs (CNN),...

Je détaillerai certaines de ces techniques plus tard.

2.2.2 Apprentissage non-supervisé

C'est le même principe que pour l'apprentissage supervisé sauf que les données fournies à l'algorithme ne sont pas étiquetées; on ne lui précise pas à quelle étiquette appartient chaque observation.

https://www.mysciencework.com/media/20151129cc7873fdc715c356998f025fb4574910f97a7cc6.png

Par exemple, admettons que nous souhaitions classifier des articles scientifiques en fonction du sujet sur lequel ils portent. On fournit à l'algorithme des articles scientifiques sans préciser pour chacun d'entre eux sur quel sujet il porte, puis on utilise cet algorithme pour classifier automatiquement d'autres articles scientifiques en fonction de leur sujet.

Techniques d'apprentissage non-supervisé: K-means clustering, classification hiérarchique, ANN,...

2.2.3 Apprentissage par renforcement

Ce type d'apprentissage automatique ne nécessite pas de données étiquetées (comme le non-supervisé). Le principe est le suivant: l'algorithme adopte une stratégie en fonction des données fournies pour effectuer « ce qu'il convient de faire en différentes situations, de façon à optimiser une récompense quantitative au cours du temps. » (Wikipédia)

https://pro.ing.dk/sites/mi/files/styles/top_image/public/2019-06/d4aes5euuaauqrh.jpg?itok=0UcJh2Jw

Exemple: AWS Deep Racer est un championnat mondiale de voiture autonomes miniatures.
Ces voitures utilisent des algorithmes d'apprentissage automatique qui apprennent à ajuster la trajectoire de la voiture en analysant les données de leur environnement et grâce à une fonction de récompense qui est positive lorsque la trajectoire est bonne et négative en cas de sortie de piste ou de mauvaise trajectoire.

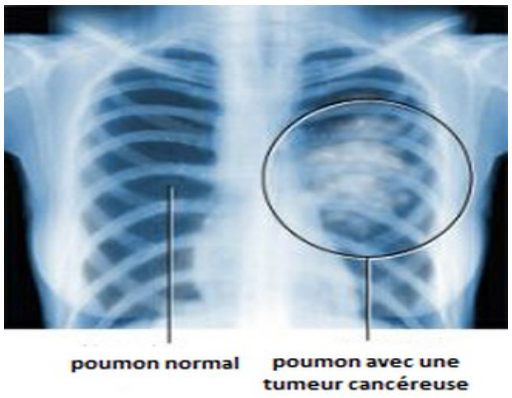
Techniques d'apprentissage par renforcement: Monte Carlo, Q-learning, Temporal difference learning,...

2.3 Qu'est-ce que la classification d'images?

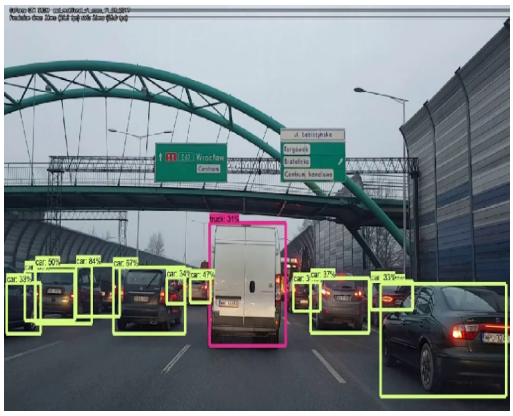
La classification d'images consiste à regrouper des images dans différentes catégories prédéfinies (comme dans l'exemple précédent chiens et chats) en fonction d'invariants caractéristiques de chaque catégorie.

Elle est notamment utilisée en vision par ordinateur et possède différentes applications; par exemple :

- la catégorisation d'images médicales pour le diagnostic d'un cancer
- la reconnaissance d'objets pour les véhicules autonomes



http://activite-culminante-le-cancer.weebly.com/uploads/1/6/3/1/16317064/6573901.png?328



https://i.ytimg.com/vi/7p2XL8wApfo/maxresdefault.jpg

2.3.1 Étapes de la classification d'images

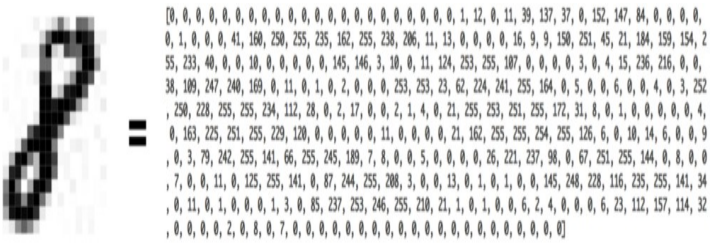
Plusieurs étapes doivent être réalisées pour classifier des images:

- **Pré-traitement de l'image**: cette étape est cruciale puisqu'elle sert à optimiser les caractéristiques de l'image afin qu'on puisse utiliser notre modèle; et aussi à supprimer les caractéristiques non pertinentes (par exemple dans de nombreux cas les couleurs ne sont pas déterminantes pour la classification).
- **Détection de l'objet**: il faut ensuite pouvoir détecter l'objet à classifier sur l'image, identifier sa localisation.
- **Extraction des caractéristiques intéressantes**: il s'agit d'identifier des motifs caractéristiques d'une catégorie à l'aide de méthode d'apprentissage statistique.
- **Classification de l'objet**: on compare les motifs de l'image avec ceux des catégories pré-définies.

Je m'attarderai uniquement sur l'étape de pré-traitement étant donné que les autres sont réalisées par le modèle d'apprentissage automatique.

2.3.2 Pré-traitement de l'image

La première chose à faire est de convertir chaque pixel de l'image en nombre afin que l'ordinateur puisse effectuer des calculs sur l'image. Voici un exemple de la manière dont est représenté l'image d'un huit écrit à la main sous forme de matrice:



https://iq.opengenus.org/basics-of-machine-learning-image-classification-techniques/

Ensuite, différentes étapes doivent être réalisées pour pré-traiter l'image:

- **Lecture de l'image**: en créant une variable dans laquelle on stocke le chemin des données des images à traiter; et en créant une fonction permettant de charger les données des images dans des matrices.
- **Redimensionnement des images**: il s'agit là de normaliser la taille des images pour faciliter les calculs sur les matrices.
- **Enrichissement des données des images**: cette étape a pour but de générer davantage de données à partir de celles disponibles, et de prévenir le sur-apprentissage (c'est-à-dire une mauvaise généralisation lors de la prédiction du modèle).

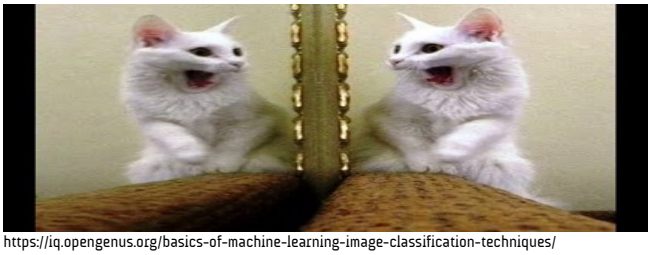
Voici différentes techniques d'enrichissement des données:

- **Niveaux de gris**: chaque pixel est convertit en nombre représentant le degré de noirceur.

https://iq.opengenus.org/basics-of-machine-learning-image-classification-techniques/

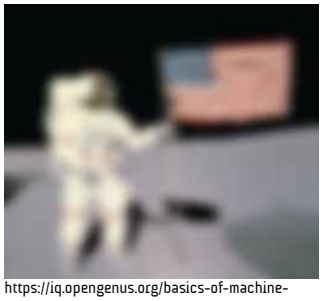
https://iq.opengenus.org/basics-of-machine-learning-image-classification-techniques/

- **Image symétrique**: obtention de l'image symétrique par rapport à l'image de départ.



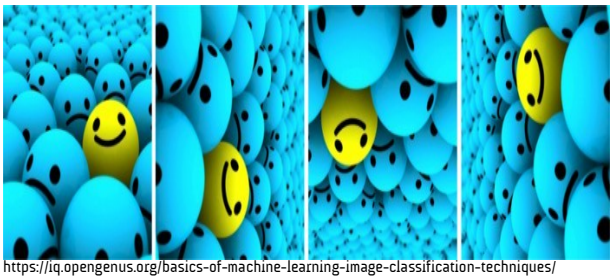
https://iq.opengenus.org/basics-of-machine-learning-image-classification-techniques/

- **Flou gaussien**: obtention de l'image floutée à l'aide d'une fonction gaussienne.



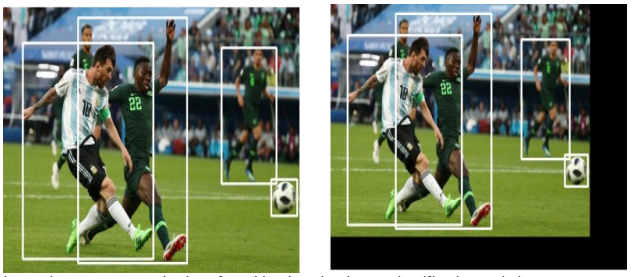
https://iq.opengenus.org/basics-of-machine-learning-image-classification-techniques/

- **Rotation de l'image**: on effectue plusieurs rotations de 90° de l'image de départ.



https://iq.opengenus.org/basics-of-machine-learning-image-classification-techniques/

- **Translation**: l'image est déplacée selon l'axe des abscisses ou des ordonnées pour que l'algorithme apprenne à reconnaître l'objet quelle que soit sa position sur l'image.



https://iq.opengenus.org/basics-of-machine-learning-image-classification-techniques/

2.4 Techniques d'apprentissage automatique: SVM et KNN

J'ai également effectué des recherches sur d'autres techniques comme les arbres de décision, les réseaux de neurones artificiels et convolutifs (ANN et CNN). Néanmoins je ne m'attarderai ici que sur les techniques SVM et KNN étant donné que ce sont celles que j'ai utilisées pour la mise en application.

2.4.1 Machines à vecteurs supports (SVM)

Cette technique d'apprentissage supervisé, initialement utilisée pour les cas de classification bi-classes (il existe cependant des extensions pour les cas mono-classe et multi-classes), consiste à séparer deux classes par une limite linéaire que l'on nomme hyperplan.

L'hyperplan qui permet de minimiser le risque d'erreur est celui qui possède une marge maximale, c'est-à-dire qu'il se situe à la plus grande distance entre les points les plus proches des deux classes.

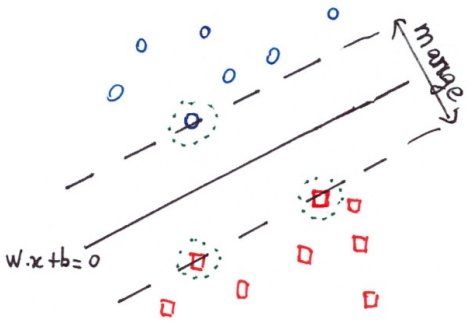


Figure 1: Marge maximale

Il existe cependant des cas où la variance des observations des classes dans les données d'apprentissage est forte, auquel cas on pourrait avoir quelques observations isolées d'une classe donnée se retrouvant proches des observations de l'autre classe.

Figure 2: Hard margin

On s'expose alors, si l'on choisit un hyperplan séparant strictement les observations des deux classes (**hard margin**, voir figure 2), à un risque de sur-apprentissage.

Une solution serait d'accepter quelques erreurs de classification lors de l'apprentissage et ainsi choisir un hyperplan qui accepterait de classifier certaines observations isolées d'une classe donnée dans l'autre classe (**soft margin**, voir figure 3).

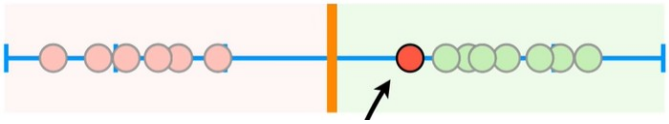


Figure 3: Soft margin

Je viens en fait d'illustrer ici le **dilemme biais-fluctuation** (bias-variance tradeoff) concernant l'apprentissage supervisé: a-t-on intérêt à utiliser un modèle se trompant peu sur les données d'apprentissage ou bien un modèle acceptant davantage d'erreurs lors de l'apprentissage?
Le problème consiste à savoir quel nombre d'erreurs accepter lors de l'apprentissage pour optimiser la précision de prédiction du modèle; car si le nombre d'erreurs accepté lors de l'apprentissage est trop faible, on s'expose à un risque de sur-apprentissage (mauvaise généralisation), et s'il est trop élevé, on s'expose à un risque de sous-apprentissage.

Pour optimiser ce paramètre du nombre d'erreurs à accepter lors de l'apprentissage, on peut utiliser la technique du **cross validation**.

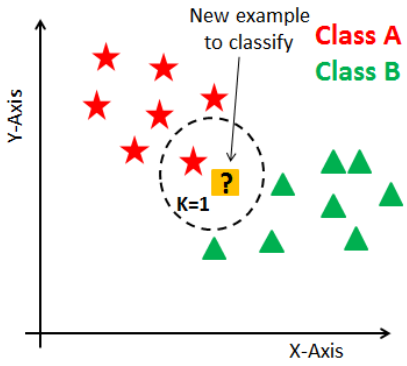
Le cross validation consiste à faire varier les valeurs des paramètres du modèle et d'effectuer plusieurs tests sur les données d'apprentissage afin de sélectionner la valeur donnant la meilleure précision de prédiction. Une fois la valeur optimale trouvée, on l'utilise pour faire les prédictions sur les données test. Différentes techniques de cross validation existent:

- **Validation set** (ou encore hold out, data split): on sépare les données d'apprentissage en deux partitions dont l'une est utilisée pour l'apprentissage et l'autre pour le test (par exemple 70% des données pour l'apprentissage du modèle et 30% pour le test). Cette technique est utile surtout lorsque le nombre de données est très grand, néanmoins, elle est soumise à un biais d'échantillonnage puisque ce dernier est aléatoire (on ne sait pas si le découpage des données est optimal).
- **K-fold cross validation**: on sépare les données d'apprentissage en k partitions (en général 5 ou 10 partitions). On utilise k-1 partitions pour l'apprentissage et 1 pour le test. Cette opération est répétée k fois afin que chacune des partitions ait joué le rôle de test. Le problème de cette technique réside dans le fait que le nombre d'éléments par classe dans chaque partition peut être déséquilibré.
- **Stratified k-fold cross validation**: on utilise le même procédé que pour la technique précédente, seulement, elle veille à ce que le nombre d'éléments par classe dans chaque partition soit équilibré.
- **Leave One Out Cross Validation (LOOCV)**: une donnée est utilisée comme test et tout le reste sert à l'apprentissage du modèle (par exemple si on a 1000 données dans l'échantillon d'apprentissage, on utilise 999 pour l'apprentissage et 1 pour le test). On répète cette opération autant de fois qu'il y a de données afin que chaque donnée ait été utilisée comme test une fois. Cette technique est peu utilisée étant donné le temps de traitement considérable qu'elle nécessite.

2.4.2 K plus proches voisins (KNN)

Cette technique d'apprentissage supervisé relativement simple consiste à compter le nombre de voisins les plus proches d'une observation que l'on souhaite classifier. Il s'agit ensuite de voir quelle classe est la plus représentée parmi ces voisins.

Par exemple, prenons deux classes A et B. On dispose d'un set de données étiquetées (A ou B) et on souhaite classer une nouvelle observation. On définit tout d'abord le nombre de voisins à considérer (k) puis on observe les k voisins les plus proches de l'observation à classer. S'il y a, parmi les k voisins, davantage d'observations appartenant à la classe A, alors l'observation sera classée dans la classe A; dans le cas contraire elle sera classée dans la classe B.



https://iq.opengenus.org/basics-of-machine-learning-image-classification-techniques/

Un problème auquel est confronté cette technique concerne la notion de voisins « les plus proches ». En effet, cette formulation est imprécise et il faut clairement définir le type de distance à utiliser pour rendre compte de la proximité des voisins. Les distances les plus utilisées sont la distance Euclidienne et celle de Manhattan.

Il faut aussi trouver la valeur du k pour laquelle le modèle est le plus précis. Pour cela, on peut utiliser une des techniques du cross validation évoquées précédemment.

3 Mise en application (R Studio)

Passons maintenant à la mise en application des deux techniques d'apprentissage supervisé évoquée précédemment (SVM et KNN).

Je commencerai d'abord par présenter la base de données utilisée (MNIST), puis je détaillerai les applications réalisées (avec méthode et résultats).

3.1 Base de données MNIST

MNIST (Modified National Institute of Standards and Technology) est une base de données comportant plusieurs images de chiffres écrits à la main. Elle comporte:

- 60000 images dans les données d'apprentissage
- 10000 images dans les données test

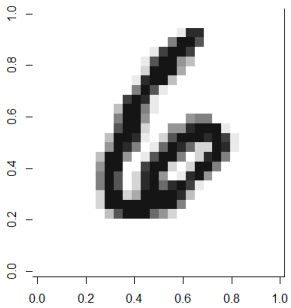
Ces images ont déjà été pré-traitées:

- transformées en niveaux de gris
- taille normalisée (28*28 pixels)
- chiffres centrés sur l'image
- images floutées

J'ai récupéré cette base de données sur le site de Yann LeCun (http://yann.lecun.com/exdb/mnist/).
Le format des fichiers ne correspondant pas à un format standard d'images, il a fallut que je trouve un code permettant de les charger sur R Studio. J'ai récupéré ce code sur le forum github (https://gist.github.com/daviddalpiaz/ae62ae5ccd0bada4b9acd6dbc9008706). Il comporte notamment:

- 2 fonctions permettant de charger les images et leurs labels (étiquettes) correspondant
- 4 variables pour stocker les données d'apprentissage et de test, les labels des données d'apprentissage et ceux des données test
- 1 fonction permettant de visualiser les images

Voici un exemple d'image correspondant à un 6:



3.2 Modèle SVM linéaire (multi-classes)

Il s'agit du premier essai que j'ai réalisé sur ces données.

3.2.1 Méthode

Le nombre de données d'apprentissage étant assez conséquent, j'ai réalisé un échantillon de 5000 données (parmi les 60000) pour diminuer le temps de traitement.

Les librairies utilisées sont les suivantes:

- kernlab: pour la fonction ksvm servant à entraîner le modèle.
- Caret: pour la fonction confusionMatrix servant à présenter les résultats de prédictions du modèle dans une matrice.

La fonction ksvm possède deux paramètres particulièrement importants:

- Le kernel (noyau): il s'agit d'une astuce utilisée pour traiter les cas où les données ne sont pas linéairement séparables. Plusieurs noyaux existent, parmi lesquels le noyau linéaire utilisé ici.
- Le paramètre C (cost of constraints violation): il désigne le coût appliqué aux erreurs lors de l'apprentissage. Plus il est élevé, moins l'algorithme accepte d'erreurs lors de l'apprentissage; moins il est élevé, plus on accepte d'erreurs lors de l'apprentissage. Ici, la valeur par défaut a été gardée (C=1).

Une fois le modèle entraîné sur les 5000 données d'apprentissage, il est testé sur les 10000 données test.

3.2.2 Résultats

Voici les résultats de la matrice de confusion:

```
          Reference
Prediction    0    1    2    3    4    5    6    7    8    9
         0  953    0    8    5    2   15    9    0   17    3
         1    0 1117   13    4    0    5    2   10   14    6
         2    2    5  933   24    7    9   13   21   27    6
         3    3    2   15  883    0   47    1    8   26   10
         4    2    0    9    2  934    7    9   17    7   66
         5   13    4    1   33    0  753   18    0   38    9
         6    4    1   10    2    4   10  904    0   10    1
         7    1    0   16   14    3    3    1  939   11   19
         8    1    6   25   34    3   37    1    3  816   11
         9    1    0    2    9   29    6    0   30    8  878

Overall Statistics

               Accuracy : 0.911
```

Les lignes correspondent aux résultats prédits et les colonnes aux chiffres réels des données test.

Si on prend par exemple le chiffre à la ligne 0 et la colonne 2, on voit que le modèle a prédit 8 fois un 2 alors qu'il s'agissait d'un 0.

On a sous la matrice la **précision** (Accuracy) du modèle, qui est obtenue en sommant le nombre de bonnes prédictions (trace de la matrice) que l'on divise par le nombre total de prédictions réalisées (10000). Ici, on a obtenu une précision de 91.1%.

D'autres résultats accompagnent la matrice de confusion:

Statistics by Class:

	Class: 0	Class: 1	Class: 2	Class: 3	Class: 4	Class: 5	Class: 6	Class: 7	Class: 8	Class: 9
Sensitivity	0.9724	0.9841	0.9041	0.8743	0.9511	0.8442	0.9436	0.9134	0.8378	0.8702
Specificity	0.9935	0.9939	0.9873	0.9875	0.9868	0.9873	0.9954	0.9924	0.9866	0.9905
Pos Pred Value	0.9417	0.9539	0.8911	0.8874	0.8870	0.8665	0.9556	0.9325	0.8709	0.9117
Neg Pred Value	0.9970	0.9980	0.9889	0.9859	0.9946	0.9848	0.9940	0.9901	0.9826	0.9855
Prevalence	0.0980	0.1135	0.1032	0.1010	0.0982	0.0892	0.0958	0.1028	0.0974	0.1009
Detection Rate	0.0953	0.1117	0.0933	0.0883	0.0934	0.0753	0.0904	0.0939	0.0816	0.0878
Detection Prevalence	0.1012	0.1171	0.1047	0.0995	0.1053	0.0869	0.0946	0.1007	0.0937	0.0963
Balanced Accuracy	0.9830	0.9890	0.9457	0.9309	0.9690	0.9157	0.9695	0.9529	0.9122	0.9304

Parmi ces critères (en plus de l'accuracy déjà mentionnée), deux nous intéresseront particulièrement:

- La **sensibilité** (sensitivity): correspond à la proportion de bonnes prédictions pour une classe donnée par rapport au nombre total d'observations pour cette même classe. Par exemple, pour la classe 0, on voit sur la matrice de confusion qu'il y a 953 bonnes prédictions pour 980 observations. D'où une sensibilité pour la classe 0 de: 953/980 = 0.9724.
- La **spécificité** (spécificity): correspond à la proportion d'éléments n'appartenant pas une classe reconnus comme tels. Par exemple, admettons qu'on ait 1000 exemples de chaque chiffre (de 0 à 9) et que le modèle ait 8750 bonnes prédictions pour les chiffres de 1 à 9, et 50 mauvaises prédictions pour la classe 0 . Ainsi, la spécificité pour la classe 0 serait de 8750/(8750+50).

3.3 Comparaison modèle SVM vs KNN (multi-classes)

L'objectif ici est de savoir lequel des modèles SVM (linéaire et gaussien) ou KNN est le plus précis pour la classification des chiffres écrits à la main.

Cette comparaison a été réalisée en deux fois: d'abord SVM linéaire vs KNN, puis SVM gaussien vs KNN, en utilisant la commande set.seed(100) pour que l'échantillonnage reste le même.

La librairie class comportant la fonction knn est utilisée pour entraîner le modèle KNN.

3.3.1 Méthode

La méthodologie utilisée se décrit comme suit:

- Sélection aléatoire d'un échantillon de 2000 images (parmi les 60000) pour les données d'apprentissage.
- Entraînement du modèle SVM (kernel linéaire ou gaussien) avec paramètres par défaut.
- Création des partitions pour l'optimisation du k (pour le modèle KNN) avec la technique du validation set (70% des données pour l'apprentissage et 30% pour le test).
- Entraînement du modèle KNN avec optimisation du k (valeurs allant de 1 à 15).
- Prédiction des modèles (SVM linéaire/gaussien et KNN) sur les 10000 données test de MNIST.
- Calcul de la précision et des sensibilités pour chacun des modèles.

Toutes ces étapes sont répétées 50 fois afin d'obtenir une moyenne pour les précisions et les sensibilités.

3.3.2 Résultats SVM linéaire vs KNN

accuracy_svm	accuracy_knn
89.5%	89.7%

Figure 4: Précisions SVM linéaire vs KNN

	sensitivities_svm	sensitivities_knn
Classe 0	96.1 %	97.6 %
Classe 1	98.1 %	99.5 %
Classe 2	86.4 %	83.5 %
Classe 3	86.5 %	89.7 %
Classe 4	90.7 %	84.9 %
Classe 5	83.6 %	86.3 %
Classe 6	92.9 %	95.7 %
Classe 7	90.0 %	89.9 %
Classe 8	82.7 %	80.9 %
Classe 9	86.2 %	87.8 %

Figure 5: Sensibilités SVM linéaire vs KNN

3.3.3 Résultats SVM gaussien vs KNN

accuracy_svm	accuracy_knn
92.6 %	89.7%

Figure 6: Précisions SVM gaussien vs KNN

	sensitivities_svm	sensitivities_knn
Classe 0	97.7 %	97.6 %
Classe 1	98.5 %	99.5 %
Classe 2	89.9 %	83.5 %
Classe 3	91.1 %	89.7 %
Classe 4	92.8 %	84.9 %
Classe 5	90.0 %	86.3 %
Classe 6	95.3 %	95.7 %
Classe 7	90.8 %	89.9 %
Classe 8	88.6 %	80.9 %
Classe 9	90.2 %	87.8 %

Figure 7: Sensibilités SVM gaussien vs KNN

3.3.4 Remarques

J'ai rencontré certaines difficultés concernant l'optimisation des paramètres pour les modèles SVM. J'ai tout d'abord tenté une optimisation avec la méthode du k-fold cross validation (en utilisant la fonction train de la librairie caret), néanmoins le temps de traitement était trop long pour pouvoir répéter 50 fois les opérations décrites plus haut.

Puis, j'ai essayé de réaliser cette optimisation manuellement en utilisant une technique de grid search qui consiste à placer les différentes valeurs des paramètres qu'on veut comparer dans une matrice, puis à l'aide d'une boucle, entraîner le modèle avec chacune des valeurs contenues dans la matrice. Cette méthode n'a pas fonctionné.

C'est pourquoi les paramètres des modèles SVM ont été conservés par défaut.

3.4 Comparaison modèle SVM (gaussien) vs KNN (bi-classes)

L'objectif est de réitérer la comparaison effectuée précédemment sur un cas de classification bi-classes.

Cette fois-ci, seul le modèle SVM gaussien est retenu car il donne en général de meilleures résultats que le modèle SVM linéaire (après quelques recherches et tests préalables comparant la performance du modèle SVM avec les deux types de noyau).

Pour simuler le cas bi-classes tout en conservant la base de données MNIST, seuls les chiffres 4 et 9 ont été sélectionnés.

3.4.1 Méthode

La méthodologie est similaire à celle utilisée pour le cas multi-classes:

- Sélection d'un échantillon aléatoire de 1000 images contenant des 4 et des 9 (à proportions égales) pour les données d'apprentissage.
- Sélection d'un échantillon de 1991 images contenant tous les 4 et 9 présents dans les données test MNIST (10000 images), pour les données test.
- Entraînement du modèle SVM gaussien avec paramètres par défaut.
- Entraînement du modèle KNN avec optimisation du k en utilisant la méthode du validation set, comme détaillé précédemment.
- Prédiction des modéles sur l'échantillon test (1991 images)
- Calcul des précisions, sensibilités et spécificités pour chacun des modèles.

De même que pour le cas multi-classes, ces étapes sont répétées 50 fois pour obtenir les moyennes de précision, sensibilité et spécificité.

3.4.2 Résultats

	accuracy	sensitivity	specificity
svm	97.3 %	97.3 %	97.3 %
knn	96.3 %	94.1 %	98.4 %

Figure 8: Résultats SVM gaussien vs KNN (bi-classes)

3.4.3 Remarque

J'ai tenté de réaliser une classification bi-classes sur un autre jeu de données (base de données cats vs dogs récupérée sur kaggle), néanmoins les résultats obtenus sont trop faibles (que ce soit avec le modèle SVM ou KNN) sans que je puisse trouver la raison à cela (peut-être un problème au niveau du pré-traitement des données, en sachant la plupart des personnes ayant effectué la classification sur ce jeux de données ont utilisé des réseaux de neurones).

3.5 One-class classification (mono-classe)

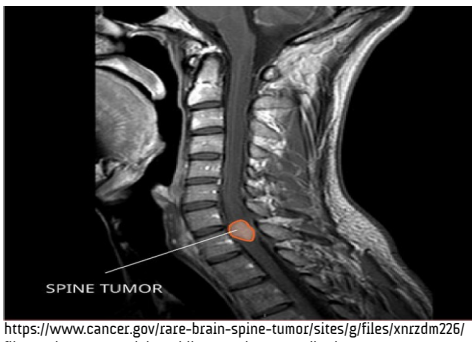
3.5.1 Principe

L'objectif de ce type de classification est de reconnaître des objets appartenant à une classe donnée et savoir les différencier d'autres objets n'appartenant pas à cette classe (on parle de détection d'anomalies ou encore **novelty detection**).

Le modèle est entraîné sur des données non-étiquetées contenant uniquement des objets appartenant à la classe cible (**classe normale**).
La prédiction se fait sur des données test contenant à la fois des objets appartenant à la classe normale et des objets n'appartenant pas à cette classe (**anomalies**).

La détection d'anomalies est par exemple utilisée pour la détection d'une activité anormale dans un réseau internet ou la détection d'une tumeur sur une IRM.


https://cdn.comparitech.com/wp-content/uploads/2018/04/Network-Security.jpg



https://www.cancer.gov/rare-brain-spine-tumor/sites/g/files/xnrzdm226/files/styles/cgov_article/public/cgov_image/media_image/100/200/8/files/Ependymoma%20%28Article%29.jpg?h=3753a9da&itok=NwWiArIn

3.5.2 Méthode

La méthodologie utilisée est comme suit:

- Sélection d'un échantillon aléatoire 1000 images de 4 pour les données d'apprentissage (la classe 4 est utilisée pour simuler la classe normale).
- Sélection d'un échantillon de 1991 images contenant tous les 4 et 9 présents dans les données test MNIST (10000 images), pour les données test (la classe 9 est utilisée pour simuler les anomalies).
- Entraînement d'un modèle SVM gaussien avec paramètres par défaut, sauf le paramètre nu.
- Prédiction du modèle sur l'échantillon test.
- Calcul de la précision, de la sensibilité et de la spécificité.

6 valeurs différentes sont considérées pour le paramètre nu du modèle SVM gaussien: 0.01, 0.05, 0.1, 0.15, 0.2 et 0,3.
Chacune des étapes détaillées est répétée 50 fois pour chaque valeur de nu, et les moyennes de précision, sensibilité et spécificités sont calculées.

La valeur du paramètre nu correspond à la limite supérieure de la proportion d'erreurs autorisée lors de l'apprentissage et la limite inférieure de la proportion de vecteurs supports utilisés par rapport au nombre total des données d'apprentissage. Il prend des valeurs allant de 0 à 1.

Par exemple, pour nu = 0.1, on autorise au maximum 10% d'erreurs lors de l'apprentissage et on utilise un nombre de vecteurs supports supérieur à 10% des données d'apprentissage.

3.5.3 Résultats

A noter que la spécificité joue un rôle important pour mesurer la capacité du modèle à détecter les anomalies. Une bonne performance générale pour le one-class classification se traduit par un bon équilibre entre spécificité et sensibilité.

Nu	accuracy	sensitivity	specificity
0.01	59.3 %	93.8 %	25.9 %
0.05	59.3 %	92.3 %	27.2 %
0.1	60.3 %	87.7 %	33.7 %
0.15	61.1 %	83.3 %	39.5 %
0.2	61.9 %	78.7 %	45.4 %
0.3	62.7 %	69.9 %	55.6 %

Figure 9: Résultats One-class classification SVM gaussien

4 Bilan personnel

Voici globalement ce que je retire de ce stage:

- Une bonne introduction au machine learning avec la découvertes des différentes techniques utilisées

- Une démystification de la manière dont on pratique le machine learning avec la mise en application de certaines des techniques sur le logiciel R Studio; malgré que la compréhension du fonctionnement des algorithmes utilisés reste superficielle

- L'importance de la compréhension des données utilisées et de la maîtrise de leur pré-traitement (je l'ai particulièrement remarqué lorsque j'ai dû trouver un moyen d'importer les données et lors de l'application des fonctions SVM et KNN sur ces données)

- La découverte de différents types de classification et d'une méthode pour la détection d'anomalies

- Des méthodes pour optimiser les paramètres du modèle et pour la comparaison de l'efficacité de différents modèles

- L'habitude d'utiliser la recherche web pour résoudre un problème ou clarifier des points mal compris (recherche de codes sur les forums, tutoriels, spécifications de fonctions,...)

- Une meilleure aisance avec le logiciel R

- L'envie d'en apprendre davantage sur le sujet et sur les applications qui en découlent

5 Références

- Techniques de machine learning pour la classification d'images : https://iq.opengenus.org/basics-of-machine-learning-image-classification-techniques/
- Introduction aux réseaux de neurones convolutifs: https://developers.google.com/machine-learning/practica/image-classification/convolutional-neural-networks
- Cours d'introduction à l'apprentissage automatique école des Mines Nancy: https://members.loria.fr/FSur/enseignement/apprauto/poly_apprauto_FSur.pdf
- Apprentissage par renforcement et conduite autonome (AWS Deep Racer): https://www.quantmetry.com/reinforcement-learning-conduite-autonome-essais-aws-deep-racer/
- Charger les données MNIST sur R: https://gist.github.com/daviddalpiaz/ae62ae5ccd0bada4b9acd6dbc9008706
- Conférence Deep learning, Yann LeCun: https://www.youtube.com/watch?v=RgUcQceqC_Y
- Base de données MNIST: http://yann.lecun.com/exdb/mnist/
- Explication du paramètre C (cost of constraints) de la fonction ksvm: https://stats.stackexchange.com/questions/225409/what-does-the-cost-c-parameter-mean-in-svm?rq=1
- Forum github: https://github.community/
- Forum stackoverflow: https://stackoverflow.com/
- Méthodes pour évaluer le meilleur modèle sur R avec la librairie caret: https://machinelearningmastery.com/how-to-estimate-model-accuracy-in-r-using-the-caret-package/
- Chaîne youtube StatQuest: https://www.youtube.com/channel/UCtYLUTtgS3k1Fg4y5tAhLbw

- Chaîne youtube du Dr Bharatendra Rai (tutoriels sur le machine learning): https://www.youtube.com/channel/UCuWECsa_za4gm7B3TLgeV_A

Annexes

Annexe 1 – Code R chargement des données MNIST

```
################# Chargement des datasets ######################

# Les fichiers nécessitaient un traitement pour être lus sous R, j'ai récupéré ce code à l'adresse suivante :
https://gist.github.com/daviddalpiaz/ae62ae5ccd0bada4b9acd6dbc9008706

# J'ai simplement modifié le nom des variables contenant les datasets : train, test, train$y et test$y


# modification of https://gist.github.com/brendano/39760

# automatically obtains data from the web

# creates two data frames, test and train

# labels are stored in the y variables of each data frame

# can easily train many models using formula `y ~ .` syntax


# download data from http://yann.lecun.com/exdb/mnist/

download.file("http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz",

              "train-images-idx3-ubyte.gz")

download.file("http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz",

              "train-labels-idx1-ubyte.gz")

download.file("http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz",

              "t10k-images-idx3-ubyte.gz")

download.file("http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz",

              "t10k-labels-idx1-ubyte.gz")


install.packages('R.utils')


# gunzip the files

R.utils::gunzip("train-images-idx3-ubyte.gz")

R.utils::gunzip("train-labels-idx1-ubyte.gz")

R.utils::gunzip("t10k-images-idx3-ubyte.gz")
```

```
R.utils::gunzip("t10k-labels-idx1-ubyte.gz")
```

```
# load image files
load_image_file = function(filename) {
  ret = list()
  f = file(filename, 'rb')
  readBin(f, 'integer', n = 1, size = 4, endian = 'big')
  n    = readBin(f, 'integer', n = 1, size = 4, endian = 'big')
  nrow = readBin(f, 'integer', n = 1, size = 4, endian = 'big')
  ncol = readBin(f, 'integer', n = 1, size = 4, endian = 'big')
  x = readBin(f, 'integer', n = n * nrow * ncol, size = 1, signed = FALSE)
  close(f)
  data.frame(matrix(x, ncol = nrow * ncol, byrow = TRUE))
}

# load label files
load_label_file = function(filename) {
  f = file(filename, 'rb')
  readBin(f, 'integer', n = 1, size = 4, endian = 'big')
  n = readBin(f, 'integer', n = 1, size = 4, endian = 'big')
  y = readBin(f, 'integer', n = n, size = 1, signed = FALSE)
  close(f)
  y
}

# load images
train = load_image_file("train-images-idx3-ubyte") # images en ligne et pixels en colonne
test  = load_image_file("t10k-images-idx3-ubyte")

# load labels
train$y = as.factor(load_label_file("train-labels-idx1-ubyte"))  # on définit ici les étiquettes comme des niveaux pour
que le modèle les considère comme des catégories et non pas des scores
```

```
test$y  = as.factor(load_label_file("t10k-labels-idx1-ubyte"))   # vecteurs comportant les labels


# helper function for visualization
show_digit = function(arr784, col = gray(12:1 / 12), ...) {
  image(matrix(as.matrix(arr784[-785]), nrow = 28)[, 28:1], col = col, ...)
}
```

Annexe 2 – Code R modèle SVM linéaire

```
install.packages('kernlab')
install.packages('caret')

library(kernlab) # contient ksvm
library(caret)  # contient confusion matrix

set.seed(100) # pour fixer la graine du générateur aléatoire
              # et obtenir les mêmes résultats avec sample par exemple d'une session à l'autre ou même d'un ordi à l'autre

# échantillon car temps de traitement trop long avec toutes les données

sample_indices <- sample(1: nrow(train), 5000) # extracting subset of 5000 samples for modelling
train <- train[sample_indices, ]



######################## modèle svm avec noyau linéaire   kernel = vanilladot, C = 1 ######################


svm_linear <- ksvm(train$y ~ ., data = train, scaled = FALSE, kernel = "vanilladot", C = 1)
# kernel fonction de similarité, sorte de distance entre les observations, vanilladot = linéaire
# scaled = TRUE  normalise les données, soustrait la moyenne et divise par l'écart-type, pas besoin ici
# C = cost of constraint violation,  plus il est grand, plus on pénalise les obs dans les marges --> risque de sur-apprentissage et donc baisse de la capacité de généralisation
# plus il est petit, moins on pénalise les obs dans les marges --> risque de sous-apprentissage

svm_linear
```

```
eval_svm_linear <- predict(svm_linear, newdata = test)
eval_svm_linear

confusionMatrix(eval_svm_linear, test$y)

# Précision = 91.1 %
```

Annexe 3 – Code R comparaison SVM vs KNN (multi-classes)

```
library(class)
library(kernlab)
library(caret)

################ choix nombre d'échantillonnages #################

n=50

set.seed(100)

################# Initialisation matrices et vecteurs ###################

vect_accuracy_svm <- NULL
vect_accuracy_knn <- NULL

matrix_sensitivities_svm = matrix(0, ncol=10, nrow=50,byrow=T)
matrix_sensitivities_knn = matrix(0, ncol=10, nrow=50,byrow=T)

accuracy_svm = 0




pt<-proc.time()
for(i in 1:n){

  ############## Echantillon #################
```

```
sample_indices <- sample(1: nrow(train), 2000) # extracting subset of 2000 samples for modelling
train_svm_knn <- train[sample_indices, ]


############ Partitions ############

partition <- createDataPartition(train_svm_knn$y,times=1,p=0.70,list=F)
train_partition <- train_svm_knn[partition,]
test_partition <- train_svm_knn[-partition,]




############ Combinaisons des paramètres svm ############

# p = ncol(train_partition)
# grid_svm <- expand.grid(cost=1,sigma=c(1/p,2/p,1/(2*p)))


###################### Modèle svm ########################


  model_svm <- ksvm(train_partition$y ~ ., data = train_partition ,kernel = "rbfdot", C = 1,
                    kpar = "automatic", scaled = F )




  # if(cm_svm$overall[['Accuracy']] > Accuracy_svm){
  # Accuracy_svm = cm_svm$overall[['Accuracy']]
  #   best_cost = grid_svm[i,1]
  #   best_gamma = grid_svm[i,2]
```

```
# }

# cat("Best accuracy model reached for cost = ",best_cost,"\n","gamma = ",best_gamma,"\n", "with accuracy =
",Accuracy_svm,sep=" ","\n")

# best_model_svm <- svm(train_partition$y ~ ., data = train_partition,kernel="radial",
                      # cost = best_cost, gamma = best_gamma, scale = FALSE)



################## modèle knn ################

Accuracy_knn = 0

for(j in 1:15){
  model_knn <- knn(train = train_partition[,-785], test = test_partition[,-785],cl = train_partition$y, k = j)



  cm_knn <- confusionMatrix(model_knn, test_partition$y)



  if(cm_knn$overall[['Accuracy']] > Accuracy_knn){
    Accuracy_knn = cm_knn$overall[['Accuracy']]
    best_k = j
  }
}
cat("Best accuracy model reached for k = ",best_k,"\n", "with accuracy = ",Accuracy_knn,sep=" ","\n")



################### test svm sur les 10000 ###################

eval_svm <- predict(model_svm, newdata = test)

cm_svm <- confusionMatrix(eval_svm, test$y)
```

```
    ################ Récupération accuracy sensitivities svm ###################

    accuracy_svm <- cm_svm$overall[['Accuracy']]
    sensitivities_svm <- cm_svm$byClass[,"Sensitivity"]



    vect_accuracy_svm <- c(vect_accuracy_svm,accuracy_svm)

    matrix_sensitivities_svm[i,] <- sensitivities_svm

    ################## Test knn sur 10000 avec meilleure valeur de k ###################

    best_model_knn <-  knn(train = train_partition[,-785], test = test[,-785],cl = train_partition$y, k = best_k)
    cm_best_knn <- confusionMatrix(best_model_knn, test$y)

    ################## Récupération accuracy et sensitivities knn ###################

    accuracy_best_knn <- cm_best_knn$overall[['Accuracy']]
    sensitivities_knn <- cm_best_knn$byClass[,"Sensitivity"]

    vect_accuracy_knn <- c(vect_accuracy_knn,accuracy_best_knn
    matrix_sensitivities_knn[i,] <- sensitivities_knn

}

proc.time()-pt # 6776 s

mean_accuracy_svm = mean(vect_accuracy_svm)
mean_sensitivities_svm = colMeans(matrix_sensitivities_svm)

mean_accuracy_knn = mean(vect_accuracy_knn)
mean_sensitivities_knn = colMeans(matrix_sensitivities_knn)

setwd("C:/Users/33650/Documents/CoursL2/MachineLearning")
```

```
table_means_sensitivities <- data.frame(mean_sensitivities_svm,mean_sensitivities_knn)
table_means_accuracies <- data.frame(mean_accuracy_svm,mean_accuracy_knn)

write.table(table_means_accuracies,"Mean_accuracies.csv",row.names=FALSE,dec=".", na=" ",append=FALSE)
write.table(table_means_sensitivities,"Mean_sensitivities.csv",row.names=FALSE,dec=".", na=" ",append=FALSE)
```

Annexe 4 – Code R comparaison SVM vs KNN (bi-classes)

```
library(class)
library(kernlab)
library(caret)

################ choix nombre d'échantillonnages #################

n=50

set.seed(100)

################# Initialisation vecteurs ###################

vect_accuracy_svm <- NULL
vect_sensitivity_svm <- NULL
vect_specificity_svm <- NULL

vect_accuracy_knn <- NULL
vect_sensitivity_knn <- NULL
vect_specificity_knn <- NULL

pt<-proc.time()
for(i in 1:n){

  ############ Initialisation paramètres ############
```

```
napp=1000
ch1=4
ch2=9

############ Echantillons d'apprentissage et de test ###############

train.1SVM=train[sample(which(train$y==ch1 | train$y==ch2) ,napp),]
test.1SVM=test[which(test$y==ch1 | test$y==ch2 ),]
test.1SVM$y = as.factor(ifelse(test.1SVM$y == 4,0,1))
train.1SVM$y = as.factor(ifelse(train.1SVM$y == 4 , 0, 1))



##### enlever les colonnes nulles ########
#ind=which(colMeans(train.1SVM)==0)
#train.1SVM=train.1SVM[,-ind]
#test.1SVM=test.1SVM[,-ind]

###### Entrainement du modèle svm ###########

model_binary_svm =ksvm(train.1SVM$y ~ .,data = train.1SVM, kernel="rbfdot", C=1, kpar="automatic",scaled=F)

######## Prédiction svm #############

eval_binary_svm = predict(model_binary_svm,newdata = test.1SVM)



vect_accuracy_svm <- c(vect_accuracy_svm,confusionMatrix(eval_binary_svm,test.1SVM$y)$overall[1])
vect_sensitivity_svm <- c(vect_sensitivity_svm,confusionMatrix(eval_binary_svm,test.1SVM$y)$byClass[1])
vect_specificity_svm <- c(vect_specificity_svm,confusionMatrix(eval_binary_svm,test.1SVM$y)$byClass[2])

###### Entrainement du modèle knn ###########
Accuracy_knn = 0
Sensitivity_knn = 0
Specificity_knn = 0

# Optimisation du k
```

```
  for(j in 1:15){
    model_binary_knn <- knn(train = train.1SVM[,-785], test = test.1SVM[,-785],cl = train.1SVM$y, k = j)


    cm_knn <- confusionMatrix(model_binary_knn, test.1SVM$y)


    if(cm_knn$overall[['Accuracy']] > Accuracy_knn){
      Accuracy_knn = cm_knn$overall[['Accuracy']]
      Sensitivity_knn = cm_knn$byClass[1]
      Specificity_knn = cm_knn$byClass[2]
      best_k = j
    }
  }
  cat("Best accuracy model reached for k = ",best_k,"\n", "with accuracy = ",Accuracy_knn,sep=" ","\n")

  vect_accuracy_knn <- c(vect_accuracy_knn,Accuracy_knn)
  vect_sensitivity_knn <- c(vect_sensitivity_knn,Sensitivity_knn)
  vect_specificity_knn <- c(vect_specificity_knn,Specificity_knn)

}

proc.time()-pt

mean_accuracy_svm = mean(vect_accuracy_svm)
mean_sensitivity_svm = mean(vect_sensitivity_svm)
mean_specificity_svm = mean(vect_specificity_svm)

mean_accuracy_knn = mean(vect_accuracy_knn)
mean_sensitivity_knn = mean(vect_sensitivity_knn)
mean_specificity_knn = mean(vect_specificity_knn)



table_means_svm = data.frame(mean_accuracy_svm,mean_sensitivity_svm,mean_specificity_svm)
```

```
table_means_knn = data.frame(mean_accuracy_knn,mean_sensitivity_knn,mean_specificity_knn)

setwd("C:/Users/33650/Documents/CoursL2/MachineLearning")
write.table(table_means_svm,"Means_binary_svm.csv",row.names=FALSE,dec=".", na=" ",append=FALSE)
write.table(table_means_knn,"Means_binary_knn.csv",row.names=FALSE,dec=".", na=" ",append=FALSE)
```

Annexe 5 – Code R One class classification (SVM gaussien)

```
library(kernlab)
library(caret)

################ choix nombre d'échantillonnages #################

n=50

set.seed(100)

################# Initialisation vecteurs ###################

vect_accuracy <- NULL
vect_sensitivity <- NULL
vect_specificity <- NULL


pt<-proc.time()
for(i in 1:n){

############ Initialisation paramètres #############

napp=1000
norm=9
out=4
```

```
############ Echantillons d'apprentissage et de test ###############

#out_app <- train[sample(which(train$y==out),50),-785]
train.1SVM=train[sample(which(train$y==norm),napp),-785]
test.1SVM=test[which(test$y==norm | test$y==out),]
truth=ifelse(test.1SVM$y==norm,"normal","abnormal")
test.1SVM=test.1SVM[,-785]

##### enlever les colonnes nulles ########
#ind=which(colMeans(train.1SVM)==0)
#train.1SVM=train.1SVM[,-ind]
#test.1SVM=test.1SVM[,-ind]

###### Entrainement du modèle ##########

model_OCC=ksvm(as.matrix(train.1SVM),type = "one-svc",kernel="rbfdot",nu=0.9, kpar="automatic",scaled=F)

######## Prédiction #############

pred=ifelse(predict(model_OCC,test.1SVM)==TRUE,"normal","abnormal")
t=table(pred,truth)

vect_accuracy <- c(vect_accuracy,confusionMatrix(t,positive = "normal")$overall[1])
vect_sensitivity <- c(vect_sensitivity,confusionMatrix(t,positive = "normal")$byClass[1])
vect_specificity <- c(vect_specificity,confusionMatrix(t,positive = "normal")$byClass[2])

}

proc.time()-pt

mean_accuracy = mean(vect_accuracy)
mean_sensitivity = mean(vect_sensitivity)
mean_specificity = mean(vect_specificity)

table_means = data.frame(mean_accuracy,mean_sensitivity,mean_specificity)
```

```
setwd("C:/Users/33650/Documents/CoursL2/MachineLearning")
write.table(table_means,"Means_OCC.csv",row.names=FALSE,dec=".", na=" ",append=FALSE)
```