HEALTH DATA CHALLENGE

Nom du Challenge : HADACA - Equipe Cancer
Groupe : 4
Membres :
Sonia Fettat <sonia.fettat@u-psud.fr>, Sara Mekedem <sara.mekedem@u-psud.fr>,
Johann Equilbec <johann.equilbec@u-psud.fr>, Jean-Baptiste Bourgeois<jean-baptiste.bourgeois@u-psud.fr>, Arezki Djaroud <arezki.djaroud@u-psud.fr> & Elias Brun <elias.brun@u-psud.fr>
Vidéo: https://www.youtube.com/watch?v=1ioR6sPtct0
Diapos : https://drive.google.com/file/d/1_Z2VBYucWmAvEvgL8Em9OpEQsVVW46an/view?usp=sharing
Challenge URL : https://codalab.lri.fr/competitions/333
Github repository : https://github.com/Cancer-MP/Cancer
Dernière soumission de code : 8351

Introduction

Le cancer est l'une des maladies les plus répandues au monde. Seulement en 2017, 400 000 nouveaux types de cancer ont été découverts. Malheureusement,même à l'ère des technologies, il n'y a toujours pas de remède absolu. Nous avons choisi le projet HADACA car c'est un moyen d'aider la recherche à avancer et lutter contre le cancer .
Ce challenge consiste à atteindre une bonne classification dans un certain contexte. Le but de celui-ci est d'effectuer une tâche de classification multiclasse sur les données, plus précisément, apprendre à une machine à prédire la phase d'avancement du cancer.
Pour ce faire, nous disposons de 5000 prélèvements d'ADN effectués sur 5000 patients diagnostiqués à différents stades du cancer. Nous avons en tout 10 classes à prédire dans notre ensemble de données.
Notre tâche consiste à améliorer les résultats de la classification en fonction des étapes de ces patients, le résultat final de ce projet est l'identification de 10 stades de cancer différents parmis une population spécifique.
Notre projet utilise une métrique de précision, comme indiqué dans la partie Résultats du ReadMe.
Les méthodes de classification et d'exploration de données constituent un moyen efficace de classer les données. Surtout dans le domaine médical, où ces méthodes sont largement utilisées en diagnostic et en analyse pour prendre des décisions.
Pour réaliser ce challenge nous avons séparé le travail à réaliser en 3 parties : preprocessing, prédiction et visualisation.
Il peut utiliser différents types d'algorithmes d'apprentissage telle que **les réseaux de neurones** , ils peuvent être utilisés pour transformer les entités afin de former des limites de décision non linéaire assez complexes.
Pour calculer ces fonctions complexe il faut utiliser **un neuron formel qui permet de calculer la somme pondérée des entrées** (poids synaptique) puis appliquer à cette valeur une fonction d'activation, généralement non linéaire , on obtient une valeur filan qui est la sortie du neuron .
Nous avons vu aussi **une règle de Hebb** qui est une règle d'apprentissage des réseaux des neurones dans le contexte de l'étude d'assemblées de neurones , elle est basé sur la plasticité synaptique, la force communication entre deux neurones peut changer,être renforcer ou diminuer .

Preprocessing:

1. Description:

La partie preprocessing consiste à mettre en forme les données pour permettre par la suite une classification de celle-ci. C'est une partie importante car suivant la forme des données l'ordinateur va plus ou moins bien apprendre à reconnaître les données par la suite et donc à les classer.

Pour la mise en forme des données nous avons choisi de jouer sur la variance de celles-ci et sur la factorisation des matrices de données.

Pour cette partie nous avons implémenté trois fonctions pour le Preprocessing:

- La première est la méthode *init()* qui initialise toutes les données que nous allons utiliser par la suite.
- La seconde est la méthode fit() qui entraîne le modèle avec les paramètres initialisés dans la fonction init.
- La troisième est la méthode *transform()* transforme la matrice X donnée en paramètre en supprimant les données dont la variance est trop faible puis la taille de la matrice est diminué en perdant un minimum d'informations.

2. Difficultés et solutions:

Sur la première version de code nous utilisons deux méthodes pour réduire le nombre de features par données et on s'est rendu compte grâce à la cross-validation que nous avions un taux de réussite de seulement 17%.

Nous avons aussi eu des problèmes avec l'utilisation des pipelines. Nous avons tenté d'utiliser un pipeline pour rechercher les meilleurs paramètres à donner au PCA() et au TruncatedSVD() (les deux fonctions de réductions de features/dimension mentionnées plus haut). Nous avons été confronté à une erreur que nous ne comprenions pas et que nous n'avons pas su résoudre lors de la recherche des meilleurs paramètres à fournir.

Sur la seconde version nous utilisons la méthode PCA() pour réduire la taille de la matrice et nous obtenons un taux de réussite de 99%.

Sur la troisième version nous faisons la même chose que sur la seconde version de code mais cette fois-ci nous avons utiliser une autre méthode de réduction de dimension, SVD().

Pour résoudre ces problèmes nous avons fait cela simplement:

a) N'ayant pas compris le fonctionnement d'un pipeline nous avons tout simplement supprimer le pipeline et nous avons fait la recherche de paramètre à la main.
b) Nous avons refait un code en partant de zéro mais sur la même base que le précédent, c'est-à-dire diminuer le nombre de données et de features par données à traiter. Cette fois-ci cependant nous avons décidé de n'utiliser qu'une seule fonction de factorisation de matrice PCA() pour la seconde version de code et SVD() pour la troisième version de code..

3. Présentation des fonctions utilisées:

Pour la définition du *fit()* et du *transform()* de la classe Preprocessing, nous avons utilisé deux fonctions, bien distinctes :

- PCA() [1] : C'est une fonction appartenant à la bibliothèque sklearn.décomposition, qui réduit le nombre de features en décomposant les données en valeur singulière pour les projeter dans un espace de dimension inférieur, lors de l'initialisation de la fonction. Elle groupe les données en "tendances" principales, ainsi, les données les moins significatives sont rassemblées vers d'autres données plus importantes. Cela permet de garder nos données significatives, tout en réduisant le nombre de features. Ex: "self.transform.pca = PCA(n_components) où n_components correspond au nombre de features que l'on veut obtenir après réduction.

- La fonction TruncatedSVD() [2] se déclare de la même façon que PCA() mais cette fois-ci elle fait la réduction de dimension en tronquant la valeur singulière de chaque donnée. Elle ne centralise pas les données avant de calculer la décomposition des valeurs singulières, contrairement au PCA(). Toute matrice X peut être décomposée sous la forme X = USV tel que U et V sont des matrices unitaires et S est une matrice diagonale : les "valeurs singulières" formant la diagonale. Ces valeurs singulières sont les données qui vont être utilisées ensuite. C'est le principe de la Single Value Decomposition : SVD().

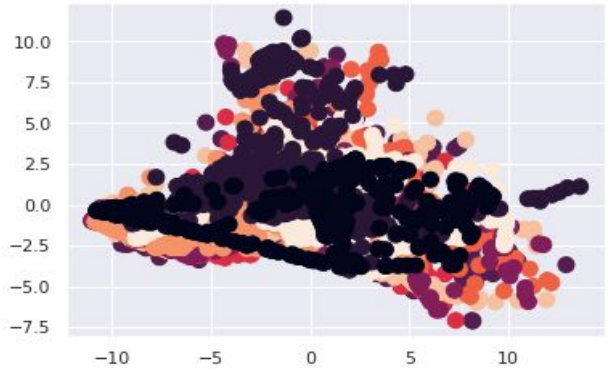

Figure 1 : Projection en 2D de nos données en appliquant la méthode PCA()

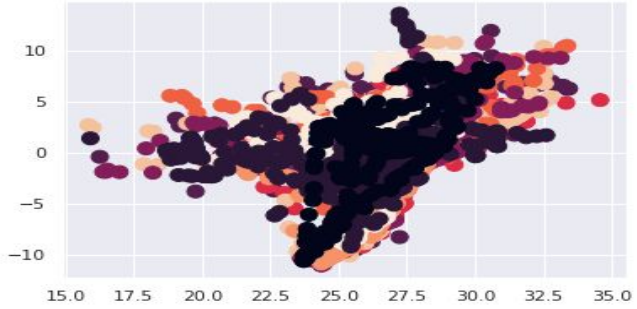

Figure 2: Projection de nos données en 2D en appliquant la méthode SVD()

Prédiction:

Notre but dans cette partie était d'établir un classifieur qui puisse correctement disposer nos données dans la bonne classe, en l'occurrence chacune correspondant à un stade d'avancement du cancer. Il fallait donc récupérer les données traitées par le préprocesseur pour enfin établir une prédiction pertinente. Nous avons ainsi comparé plusieurs algorithmes appliqués à des données d'entraînement, puis par cross-validation (un modèle d'entraînement fondé sur une technique d'échantillonnage) il a fallu déterminer laquelle des méthodes testées étaient plus fiables, plus précises.

D'après les tests que nous avons appliqué au set de données, nous avons choisi l'algorithme **Linear regression or SVM** car car comme nous pouvons observer dans la table ci-dessous, les meilleurs résultats ont été obtenu par celui-ci, avec un net avantage sur les autres méthodes.

Method	**NaiveBayes or Gaussian classifier**	**Linear regression or SVM**	**Decision Tree**	**Random Forest**	**SGD**
Training	0.3860	0.8991	1.0000	0.5125	0.3045
Cross Validation	0.3589	0.8776	0.8292	0.4424	0.3026

Tableau 1: ***Résultats préliminaires de la cross-validation.***

La méthode Decision Tree engendrait de bons résultats également mais cela variait en fonction du DataSet et nous cherchions une méthode plus constante. Ceci est dû au fait que Decision Tree va créer branche après branche jusqu'à ce que toutes les données soient exclusivement placées dans leurs classes. Nous avons beau eu changé les paramètres le score était

trop aléatoire nous nous sommes donc dirigés vers la méthode SVC qui en plus de la précision, nous était favorables par sa facilité à traiter une grande quantité de données. Au vu de la nette différence entre les résultats par cette méthode et les autres, nous avons jugé inutile de tester les autres méthodes en modifiant les paramètres (excepté Decision Tree) et avons jugé plus pertinent de nous concentrer uniquement sur la méthode SVC en modifiant ses paramètres pour obtenir le meilleur résultat possible. Pour commencer, malgré les résultats très précis de cette méthode on a remarqué qu'à chaque fois qu'on appliquait cette méthode sur d'autres données les résultats devenaient sensiblement moins précis. Cela étant dû au fait que nos données précédentes étaient connues par la méthode SVC (données entraînées). Nous avons donc refait plusieurs tests en appliquant d'autres valeurs à nos données (on a retenue la valeur de 52 "n_components" = 52) ainsi qu'aux paramètres de la méthode(nous avons fait varier le terme d'erreur "C" de 1 à 1000) et c'est comme cela que nous avons augmenté notre score de prédiction.

Pour cette partie on c'est occupé de la fonction predict (def predict(self x)) Cette fonction fournit des prédictions d'étiquettes sur des données (de test).

Plus précisément cette fonction prend les données échantillonnées par le preprocessing (échantillon d'apprentissage) et nous renvoie des prédictions sur les données, en clair elle donne une prédiction sur le stade d'avancement du cancer d'un patient et enfin on classe ses données selon les prédiction que nous avons eu.

Résultats:

Tout d'abord pour juger de l'efficacité de notre preprocessing et notre classifieur nous avons utilisé la cross-validation sur l'ensemble d'entraînement pour obtenir un score. Dans la première version du code, avec l'utilisation de deux fonctions de réduction de dimension de matrice nous avons obtenu un score très faible de 17% de réussite, alors qu'en utilisant uniquement le classifieur nous obtenions un score de 87% (cf tableaux de résultats). Puis avec la seconde version du code, avec une seule fonction de réduction de matrice (PCA()) nous avons obtenu 99% de réussite. Pour la troisième version du code, nous obtenons le même score avec la fonction SVD mais avec un n_components légèrement plus élevé(47 pour le PCA, 53 pour le SVD, cf. Figure 6).

Méthodes de preprocessing	**PCA() + TruncatedSVD()**	**PCA()**	**SVD()**
Cross validation	0.17 +/- 0.03 → 17% +/- 3%	0.99 +/- 0.03 → 99% +/- 3%	0.99+/-0.03→ 99% +/- 3%

Tableau 2: ***Résultats de la cross validation sur l'ensemble d'entraînement.***

Nous avons essayé de faire varier les valeurs de n_components dans les deux fonctions de réduction de dimension de matrice (PCA et SVD) .Pour le SVD on obtient un score maximal de 99% +/- 3% à la cross-validation pour un n_components supérieur ou égal à 52. Pour la fonction PCA le score maximal à la cross-validation est de 99% +/- 3% lorsque n_components est supérieur ou égal à 52 également. On peut donc se servir autant d'une fonction que de l'autre pour le preprocessing. Nous avons choisi d'utiliser la fonction PCA() (cf. Figure 3).

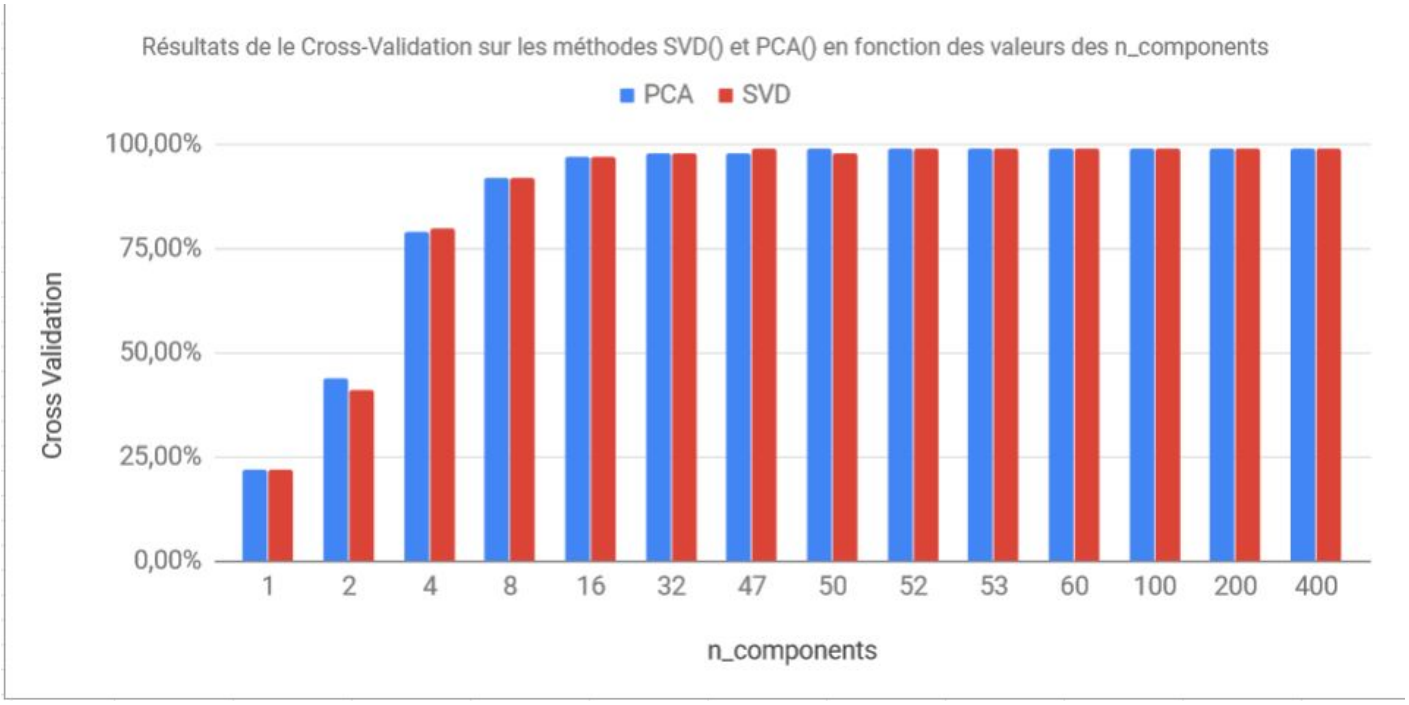

Figure 3: Graphe de colonne qui représente les résultats de la cross-validation sur les méthodes SVD() et PCA() en fonction des valeurs de n_components

Evolution score prediction	**Implémentation SVC (paramètres par défaut)**	**Mise à jour des paramètres**	**Avec preprocessing**
Cross validation	89% +/- 3%	97% +/- 2%	99% +/- 3%
Soumission Codalab	87,7%	95,27%	99,62%

<u>Tableau 4:</u> Résultats de la cross validation sur nos score sur Codalab avant et après réduction de la matrice SVD() avec le preprocessing.

Après les tests des différentes méthodes et avoir opté pour les SVC, nous avons modifié le terme d'erreur afin d'obtenir une meilleur classification des données puis en appliquant cette méthode aux données traitées par le préprocesseur nous avons obtenu ce score de 0,9962.

RESULTS

#	User	Entries	Date of Last Entry	Prediction score ▲	Duration ▲	Detailed Results
1	Cancer	25	03/31/19	0.9962 (1)	0.00 (1)	View
2	luc.gibaud	4	01/24/19	0.9843 (2)	0.00 (1)	View
3	Malikkazi	3	01/24/19	0.9804 (3)	0.00 (1)	View
4	takfarinas.nait-larbi	51	03/22/19	0.9790 (4)	0.00 (1)	View
5	doctor	25	03/22/19	0.9790 (4)	0.00 (1)	View

<u>Figure 4:</u> figure représentant notre classement Codalab après application de la méthode SVC et des données traites par le preprocessing.

Ci-dessus la soumission Codalab de notre code. Nous avons donc jusqu'au 1er avril 2019 le meilleur score avec un taux de 99,62%.

<u>Visualisation:</u>

La représentation visuelle des données est le moyen le plus efficace de transmettre des informations significatives,elle consiste à transformer des données complexes en représentations visuelles simples, pour faciliter l'exploitation et la compréhension des données.

Dans notre projet les données sont une matrice de (nombre de patients)*(nombre de caractéristiques) les caractéristiques de l'ADN choisies correspondent aux composants 1 et 2. Comme il ya beaucoup de données , elle sont divisées en plusieurs parties appelées "composant" , nous allons concentrer nos travaux sur les "composant 1 " et "composant 2", nous avons choisi de représenter le nombre de patient en fonction du nombre de quantité de chaque composant sous forme d'un histogramme ,c'est un moyen rapide pour étudier la répartition de nos données.

(6)

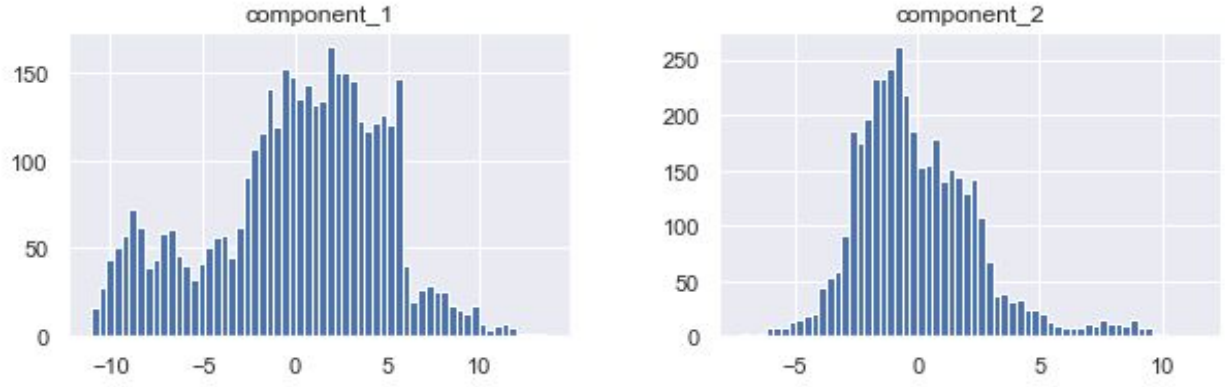

Figure 5 : Histogramme représentant le nombre de patient en fonction de chaque composant

Nous avons également utiliser une heatmap pour représenter nos données de façon à afficher une vu plus générale des données , car il est difficile de faire la distinction entre les nuances de couleur et d'extraire des points de données spécifiques , on peut aussi l'utiliser pour afficher les modifications de nos données au fil du temps.

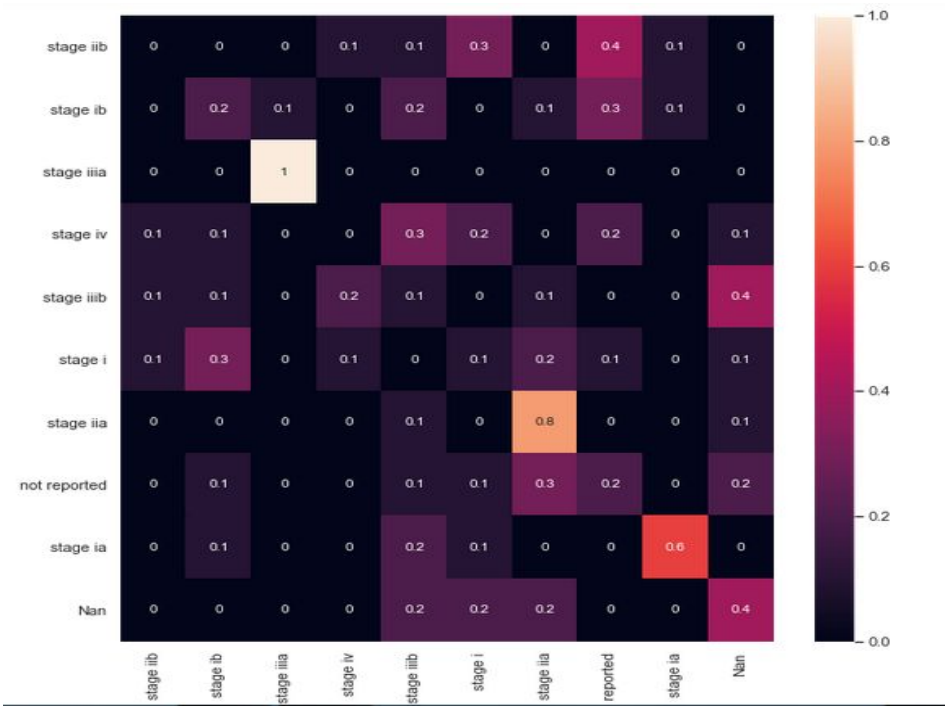

Figure 6: Matrice de confusion représentant nos données.

Nous avons aussi visualiser les différents résultats de test par cross-validation des différentes méthodes de classification de la partie prédiction sous forme de bargraphe, pour éventuellement voir la méthode la plus adaptée à notre challenge

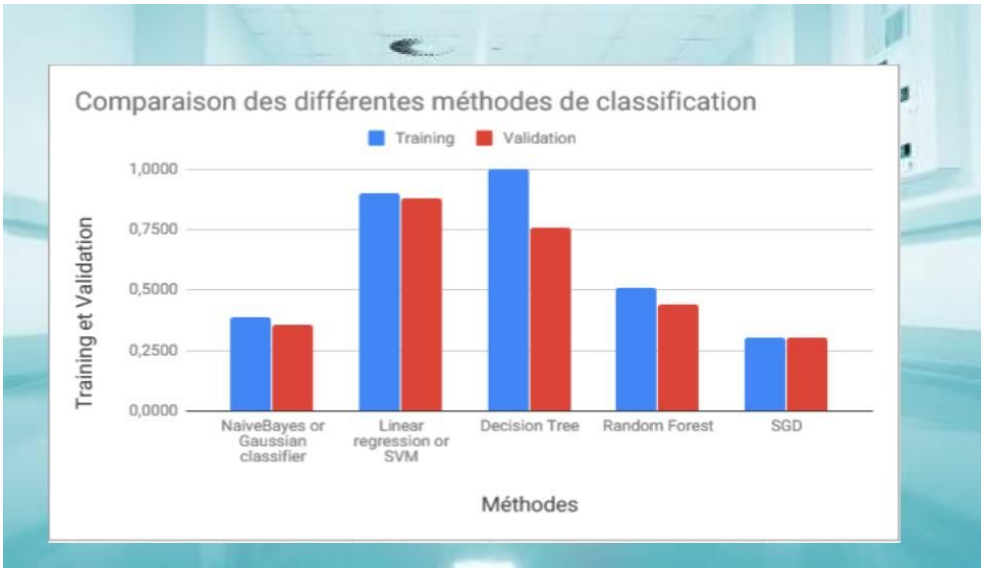

Figure 7 : Comparaison des différentes méthodes de classification

Et de façon plus générale , nous avons choisi de représenter sous forme d'un graphe de colonne le tableau de résultats pour le preprocessing ,à fin de bien visualiser et comparer les résultats avec différentes méthodes . , donc on a comparé entre trois méthodes qui sont PCA() et TruncatedSVD() ensemble puis les deux méthodes séparément .On conclut que avec la méthode SVD() on obtient un score maximal de 99% +/- 3% à la cross-validation et avec une seule fonction de réduction de matrice PCA() nous avons obtenu 99% de réussite.

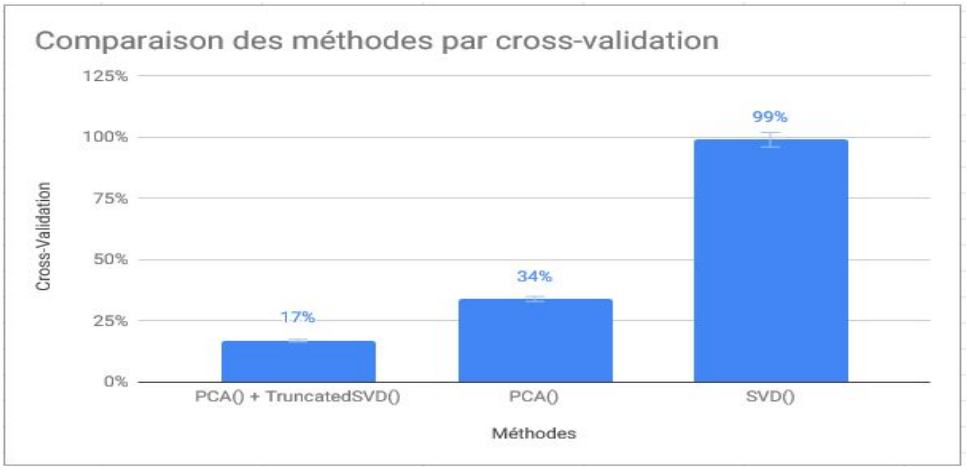

Figure 8: Graphe de colonne qui représente les résultats des méthodes

IV/ Discussion et conclusion :

C'est la première fois que nous travaillons en groupe sur un projet informatique relié à la médecine avec un but bien défini , pour ce projet nous avons utilisé un apprentissage supervisé qui répond aux problématiques de classification de données, et surtout aide amplement à la détermination du stade du cancer développé par un patient.

Ce système de diagnostic est particulièrement utile pour pouvoir générer un traitement en fonction de l'évolution de cette maladie. C'est pourquoi nous avons vraiment pris du temps pour pouvoir surpasser nos attentes, aller plus loin que ce que nous savions faire, aller chercher les connaissances nécessaires pour réaliser un classifieur qui puisse vraiment aider à l'évolution de ce projet dont l'objectif est tellement admirable.

Forcément quand nous nous sommes engagés dans ce projet, nous avions pleins d'ambitions, mais plus nous avancions plus nous rencontrions des problèmes, ce qui a ralenti considérablement notre productivité. Au final nous n'avons pas pu faire tous les tests sur les paramètres d'autres classifieurs (Random Forest), pour nous concentrer sur celui où il n'y avait pas autant de choses à changer et ainsi faire un classifieur à la hauteur de ce que nous espérions.

Au niveau de la gestion du projet en équipe, nous avons réussi à bien nous répartir les tâches à l'aide de notre chef de groupe, chacun a bien fait ce qui lui était demandé et le travail a toujours été équitablement réparti, ainsi, en tant qu'équipe, nous avons pu atteindre nos objectifs dans les temps et l'ambiance générale du groupe était bonne.
Une expérience à renouveler (ou pas) !

"98% de la santé, aujourd'hui, c'est du curatif. L'intelligence artificielle permettra de basculer sur une médecine plus préventive." Laurent Schlosser, Microsoft.

Message aux étudiants de l'an prochain : "Choisissez un challenge qui vous passionne pour y voir l'application de l'informatique et surtout de l'intelligence artificielle ainsi que son utilité dans ce même domaine. Ce sera plus satisfaisant de travailler dans un domaine qui vous intéresse. Bon courage!"

V/ Bibliographie

(1) Scikit-Learn, scikit-learn developers (BSD License), 2007 - 2018, **Principal component analysis**. Repéré à: https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html

(2) Scikit-Learn, scikit-learn developers (BSD License), 2007 - 2018, **TruncatedSVD**, Repéré à: https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.TruncatedSVD.html

(3) Seaborn, Michael Waskom, Copyright 2012-2018, **Heatmap** Repéré à: https://seaborn.pydata.org/generated/seaborn.heatmap.html

(4) Scikit-Learn, scikit-learn developers (BSD License), 2007 - 2018, **SVC** Repéré à: https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

(5) Scikit-Learn, scikit-learn developers (BSD License), 2007 - 2018, **Pipeline**. Repéré à: https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html

Pour faire nos figures:

(6) The python graph gallery, Copyright © 2017 The python graph gallery, 2017, **HISTOGRAM.** Repéré à: https://python-graph-gallery.com/histogram/

(7) The python graph gallery, Copyright © 2017 The python graph gallery, 2017, **SCATTER PLOT.** Repéré à: https://python-graph-gallery.com/scatter-plot/

(8) The python graph gallery, Copyright © 2017 The python graph gallery, 2017, **CONNECTED SCATTER PLOT.** Repéré à: https://python-graph-gallery.com/connected-scatter-plot/

BONUS

Définition de "Cross-Validation" :

/*La **validation croisée** (« *cross-validation* ») est en apprentissage automatique (statistique), une méthode d'estimation de fiabilité d'un modèle fondé sur une technique d'échantillonnage.

Dans ce projet on a utilisé cette méthode dans notre read me pour pouvoir estimer la fiabilité de nos différentes méthodes d'apprentissage "SVC, PCA ..." et choisir la plus efficace, et aussi choisir la meilleur valeur pour nos composantes "n_components".*/

Mieux : La validation croisée est, en apprentissage automatique, une méthode d'estimation de fiabilité d'un modèle fondé sur une technique échantillonnage, En fait, il y a au moins trois techniques de validation croisée : « testset validation » ou « holdout method », « k-fold cross-validation » et « leave-one-out cross validation »,

Supposons posséder un modèle statistique avec un ou plusieurs paramètres inconnus, et un ensemble de données d'apprentissage sur lequel on peut entraîner le modèle. Le processus d'apprentissage optimise les paramètres du modèle afin que celui-ci corresponde aux données le mieux possible. Si on prend ensuite un échantillon de validation indépendant issu de la même population d'entraînement, il s'avérera en général que le modèle ne réagit pas aussi bien à la validation que durant l'entraînement. La validation croisée est un moyen de prédire l'efficacité d'un modèle sur un ensemble de validation hypothétique lorsqu'un ensemble de validation indépendant et explicite n'est pas disponible.

Echantillonnage:

L'échantillonnage est une technique de sélection d'une partie des données de notre dataset (les plus utiles) qui produit une série d'échantillon à étudier.

Qu'est-ce que le sur-apprentissage ?

/*Le sur-apprentissage, appelé "overfitting" en anglais, est un terme utilisé lorsqu'un modèle s'est "trop bien entraîné". En effet, le fait de trop entraîner son modèle en lui donnant beaucoup trop de données c'est qu'il va avoir tendance a etre tres fort sur les données sur lesquelles il s'est entraîné (avec des scores de 100%) mais inefficace sur de nouvelles données sur lesquelles il ne s'est jamais entraîné (avec des scores quasi nuls 0.001%). */

Mieux : Un réseau de neurones apprend grâce à des exemples (jeux d'entraînements) qui lui sont soumis. Le but de cet apprentissage est de permettre au réseau de tirer de ces exemples des généralités et de pouvoir les appliquer à de nouvelles données par la suite.
En intelligence artificielle, on parle de surapprentissage ou de overfitting quand un modèle a trop appris les particularités de chacun des exemples fournis en exemple. Il présente alors un taux de succès très important sur les données d'entraînement (pouvant atteindre jusqu'à 100%), au détriment de ses performances générales réelles.

La métrique utilisée par notre challenge :

The submissions are evaluated using the **macro-average precision metric**. Micro- and macro-averages (for whatever metric) will compute slightly different things, and thus their interpretation differs. A macro-average will compute the metric independently for each class and then take the average (hence treating all classes equally), whereas a micro-average will aggregate the contributions of all classes to compute the average metric.

Code de la métrique :

```
import numpy as np
import scipy as sp
from functools import reduce

def binarize_predictions(array, task='binary.classification'):
    bin_array = np.zeros(array.shape)
    if (task != 'multiclass.classification') or (array.shape[1] == 1):
        bin_array[array >= 0.5] = 1
    else:
        sample_num = array.shape[0]
        for i in range(sample_num):
            j = np.argmax(array[i, :])
            bin_array[i, j] = 1
    return bin_array

def acc_stat(solution, prediction):
    TN = sum(np.multiply((1 - solution), (1 - prediction)))
    FN = sum(np.multiply(solution, (1 - prediction)))
    TP = sum(np.multiply(solution, prediction))
    FP = sum(np.multiply((1 - solution), prediction))
    return (TN, FP, TP, FN)

def mvmean(R, axis=0):
    if len(R.shape) == 0: return R
    average = lambda x: reduce(lambda i, j: (0, (j[0] / (j[0] + 1.)) * i[1] +
                                             (1. / (j[0] + 1)) * j[1]), enumerate(x))[1]
    R = np.array(R)
    if len(R.shape) == 1: return average(R)
    if axis == 1:
        return np.array(map(average, R))
    else:
        return np.array(map(average, R.transpose()))
```

```
def bac_metric(solution, prediction, task='binary.classification'):
    label_num = solution.shape[1]
    score = np.zeros(label_num)
    bin_prediction = binarize_predictions(prediction, task)
    [tn, fp, tp, fn] = acc_stat(solution, bin_prediction)
    eps = 1e-15
    tp = sp.maximum(eps, tp)
    pos_num = sp.maximum(eps, tp + fn)
    tpr = tp / pos_num  # true positive rate (sensitivity)
    if (task != 'multiclass.classification') or (label_num == 1):
        tn = sp.maximum(eps, tn)
        neg_num = sp.maximum(eps, tn + fp)
        tnr = tn / neg_num  # true negative rate (specificity)
        bac = 0.5 * (tpr + tnr)
        base_bac = 0.5  # random predictions for binary case
    else:
        bac = tpr
        base_bac = 1. / label_num  # random predictions for multiclass case
    bac = mvmean(bac)
    score = (bac - base_bac) / sp.maximum(eps, (1 - base_bac))
    return score

def bac_multiclass(solution, prediction):
    prediction = prediction.reshape((prediction.shape[0],1))
    return bac_metric(solution, prediction, task='multiclass.classification')

from sklearn import metrics
def precision_score(solution, prediction):
    prediction = prediction.reshape((prediction.shape[0],1))
    return metrics.precision_score(solution, prediction, average='macro')
```