RAPPORT SUR L'ANALYSE DES DONNÉES D'INDIVIDUS ATTEINTS DE CANCER

Awa Syr DIAGNE

&

Lama ZIDAN

Responsable :

Julien Ah-Pine

OCTOBRE 2022

Table des matières

RAPPORT SUR L'ANALYSE DES DONNÉES D'INDIVIDUS ATTEINTS DE CANCER

I. Présentation du contexte et des enjeux de la base de données et de la ou des tâches retenues .

Nous disposons d'un jeu de données qui représente toutes les données génomiques de patients atteints de cancer avec données de survie associées Ces données sont issues du "The Cancer Genome Atlas" (TCGA).

Nous avons 408 observations et 719 variables qui sont 715 gènes exprimées en nombres continus très variés. Nous avons une variable "CLINIC.OS STATUS" qui est qualitative binaire (0 : LIVING, 1 : DECEASED), il nous informe de l'état actuel du patient, s'il est décédé ou vivant et qui sera ici notre valeur cible (y à prédire).
Trois autres variables moins intéressantes pour nous, sont la durée de survie en mois "CLINIC.OS MONTHS".
Et "Row.names" qui indique le code du patient concerné.
Et le type d'étude de cancer effectué "CLINIC.study".
Il est important d'indiquer que les études dans notre base sont "luad" et "lusc" donc il s'agit d'un seul type de Cancer qui est le Lung Cancer.

Out[24]:

	Row.names	CLINIC.OS_MONTHS	CLINIC.OS_STATUS	CLINIC.study	EXP.A1CF	EXP.ABI1	EXP.ABL1	EXP.ABL2	EXP.ACKR3	EXP.ACSL3	...	EXP.ZFHX
1	TCGA.05.4249.01	38.045974	0:LIVING	luad	0.3220	2116.0026	2180.2909	1272.6949	116.2264	1660.0089	...	1461.963
2	TCGA.05.4382.01	19.910069	0:LIVING	luad	0.0000	2389.6601	2076.4421	975.4037	733.2279	2512.9329	...	1141.667
3	TCGA.05.4384.01	13.996187	0:LIVING	luad	0.0000	1534.5144	2015.8827	1032.9872	310.9346	2422.7245	...	1532.510
4	TCGA.05.4389.01	45.007726	0:LIVING	luad	36.2416	2100.5168	1302.3490	733.5570	211.0738	1677.8523	...	461.691
5	TCGA.05.4390.01	36.994617	0:LIVING	luad	0.0000	1182.1811	1359.7318	709.5641	297.6209	2526.5847	...	552.329

5 rows × 719 columns

Nous disposons d'une taille très importante de nos données. En effet, elles sont de très grandes dimensions, elles comptent 715 gènes.
L'objectif de ce projet est donc d'effectuer une réduction de dimension grâce aux différents modèles de Machine Learning pour voir si cela est utile pour résoudre le problème de haute dimension des bases de données et s'il permet de conserver que les caractéristiques discriminantes pour éviter ainsi le surapprentissage des modèles.
Notre objectif est d'analyser notre base de données pour en tirer des connaissances. C'est-à-dire savoir si les observations de notre base (les patients) auront des caractéristiques similaires en termes d'expression des leurs gènes.
Savoir aussi si les variables (les gènes) sont corrélées entre elles.
Essayer de représenter ces informations dans une espace des dimensions réduites pour mieux les visualiser et expliquer le comportement particulier de plusieurs variables.

Dans ce rapport, il sera question de tester tout d'abord le modèle SVM avec les variables initiales de notre base sans réduction de l'espace des gènes pour prédire la variable cible catégorielle "CLINIC.OS STATUS".

Ensuite, tester plusieurs méthodes de réduction de l'espace des gènes, afin de prédire notre variable cible. Et enfin conclure en comparant les résultats de prédiction sans réduction et avec réduction de dimension afin de mieux voir si cela apporte quelque chose ou pas à notre prédiction .

II. Présentation succincte des méthodes de réduction de dimension mise en œuvre

- ACP : Analyse en Composantes Principales

C'est une méthode de réduction de dimensionnalité qui traite des tables de données dont les variables sont quantitatives continues.

L'ACP permet de déterminer des nouvelles variables qui synthétisent les variables initiales en en conservant au mieux l'inertie (la variance).

Ces nouvelles variables sont des vecteurs appelées composantes principales. Les composantes principales sont les coordonnées des points du nuage lorsqu'elles sont projetées sur des vecteurs appelés axes factoriels ou principaux.

Une composante principale est une combinaison linéaire des variables initiales.
Les coefficients de ces combinaisons linéaires sont les coefficients de corrélation entre la composante principale et les différentes variables initiales.
L'étude de ces coefficients permet d'associer à toute composante principale un groupe de variables initiales qui lui est associé.

Cette méthode pourrait être appliquée soit en diagonalisant la matrice du nuage des individus ou bien la matrice du nuage des variables (les gènes).
Il n'est pas nécessaire de faire deux diagonalisations mais une seule (grâce aux liens de dualité entre l'analyse de l'espace des individus et celle de l'espace des variables).

Avant de commencer, nous devons centrer et réduire notre base puis diviser par $\sqrt{n}$.
En pratique, la réduction permet aux variables de s'affranchir de leurs unités de mesure ce qui rend l'analyse plus robuste face aux biais associés aux différences d'échelle.

Si l'on décide d'appliquer l'ACP sur les individus, on utilise la distance Euclidienne entre les individus. Et les axes principaux peuvent être obtenues en diagonalisant la matrice des produits scalaires entre individus $K=XX^T$.
et les composantes principales que l'on cherche à trouver seront obtenues en multipliant la matrice X^T avec ces axes principaux.

Alors que, si l'on décide d'appliquer l'ACP sur les variables, on utilise le coefficient de corrélation entre les variables. Et les axes principaux peuvent être obtenus en diagonalisant la matrice de données XTX.
et les composantes principales que l'on cherche à trouver seront obtenues en multipliant la matrice X avec ces axes principaux.

Il est important de préciser que les inerties des deux nuages sont identiques. En effet, le choix du nombre de composantes principales doit être fait avec soin pour ne pas manquer certaines informations liées à la liste originale de caractéristiques ainsi que conserver la variance maximale possible.

Mais comment cette méthode commence à construire ces axes et sous quelles conditions ?
Tout d'abord, nous chercherons une projection sur un axe F1 à 1 dimension, mais par cette projection sur le premier axe F1, nous perdons de l'information. Pour en perdre moins, on peut ensuite chercher un second axe d'inertie principal, F2.

Mais pour ce second axe, une condition doit être vérifiée : l'axe doit être « perpendiculaire » au premier.
Une fois la direction de ce second axe trouvé, on cherchera le 3e (F3), avec la contrainte qu'il soit orthogonal à tous les précédents : au second, mais aussi au premier.
Ensuite, nous chercherons le 4e, orthogonal à tous les précédents, ainsi de suite.
En général, les deux premières variables expliquent au maximum la variance c'est pour cette raison qu'on s'intéresse en général à l'analyse de ces deux variables F1 et F2.

Maintenant que nous avons construit notre nouvelle base avec dimension réduite,
Comment pourrait-on détecter des variables fortement corrélées aux variables synthétiques F1, F2 ?
Le cercle de corrélation, nous permet de détecter les groupes de variables qui se ressemblent ou au contraire qui s'opposent. (Développement plus détaillé plus bas dans la partie d'analyse de l'ACP).

- **ACP à noyaux :**

Appelé également Kernel PCA, cette méthode est une extension de l'ACP. A l'aide du noyau, nous utilisons des méthodes qui nous permettent d'effectuer une ACP avec un ensemble de données séparables non linéaires.

L'ACP à noyau consiste à projeter les données par l'intermédiaire d'une application non linéaire dans un espace de dimension élevée dénommé espace des caractéristiques où l'ACP linéaire est appliqué.

Donc l'idée est de transformer ces données par une application non linéaire et qui seront ensuite utilisée pour effectuer un ACP.

Avec cette transformation, nous avons un nouvel espace de plus grande dimensi$\mathbf{C}\mathbf{v}_i = \lambda_i \mathbf{v}_i$ données transformées auront une matrice de covariance et des vecteurs propres ( )de la forme

$$\mathbf{C} = \frac{1}{N}\sum_{n=1}^{N} \phi(\mathbf{x}_n)\phi(\mathbf{x}_n)^T \qquad \mathbf{v}_i = \sum_{n=1}^{N} a_{in}\phi(\mathbf{x}_n)$$

L'astuce du noyau permet de trouver les vecteurs de coefficients représentant les vecteurs propres en solutionnant le problème de vecteurs propres. Ces vecteurs propres seront associés aux plus grandes valeurs propres obtenues.

Avec un noyau k, pour calculer les produits scalaires dans cet espace transformé, il suffit juste d'utiliser l'astuce à noyau. Ce qui nous permet de trouver les vecteurs de coefficients des vecteurs propres de notre matrice de Gram (elle représente la matrice des objets) qui s'obtient en diagonalisant $K= XX^T$.

Donc ce que nous souhaitons faire est de calculer les projections donnant les composantes principales.

- **SVM**

La SVM ou encore *Support Vector Machine* (*Machine à vecteurs de support)* est un algorithme d'apprentissage qui consiste à régler les multitudes problèmes de régression et de classification. Son avantage, c'est qu'elle est efficace dans le traitement des données de grandes dimensions.

L'utilisation de la SVM a pour but de faciliter la séparation des données. Ces données sont séparées en les délimitant c'est-à- dire à créer des frontières. Ce qui va engendrer la création de groupes. Les SVM peuvent être utilisés pour résoudre des problèmes de discrimination, c'est-à-dire décider à quel groupe appartient un échantillon, ou de régression, c'est-à-dire prédire la valeur numérique d'une variable. La résolution de ces deux problèmes passe par la construction d'une fonction h qui a un vecteur d'entrée x fait correspondre une sortie y=h(x).
Vous l'aurez compris, l'idée est donc de trouver un classificateur capable de pouvoir séparer notre jeu de données en groupe.

Cette séparation des données consiste à projeter les données sur un espace vectoriel de plus grande dimension.
On va maximiser la marge, qui représente la distance entre un point et le classificateur. Entre cette distance, aucun point ne doit être sur cet espace

Cette technique de maximisation de la marge permet, quant à elle, permet à la frontière de bien s'adapter à de nouveaux échantillons en dehors de l'ensemble de données d'apprentissage.

- **Sélection de variables (Features Selection) à l'aide d'une forêt aléatoire.**

Cette méthode de sélection de variable n'a pas été abordée dans le cours mais nous avons trouvé que c'est intéressant de l'appliquer sur nos données pour voir
Si cela apporte de bons résultats dans le modèle de prédiction SVM dans la suite.

Nous allons sélectionner les 23 variables les plus importantes et qui influencent au maximum notre variable cible.
Ensuite nous allons utiliser ces 23 variables en entrée de notre modèle SVM et voir les résultats de prédiction.

Cette méthode de sélection est performante car elle permet de sélectionner les variables de grande importance et qui ont une forte influence sur les valeurs des résultats, tandis que les variables de faible importance peuvent être omises, ce qui simplifie et accélère le processus d'ajustement et de prédiction.

Le modèle de forêt aléatoire est basé sur le concept d'Arbre de décision.
Une forêt aléatoire se compose de centaines d'arbres de décision, chaque arbre étant construit à partir d'un extrait aléatoire d'observations et de variables de l'ensemble de données. Tous les arbres ne voient pas toutes les entités et observations du jeu de données.

Choisir les principales variables en fonction de leur importance signifie qu'elles sont fortement corrélées avec la variable cible et non redondantes, ce qui améliore notre modèle.

Dans le processus de construction d'un arbre, l'impureté Gini est utilisée pour déterminer quelles variables doivent être divisées à chaque nœud, ce qui signifie que la sélection des principales variables sera basée sur l'importance de Gini (ou la diminution moyenne de l'impureté).
L'impureté Gini mesure la fréquence à laquelle un élément choisi au hasard serait mal classé.

Dans les forêts aléatoires, plus une variable diminue l'impureté, plus la variable est importante.
La diminution des impuretés de chaque variable peut être moyenné sur les arbres pour déterminer les variables les plus importantes.

Puisque nous avons 715 variables, il est important d'augmenter le nombre d'arbres afin d'avoir couvert toutes les variables et de généraliser l'importance des variables qui seront extraites de tous les arbres. Nous avons choisi le nombre d'arbres à 10000, l'inconvénient de ce choix est le temps de calcul.

III. Une présentation succincte des librairies et classes utilisées pour mettre en œuvre les méthodes précédentes (y compris les pré-traitements).

Tout au long de ce projet, nous avons eu à utiliser énormément de librairie :

- **Pandas :**

Il a permis de pouvoir manipuler la base de données et de pouvoir l'analyser. Grâce à elle, nous pouvons stocker ces données dans des DataFrame. Un dataframe est une structure de données permettant de stocker les données selon deux dimensions : lignes et colonnes

- **Scikit - Learn :**

Elle est celle qu'on a utilisé le plus. Elle nous a permis de pouvoir prédire le comportement de notre jeu de données. Elle présente plusieurs classes :

- ***from sklearn.decomposition import PCA*** : elle effectue l'ACP (Analyse en Composantes Principal) et donc elle réduit la dimension linéaire de notre jeu de donnée à l'aide de la décomposition en valeur singulière (SVD)
- ***from sklearn.model_selection import train_test_split :*** idéal pour nos tests effectués , elle divise les tableaux en plusieurs sous-ensembles d'entraînements de test aléatoires.
- ***from sklearn.model_selection import GridSearchCV:*** pour appliquer la cross validation.
- ***from sklearn.ensemble import RandomForestClassifier:*** *appliquer la forêt aléatoire.*
- ***from sklearn.metrics import accuracy_score :*** donne le score de la précision de la classification
- ***from sklearn import metrics***
- ***from sklearn.decomposition import KernelPCA :*** cette classe procède à une réduction de dimension en utilisant le noyau . Elle est plutôt appelée Analyse en composantes principales du noyau.
- ***from sklearn.preprocessing import StandardScaler, KernelCenterer:*** StandardScaler permet de standardiser les données. et KernelCenterer : centrer la matrice Kernel dans l'ACP à noyaux.
- ***from sklearn.svm import SVC:*** pour appliquer le modèle SVM.

- **Seaborn:**

Cette bibliothèque nous a permis de créer nos graphiques statistiques en Python.

- **Numpy :**

Elle nous a permis de pouvoir créer nos matrices.

- ***from numpy import sqrt*** : permet de faire la racine carrée.

- **Matplotlib** : elle est essentielle pour le traçage des courbes avec la création des différents axes. Comme pour Scikit - Learn, nous avons utilisé une de ces classes :

- ***from matplotlib.collections import LineCollection :*** elle permet de tracer plusieurs lignes sur une figure .

- **Scipy :**

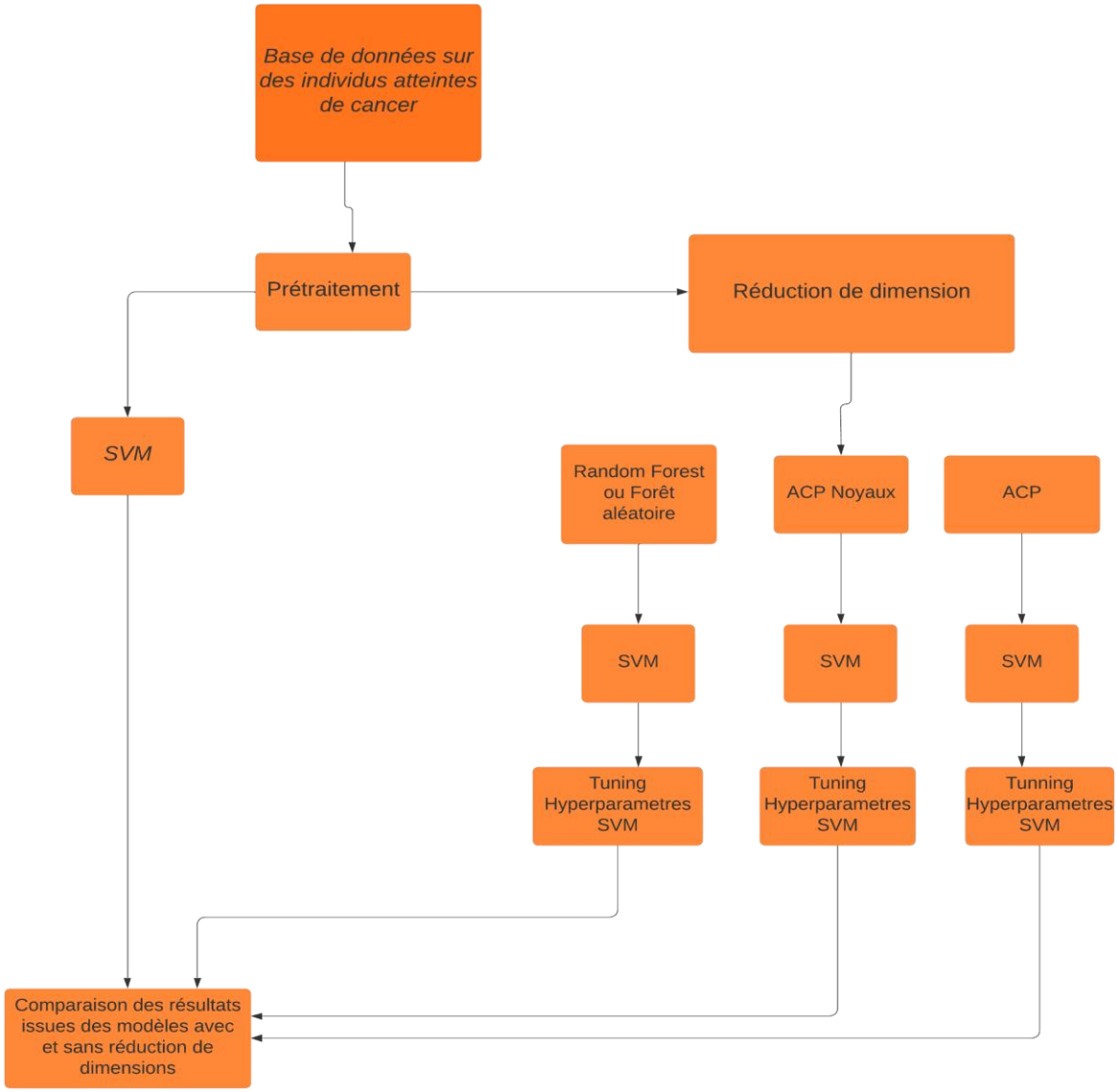
Elle concerne le calcul scientifique en général comme l'optimisation, l'algèbre linéaire .

- ***from scipy.spatial.distance import pdist, squareform :*** calcul les distances dans un espace à n dimension .
- ***from scipy import exp*** : permet de calculer l'exponentielle de tous les éléments du tableau d'entrée .
- ***from scipy.linalg import eigh :*** affiche les valeurs propres et vecteurs propres de la matrice X. Elle résout un problème de valeurs propres généralisé pour une matrice symétrique .

- **Random** :

Elle permet la génération de nombres aléatoires.

IV. Présentation de votre protocole expérimental au travers d'un diagramme de type workflow.



V. Présentation, analyse et discussion des résultats obtenus pour chaque méthode vis-à-vis des tâches initiales.

- Pré-traitement des données:

Comme nous l'avons indiqué dans la première partie, notre base contient 408 observations et 719 variables.
Les premières lignes et colonnes de notre base de données :

Out[24]:

	Row.names	CLINIC.OS_MONTHS	CLINIC.OS_STATUS	CLINIC.study	EXP.A1CF	EXP.ABI1	EXP.ABL1	EXP.ABL2	EXP.ACKR3	EXP.ACSL3	...	EXP.ZFHX
1	TCGA.05.4249.01	38.045974	0:LIVING	luad	0.3220	2116.0026	2180.2909	1272.6949	116.2264	1660.0089	...	1461.963
2	TCGA.05.4382.01	19.910069	0:LIVING	luad	0.0000	2389.6601	2076.4421	975.4037	733.2279	2512.9329	...	1141.667
3	TCGA.05.4384.01	13.996187	0:LIVING	luad	0.0000	1534.5144	2015.8827	1032.9872	310.9346	2422.7245	...	1532.510
4	TCGA.05.4389.01	45.007726	0:LIVING	luad	36.2416	2100.5168	1302.3490	733.5570	211.0738	1677.8523	...	461.691
5	TCGA.05.4390.01	36.994617	0:LIVING	luad	0.0000	1182.1811	1359.7318	709.5641	297.6209	2526.5847	...	552.329

5 rows × 719 columns

En première étape, nous avons supprimé les variables qui ne nous intéressent pas "CLINIC.study","CLINIC.OS_MONTHS","Row.names" .

En deuxième étape, nous avons affiché les valeurs manquantes, nous avons 7 valeurs manquantes dans la colonne "CLINIC.OS STATUS" et comme c'est un petit nombre et que les variable sont binaire qualitative nous avons décidé de les supprimer car on ne peut pas remplacer par la moyenne. Nous avons donc maintenant 401 lignes x 716 colonnes.

Nous avons affiché la répartition des deux modalités de la variable CLINIC.OS STATUS et nous avons remarqué que les deux modalités ne sont pas équilibrées, mais ceci n'est pas dans la cadre de notre projet, donc on ne va pas la traiter.

Ensuite, nous avons renommés les modalités de la variable CLINIC.OS STATUS par 0 et 1 au lieu de 0:LIVING, 1:DECEASED pour faciliter le traitement.

```
Entrée [441]: df.head()
```

Out[441]:

	CLINIC.OS_STATUS	EXP.A1CF	EXP.ABI1	EXP.ABL1	EXP.ABL2	EXP.ACKR3	EXP.ACSL3	EXP.ACSL6	EXP.ACVR1	EXP.ACVR2A	...	EXP.ZFHX3	EXP.ZMYM2
1	0	0.3220	2116.0026	2180.2909	1272.6949	116.2264	1660.0089	7.4050	1453.9566	390.2115	...	1461.9636	2058.5913
2	0	0.0000	2389.6601	2076.4421	975.4037	733.2279	2512.9329	6.2219	1322.1554	388.8692	...	1141.6675	1118.2683
3	0	0.0000	1534.5144	2015.8827	1032.9872	310.9346	2422.7245	7.9414	789.2486	287.7214	...	1532.5107	2143.5553
4	0	36.2416	2100.5168	1302.3490	733.5570	211.0738	1677.8523	37.9195	801.0067	225.5034	...	461.6913	1227.1812
5	0	0.0000	1182.1811	1359.7318	709.5641	297.6209	2526.5847	9.5801	481.5584	228.0057	...	552.3296	1187.2904

5 rows × 716 columns

Ensuite, nous avons déterminé une variable “X” qui sera l’ensemble de nos gènes (715).
Ainsi que la variable cible “y” qui sera 'CLINIC.OS STATUS'.

Nous avons centré et réduit nos données (l’expression des gènes) et l’avons enregistré dans la variable nommée “scaled_X”.

Entrée [249]:
```
scaled_X
```

Out[249]:

	EXP.A1CF	EXP.ABI1	EXP.ABL1	EXP.ABL2	EXP.ACKR3	EXP.ACSL3	EXP.ACSL6	EXP.ACVR1	EXP.ACVR2A	EXP.AFDN	...	EXP.ZFHX3	EXP.ZMYM2	EXP.Z
0	0.001071	0.232383	0.306503	0.493178	0.003393	0.086238	0.008132	0.396160	0.220080	0.953144	...	0.248586	0.432067	0.3
1	0.000000	0.280135	0.288990	0.361747	0.027690	0.181831	0.006833	0.350323	0.219101	0.390530	...	0.187490	0.213270	0.2
2	0.000000	0.130916	0.278777	0.387205	0.011060	0.171721	0.008721	0.164994	0.145286	0.457445	...	0.262043	0.451837	0.3
3	0.120495	0.229681	0.158448	0.254829	0.007128	0.088238	0.041643	0.169083	0.099881	0.204711	...	0.057786	0.238612	0.2
4	0.000000	0.069435	0.168125	0.244221	0.010536	0.183361	0.010521	0.057988	0.101707	0.221747	...	0.075075	0.229331	0.2
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
396	0.001663	0.150390	0.371343	0.543133	0.022022	0.071753	0.172501	0.386163	0.126976	0.392937	...	0.216340	0.171667	0.2
397	0.001336	0.291612	0.177450	0.447399	0.005097	0.321524	0.006177	0.398689	0.068427	0.483922	...	0.425143	0.282414	0.2
398	0.000000	0.109118	0.335199	0.558541	0.010427	0.166736	0.016589	0.471745	0.186656	0.345106	...	0.159167	0.289228	0.2
399	0.019358	0.090044	0.244673	0.230256	0.011388	0.054698	0.004618	0.134058	0.146583	0.703354	...	0.153092	0.243286	0.3
400	0.000000	0.120644	0.210865	0.683217	0.013013	0.073235	0.096704	0.432420	0.120612	0.986976	...	0.373634	0.195014	0.4

401 rows × 715 columns

Nous avons ensuite divisé “scaled_X” par racine de n et l’avons enregistré dans une deuxième variable qui s’appelle “scaled_X2” .

En effet, comme l’on sait qu' il est fortement recommandé avant d’appliquer l’ACP(et ACP à noyaux) de centrer et réduire les données et divisées par racine de n(401), nous allons donc utiliser “X_scaled2” pour les méthodes ACP et ACP à noyaux.
Alors que “X_scaled” sera utilisé comme X en entrée du modèle de prédiction avec les données initiales(sans réduction de dimension).

ensuite, nous avons créé une variable ‘status’ pour enregistrer la variable cible CLINIC.OS STATUS pour pouvoir l’afficher sur notre graphique des individus à la fin.

ensuite, nous avons créé une variable features pour enregistrer les noms des colonnes des variables (les gènes).

Ensuite nous avons affiché la matrice de corrélation entre les gènes:

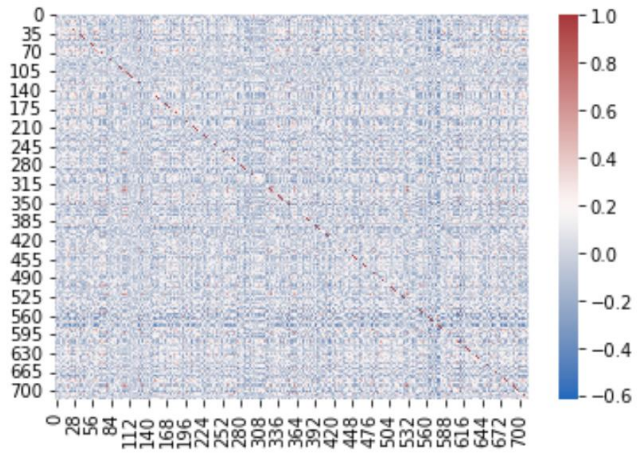
Entrée [172]:
```
#afficher la matrice de correlation entre les genes:
pw_corr=pd.DataFrame(scaled_X).corr().round(3)
print(pw_corr)
```

```
           EXP.A1CF  EXP.ABI1  EXP.ABL1  EXP.ABL2  EXP.ACKR3  EXP.ACSL3  \
EXP.A1CF      1.000     0.024    -0.027    -0.068     -0.075      0.052
EXP.ABI1      0.024     1.000    -0.295    -0.217      0.402      0.019
EXP.ABL1     -0.027    -0.295     1.000     0.437     -0.046      0.112
EXP.ABL2     -0.068    -0.217     0.437     1.000     -0.182      0.063
EXP.ACKR3    -0.075     0.402    -0.046    -0.182      1.000      0.000
...             ...       ...       ...       ...        ...        ...
EXP.ZNF429   -0.040    -0.247    -0.040     0.012     -0.229     -0.076
EXP.ZNF479   -0.012    -0.060    -0.044    -0.017     -0.030     -0.019
EXP.ZNF521   -0.048     0.015     0.082     0.070     -0.064     -0.023
EXP.ZNRF3    -0.056    -0.094     0.077     0.028     -0.002      0.061
EXP.ZRSR2    -0.028    -0.336     0.100     0.151     -0.232     -0.170

           EXP.ACSL6  EXP.ACVR1  EXP.ACVR2A  EXP.AFDN  ...  EXP.ZFHX3  \
EXP.A1CF      -0.031     -0.106       0.057    -0.012  ...      0.016
EXP.ABI1       0.005      0.076       0.176    -0.315  ...     -0.309
EXP.ABL1      -0.023      0.048      -0.159     0.116  ...      0.277
EXP.ABL2      -0.013      0.136      -0.169     0.310  ...      0.393
EXP.ACKR3      0.006      0.238       0.136    -0.296  ...     -0.140
...              ...        ...         ...       ...  ...        ...
EXP.ZNF429    -0.040     -0.052       0.037     0.203  ...      0.183
EXP.ZNF479    -0.013     -0.047      -0.051     0.026  ...     -0.031
EXP.ZNF521     0.110      0.117       0.050     0.033  ...      0.047
EXP.ZNRF3     -0.030     -0.077       0.022     0.071  ...      0.166
EXP.ZRSR2     -0.028     -0.040       0.058     0.214  ...      0.237

           EXP.ZMYM2  EXP.ZMYM3  EXP.ZNF331  EXP.ZNF384  EXP.ZNF429  \
EXP.A1CF       0.016     -0.095      -0.075      -0.006      -0.040
EXP.ABI1       0.059     -0.364      -0.225       0.111      -0.247
EXP.ABL1       0.042      0.303       0.136       0.155      -0.040
EXP.ABL2       0.088      0.251       0.169       0.012       0.012
EXP.ACKR3      0.021     -0.209      -0.249       0.174      -0.229
...              ...        ...         ...         ...         ...
EXP.ZNF429    -0.028      0.182       0.175      -0.177       1.000
EXP.ZNF479    -0.039     -0.029      -0.042       0.034      -0.000
EXP.ZNF521    -0.072      0.002       0.046      -0.036      -0.008
EXP.ZNRF3      0.143      0.202       0.010       0.142      -0.005
EXP.ZRSR2     -0.108      0.273       0.163      -0.112       0.342
```

et le Heat Map correspondante.

Entrée [47]:
```
#Le heatmap de la matrice de correlation des genes:
sns.heatmap(pw_corr,cmap='vlag')
```

Out[47]: <AxesSubplot:>



Nous pouvons bien remarquer que les variables sont corrélées entre elles.
Maintenant nos données sont prêtes au traitement.

- **Appliquer le modèle de prédiction SVM sur les variables initiales sans réduction de dimension:**

Nous avons essayé plusieurs modèles de prédiction avant de choisir la SVM.
La SVM a été la meilleure parmi les modèles que nous avons essayés(Régression Logistique, Forêt Aléatoire) en termes de training sur les données car la SVM est la plus efficace dans le traitement des données de grandes dimensions. c'est-à-dire le score le plus élevé obtenu c'était celle de la SVM.

Nous avons avant tout divisé notre base en X_train et X_test et y_train et y_test (20%).
Le modèle SVM à pris en entrée les scaled_X qui sont centrés et réduites.
Et nous avons obtenu comme scores du modèle:

```
crée [30]: print('Training score:',(SV.score(X_train,y_train)*100).round(1),'%')
           print('test score:',(SV.score(X_test,y_test)*100).round(1),'%')

           Training score: 85.3 %
           test score: 64.2 %

crée [31]:
           #train score=85.3%     test_score=64.2%
           #21.1% de gap donc le modele est overfit
```

C'est-à- dire 21.1% de gap entre les deux scores, En effet, comme le test score est très bas par rapport au training score, cela veut dire que le modèle est sur-appris, et par conséquent le modèle ne va pas être performant avec des nouvelles données.

Le overfit est dû à la complexité du modèle c'est-à-dire la grande dimensionnalité du dataset(715). En d'autre terme, le nombre d'observations est petit(401) par rapport au nombre des variables(715).

On va voir par la suite, si la réduction de dimensions va résoudre le problème d'overfitting et si elle va améliorer la prédiction du modèle SVM.

- **ACP:**

Pour commencer l'ACP, nous avons choisi 33 dimensions qui expliquent 63% de la variance des variables initiales.
Nous avons décidé de maintenir cette dimensionnalité après plusieurs essais sur le modèle SVM et nous avons remarqué que c'est la meilleure dimensionnalité qui fait balancer entre une bonne variance en ACP et un bon test score en SVM.

Donc , avec l'analyse de l'ACP , nous allons tout d'abord évaluer la valeur des variances expliquées .

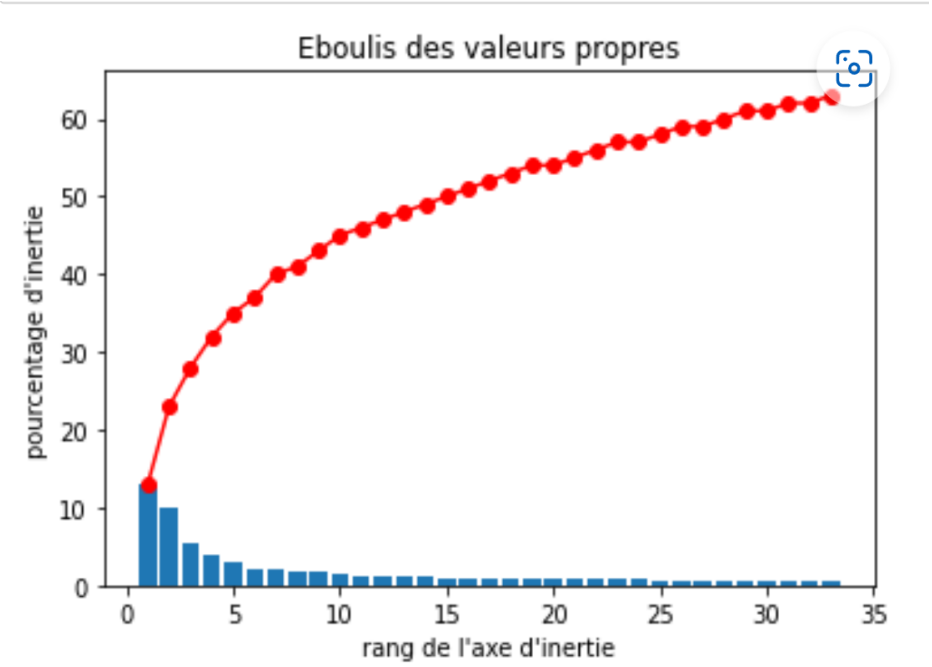
```
trée [300]: #afficher la variance expliqué arrondi à 2 chiffres pour chaque component :
            var_ratio = (pca_model.explained_variance_ratio_*100).round(2)
            var_ratio

Out[300]: array([12.97, 10.02,  5.46,  3.77,  2.99,  2.2 ,  2.12,  1.88,  1.65,
                  1.45,  1.23,  1.18,  1.12,  1.04,  0.99,  0.96,  0.89,  0.86,
                  0.83,  0.82,  0.78,  0.77,  0.74,  0.7 ,  0.67,  0.65,  0.61,
                  0.6 ,  0.59,  0.56,  0.55,  0.54,  0.53])
```

Nous avons donc pour chaque dimension , la part(au niveau de la variance) , qu'elle représente sur la variance totale.
Nous remarquons que les deux premières dimensions avec respectivement des valeurs de 12,97 et 10,02 ont une variance beaucoup plus élevée et significative que les autres .Ces résultats nous montrent que les deux premiers facteurs expliquent les mieux la variance des données. Et si on les cumule, on obtient les valeurs suivantes :

```
01]: #la variance cumulée
     var_ratio_cum = scree.cumsum().round()
     var_ratio_cum

1]: array([13., 23., 28., 32., 35., 37., 40., 41., 43., 45., 46., 47., 48.,
           49., 50., 51., 52., 53., 54., 54., 55., 56., 57., 57., 58., 59.,
           59., 60., 61., 61., 62., 62., 63.])
```

Quand on additionne les inerties associées aux 33 axes, on obtient l'inertie expliquée totale qui est égale à 63%.

Nous avons donc ici , l'éboulis des valeurs propres.



Les barres bleues représentent la variance expliquée par chaque valeur propre et la ligne rouge représente la variance cumulée pour ces 33 valeurs propres.
Comme on l'avait dit en choisissant 33 composantes, on a expliqué environ 63% de la variation des données. ce qui n'est pas très élevée mais pourrait être la meilleure dans notre cas d'étude.

Nous avons obtenu les valeurs singulières suivantes de l'ACP:

```
e [534]: #afficher les valeurs singuliers:
         print('les valeurs singulieres de lACP sont:\n',(pca_model.singular_values_).round(2))

         les valeurs singulieres de lACP sont:
          [1.24 1.09 0.8  0.67 0.6  0.51 0.5  0.47 0.44 0.41 0.38 0.37 0.36 0.35
          0.34 0.34 0.32 0.32 0.31 0.31 0.3  0.3  0.3  0.29 0.28 0.28 0.27 0.27
          0.27 0.26 0.25 0.25 0.25]
```

Nous avons enregistré notre nouvel espace réduit dans une variable 'PCA X'.
Et dans la variable 'pcs' la table de corrélation entre les variables initiales et les axes principaux.
Puis nous avons affiché la variable 'pcs' transposée pour faciliter l'interprétation.

Nous rappelons les deux objectifs d'appliquer l'ACP, en plus de la réduction de dimensions, nous pouvons étudier la variabilité des individus ainsi que le lien entre les variables.

En effet, il existe deux façons pour interpréter la corrélation entre les variables et les axes principaux, soit à l'aide du cercle de corrélation soit à l'aide du tableau de corrélation.
nous allons présenter les deux façons par la suite.

- **<u>Table de corrélations:</u>**

Nous allons nous pencher sur la table des corrélations entre les 33 axes et les variables initiales en essayant d'en pouvoir tirer des résultats:

Out[48]:

	PCA_X1	PCA_X2	PCA_X3	PCA_X4	PCA_X5	PCA_X6	PCA_X7	PCA_X8	PCA_X9	PCA_X10	...	PCA_X24	PCA_X25	PCA_X26	PCA_X27
EXP.A1CF	0.00	-0.01	-0.01	0.01	0.01	0.01	-0.00	0.03	0.01	-0.02	...	-0.02	-0.02	-0.03	0.01
EXP.ABI1	0.06	-0.01	0.03	0.04	-0.05	-0.02	-0.01	0.04	0.06	-0.04	...	-0.01	0.05	-0.02	-0.02
EXP.ABL1	-0.03	0.05	0.01	-0.05	-0.01	0.01	0.02	0.06	-0.07	0.05	...	0.00	-0.04	0.08	0.01
EXP.ABL2	-0.05	0.06	0.00	0.00	-0.05	0.08	-0.01	0.04	-0.10	0.12	...	-0.04	-0.06	-0.01	-0.10
EXP.ACKR3	0.06	0.02	0.04	-0.00	-0.05	-0.07	-0.02	0.03	0.01	-0.04	...	0.02	0.04	0.02	0.02
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
EXP.ZNF429	-0.02	-0.01	-0.02	0.01	0.01	-0.03	-0.01	-0.02	0.00	0.01	...	0.02	-0.00	-0.01	0.03
EXP.ZNF479	0.00	-0.00	0.00	-0.00	-0.00	0.01	-0.00	-0.00	-0.00	-0.01	...	0.00	-0.00	-0.01	0.01
EXP.ZNF521	-0.01	0.00	0.01	-0.00	-0.00	0.01	0.01	-0.00	0.01	0.01	...	0.01	0.02	-0.01	0.01
EXP.ZNRF3	0.00	0.02	-0.02	0.00	0.00	-0.01	0.03	-0.03	-0.01	-0.01	...	-0.04	0.07	0.01	-0.03
EXP.ZRSR2	-0.05	0.00	-0.01	-0.04	0.10	-0.07	-0.00	-0.03	0.00	0.06	...	0.02	-0.04	-0.07	-0.08

715 rows × 33 columns

Comme les axes principaux sont des nouvelles variables obtenues par des combinaisons linéaires des variables initiales, alors on peut dire que chaque axe synthétise les variables les plus corrélées avec cet axe principal.

Comme ce sont 715 variables initiales, nous ne pouvons pas en tirer de grandes choses.
Mais, nous pouvons remarquer que les corrélations entre les 33 facteurs principaux et les variables initiales ne sont pas très fortes.

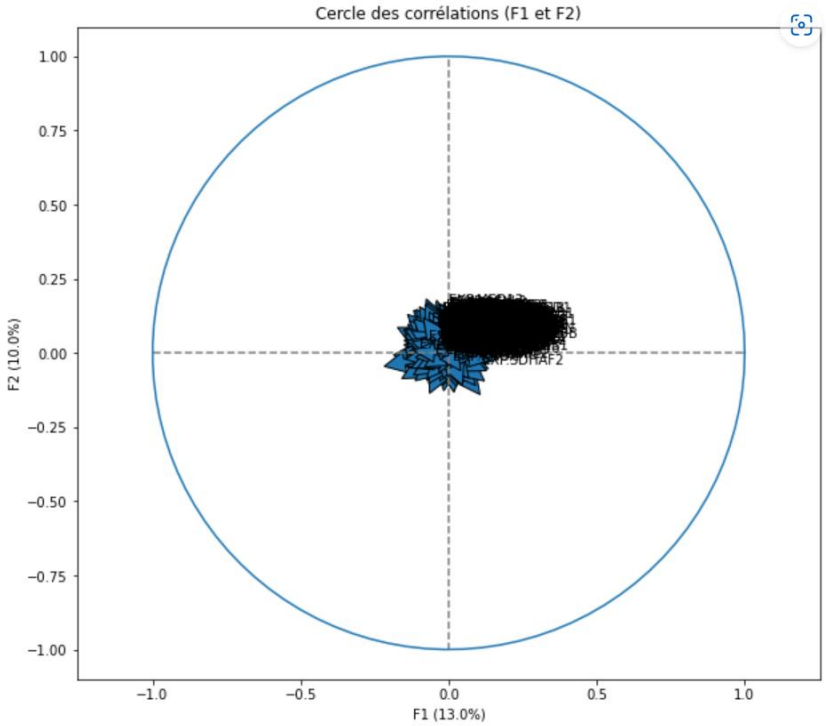
Et, si nous voulons faire l'analyse, nous pouvons rechercher parmi les 715 gènes, les gènes les plus corrélées (positivement où bien négativement) avec les axes principaux.

Plus la corrélation est forte entre une variable et un axe factoriel, plus cet axe est représentatif de cette variable.
En conséquence, nous avons la possibilité d'inférer cet axe à l'ensemble des variables les plus corrélées avec-il.

Cette étape va nous permettre par la suite de pouvoir interpréter et comprendre la dernière étape, c'est-à-dire, lorsqu'on va projeter les individus sur les axes principaux dans l'espace réduit F1 et F2.

- **Cercle de corrélations:**

Pour étudier la corrélation entre les variables. Nous allons analyser le cercle des corrélations entre les variables initiales et les axes principaux.



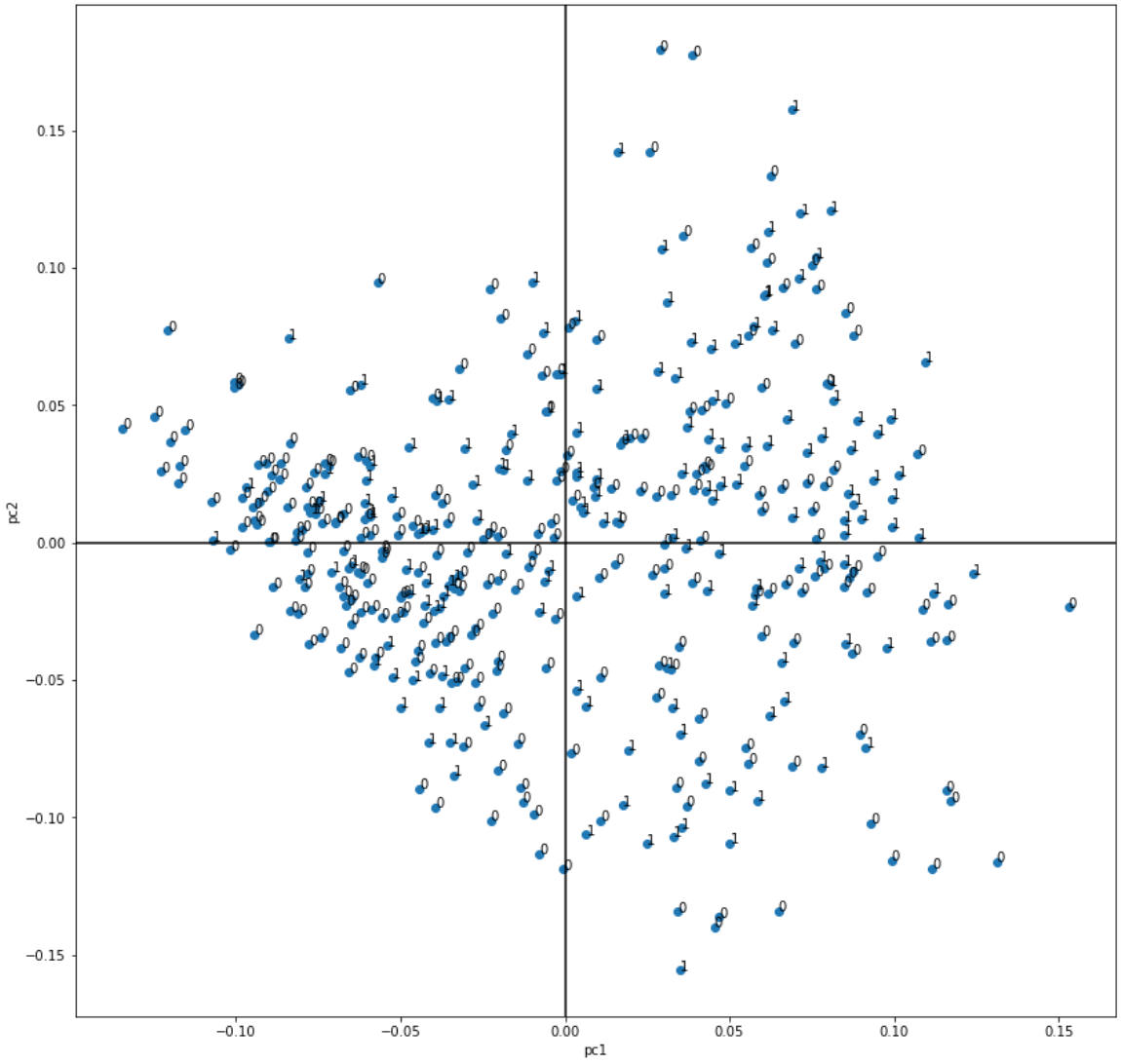
On sait que chaque flèche représente une variable de notre base de données,
et comme nous avons beaucoup de variables, on voit bien que les flèches sont l'une sur les autres. Ce qui rend difficile l'analyse de corrélation entre les variables et les axes principaux.

Mais on peut dire que, en général, l'axe des abscisses représente le premier axe d'inertie F1, et l'axe des ordonnées représente F2.
Et la projection de la flèche sur F1 correspond au coefficient de corrélation entre cette variable (flèche) et F1.

En effet, une flèche qui est petite sur le premier plan factoriel, signifie qu'elle est faiblement corrélée à la première composante principale F1, et faiblement corrélée aussi à F2.

Pour cela, il est préférable de ne interpréter que les flèches les plus longues, et si certaines variables que l'on cherche à analyser sont mal représentées, on peut tenter d'afficher le second plan factoriel, avec F3 et F4, etc.

En dernière étape d'analyse de l'ACP, on projette les individus sur le premier plan factoriel F1, F2 pour obtenir la graphique de la projection des individus sur les axes principaux dans l'espace réduit F1 et F2:
0 ce sont les individus vivants et 1 ce sont les individus décédés.



Après avoir interpréter les corrélations entre les variables initiales et les axes principaux, on peut en parallèle, interpréter les individus projetés sur ces axes de la manière suivante:
La présence d'un individu sur la partie droite de l'axe 1 signifie que cet individu est caractérisé par la variable la plus corrélée avec cet axe.

La présence d'un individu sur la partie gauche de l'axe 1 signifie que cet individu est très faiblement caractérisé par la variable la plus corrélée avec cet axe.

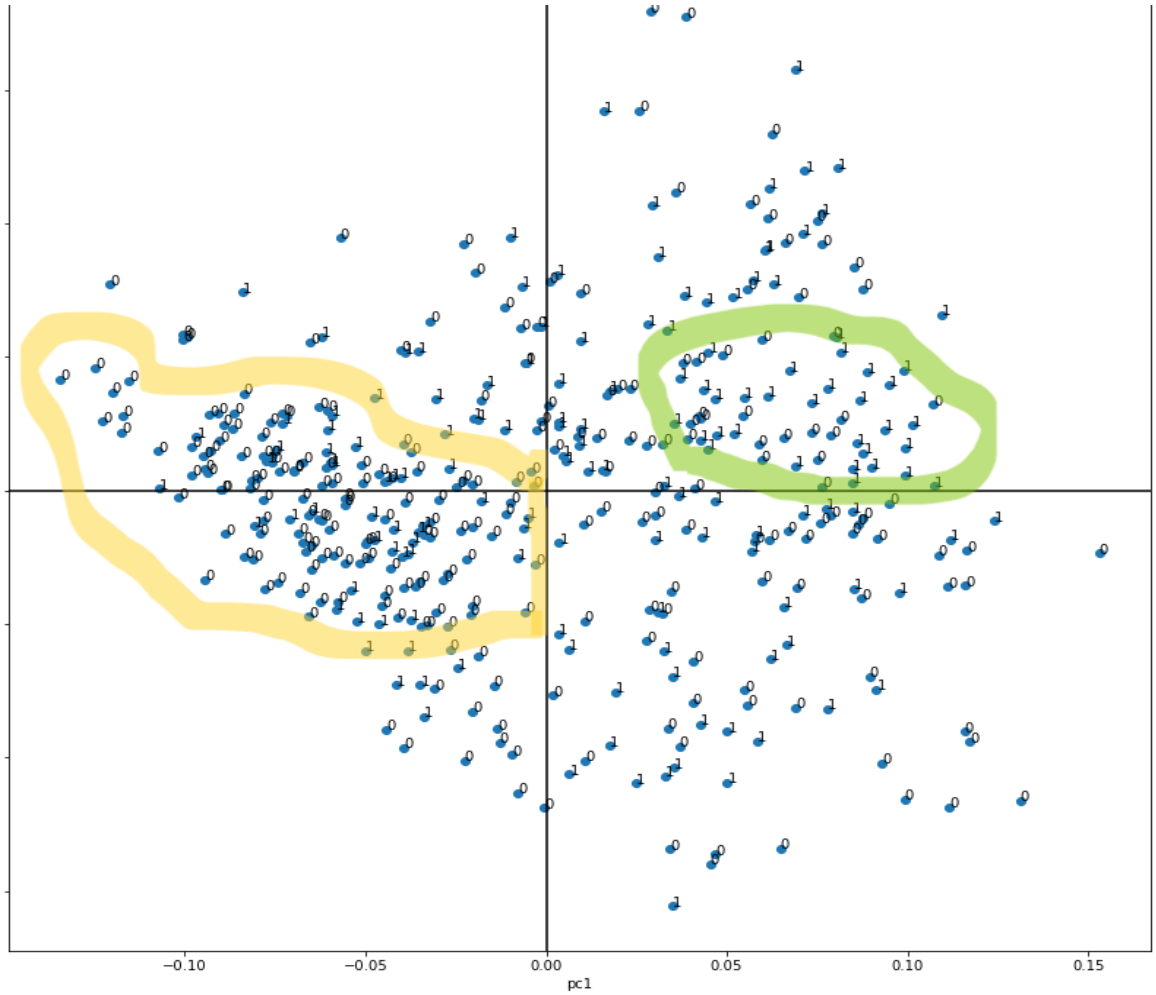
C'est-à- dire, quand on se déplace le long de l'axe des abscisses de gauche à droite, on se déplace vers les points pour lesquels la valeur de l'axe 1 est grande.

De même, nous pouvons interpréter la second axe comme suit:

La présence d'un individu sur la partie haute de l'axe 2 signifie que cet individu est caractérisé par la variable la plus corrélée avec cet axe.

La présence d'un individu sur la partie basse de l'axe 2 signifie que cet individu est très faiblement caractérisé par la variable la plus corrélée avec cet axe.

C'est-à- dire, quand on se déplace le long de l'axe des ordonnées de bas en haut , on se déplace vers les points pour lesquels la valeur de l' axe 2 est grande.
Nous pouvons remarquer que les individus sont projetés de façon spécifiques comme suit:



D'après ce graphique on constate que, les individus vivants(0) sont concentrés sur la partie gauche du premier axe(entouré en jaune).
Cela nous indique que ces individus sont très faiblement caractérisés par les génomes les plus corrélées avec cet axe.

mais comme le nombre des génomes est très grand, nous avions des difficultés à pouvoir trouver ces génomes.

Si nous supposons par exemple que la premier axe est corrélé avec les gènes suivants; ABL1, ABL2.
Alors, ce groupe d'individus très faiblement corrélés avec cet axe c'est à dire ces individus ont l'expression de ces deux gènes est très faible où bien l'effet de ces gènes est peu présente dans la corps de ce groupe d'individus et qui sont (la plupart d'entre eux 0) vivantes.

De même, on constate que, les individus décédés(1) sont concentrés sur la partie droite du premier axe(entouré en vert).
Cela nous indique que ces individus sont très fortement caractérisés par les génomes les plus corrélées avec cet axe.

Si nous supposons par exemple que la premier axe est corrélé avec les gènes suivants; ABL1, ABL2.
Alors, ce groupe d'individus très fortement corrélés avec cet axe c'est à dire ces individus ont l'expression de ces deux gènes est très forte où bien l'effet de ces gènes est très présente dans la corps de ce groupe d'individus et qui sont (la plupart d'entre eux 1)décédés.

- **Appliquer SVM sur les variables réduites issues de l'ACP:**

Nous avons obtenu comme résultats de ce modèle:

```
Entrée [54]: print('Training score:',(SV.score(X_train,y_train)*100).round(1),'%')
             print('test score:',(SV.score(X_test,y_test)*100).round(1),'%')

             Training score: 80.3 %
             test score: 69.1 %

Entrée [67]: #Training score: 80.3 %
             #test score: 69.1 %
```

Le gap a baissé de 21.1% à 11.2% ce qui nous indique que la modèle est mieux qu'avant d'appliquer l'ACP et nous avons pu diminuer le gap entre les deux scores (l'overfitting).

En appliquant l'ACP, la performance du modèle SVM a bien été améliorée.
C'est-à-dire, nous avons pu améliorer le modèle en conservant seulement 33 dimensions(63% de la variance des données). En d'autres termes, le test score a augmenté de 64.2% à 69.1% ce qui est très bien et ce qui nous indique l'intérêt d'appliquer l'ACP dans le modèle.

Mais comme il y a un overfitting (gap 21.1%), le modèle ne va pas être performant avec des nouvelles données. Nous allons essayer d'améliorer le modèle et de réduire le gap entre les scores de test et de train du modèle en déterminant les hyperparamètres de SVM.

Réglage des hyperparamètres

Dans SVC, il existe 3 hyperparamètres importants qui peuvent être réglés pour améliorer le score du modèle.

Le paramètre Kernel permet de séparer les données en sélectionnant le type d'hyperplan.
Il existe 4 types d'hyperplans que nous pouvons utiliser pour ajuster notre modèle, mais nous utiliserons le noyau RBF et le linéaire car ce sont les plus utilisés. Gamma définit l'influence d'un seul exemple d'entraînement. Plus le gamma est grand, plus les autres exemples doivent être proches pour être affectés. C Paramètre de régularisation. La force de la régularisation est inversement proportionnelle à C. Doit être strictement positive. La pénalité est une pénalité de L2 au carré. Il est commun à tous les noyaux SVM, échange une mauvaise classification des exemples de formation contre la simplicité de la surface de décision. Un C bas rend la surface de décision lisse, tandis qu'un C élevé vise à classer correctement tous les exemples d'apprentissage. Nous utiliserons la méthode GridSearchCV, qui choisira le bon ensemble de paramètres pour régler notre modèle en exécutant tous les scénarios possibles après avoir alimenté la plage de valeurs.

Après le réglage des hyperparamètres du modèle SVM avec les données réduites de l'ACP, nous obtenons les résultats suivantes:

```
Entrée [64]: print('Training score:',(grid.score(X_train,y_train)*100).round(1),'%')
             print('test score:',(grid.score(X_test,y_test)*100).round(1),'%')

             Training score: 80.3 %
             test score: 69.1 %
```

```
Entrée [67]: #Training score: 80.3%
             #test score: 69.1 %
             #meme resultats! le modele ne peut pas encore ameliorer
```

On remarque qu' il n'y a pas de changements et le modèle ne peut plus être amélioré.

- **ACP À Noyaux:**

Comme dit plus haut, l'ACP à noyau consiste à projeter les données par l'intermédiaire d'une application non linéaire dans un espace de dimension élevée dénommé espace des caractéristiques où l'ACP linéaire est appliquée.
Tout d'abord , nous avons calculé la distance euclidienne .

```
Entrée [74]: gamma=0.01
             n_components=33

             #calculer la distance Euclidienne au carré pour tous les paires de points dans le dataset:
             sq_dists = pdist(scaled_X2, 'sqeuclidean')
             sq_dists

Out[74]: array([0.05810359, 0.04577539, 0.06115714, ..., 0.05339948, 0.03917945,
                0.05634791])
```

Puis nous les avons converties en matrice symétrique.

```
Entrée [75]:
#convertir les distances en une matrice symetrique: mxm
mat_sq_dists = squareform(sq_dists)
mat_sq_dists
```

```
Out[75]: array([[0.        , 0.05810359, 0.04577539, ..., 0.04403798, 0.04818495,
                 0.04754326],
                [0.05810359, 0.        , 0.05391358, ..., 0.06790096, 0.06026519,
                 0.06345204],
                [0.04577539, 0.05391358, 0.        , ..., 0.03749297, 0.04165474,
                 0.04987605],
                ...,
                [0.04403798, 0.06790096, 0.03749297, ..., 0.        , 0.05339948,
                 0.03917945],
                [0.04818495, 0.06026519, 0.04165474, ..., 0.05339948, 0.        ,
                 0.05634791],
                [0.04754326, 0.06345204, 0.04987605, ..., 0.03917945, 0.05634791,
                 0.        ]])
```

La matrice Kernel représente la matrice à noyaux

```
Entrée [76]:
#calculer la matrice de kernel
K =np.exp(-gamma * mat_sq_dists)
K
```

```
Out[76]: array([[1.        , 0.99941913, 0.99954235, ..., 0.99955972, 0.99951827,
                 0.99952468],
                [0.99941913, 1.        , 0.99946101, ..., 0.99932122, 0.99939753,
                 0.99936568],
                [0.99954235, 0.99946101, 1.        , ..., 0.99962514, 0.99958354,
                 0.99950136],
                ...,
                [0.99955972, 0.99932122, 0.99962514, ..., 1.        , 0.99946615,
                 0.99960828],
                [0.99951827, 0.99939753, 0.99958354, ..., 0.99946615, 1.        ,
                 0.99943668],
                [0.99952468, 0.99936568, 0.99950136, ..., 0.99960828, 0.99943668,
                 1.        ]])
```

Il nous faut la centrer pour calculer les données .
L'astuce du noyau permet de trouver les vecteurs de coefficients représentant les vecteurs propres en solutionnant le problème de vecteurs propres .
Donc avec cette astuce , on peut rendre non-linéaire des méthodes linéaires

Si on veut d'abord centrer les données transformées, il faut utiliser une matrice de Gram corrigée car il faut diagonaliser la matrice de Gram (centrée) K= XXT au lieu de la matrice de covariance. C=XTX

```
Entrée [78]:
#obtenir les valeurs propres dans l'ordre décroissante avec leurs vecteurs propres
eigvals, eigvecs = eigh(K)
#afficher les valeurs propres:
print(eigvals)
```

```
[-5.94730108e-15  1.01031902e-05  1.03381629e-05  1.07257302e-05
  1.15090582e-05  1.23895646e-05  1.26846920e-05  1.32609384e-05
  1.33868348e-05  1.38269986e-05  1.43001060e-05  1.48520120e-05
  1.52039932e-05  1.61171386e-05  1.65981153e-05  1.70204558e-05
  1.71738085e-05  1.73311202e-05  1.78343787e-05  1.83528205e-05
  1.91538551e-05  1.92963256e-05  1.95191754e-05  2.01230465e-05
  2.02754958e-05  2.10055133e-05  2.13996577e-05  2.17735221e-05
  2.19650559e-05  2.26005008e-05  2.32220032e-05  2.36073743e-05
  2.39042535e-05  2.40956758e-05  2.54355595e-05  2.56837778e-05
  2.63818231e-05  2.66449697e-05  2.70440326e-05  2.77130986e-05
  2.81184450e-05  2.85297500e-05  2.87045075e-05  2.90981580e-05
  2.96924494e-05  3.06310239e-05  3.08838246e-05  3.14365855e-05
  3.21134163e-05  3.23303389e-05  3.33173512e-05  3.35869546e-05
  3.44923431e-05  3.50007786e-05  3.52312371e-05  3.56930260e-05
  3.62053438e-05  3.69290243e-05  3.79933496e-05  3.85453910e-05
  3.90891185e-05  3.94773980e-05  3.99191510e-05  4.04071018e-05
  4.08218920e-05  4.14336912e-05  4.27928318e-05  4.33140860e-05
  4.37781633e-05  4.39384015e-05  4.40370878e-05  4.49736062e-05
  4.58426358e-05  4.61574203e-05  4.71400576e-05  4.81774387e-05
```

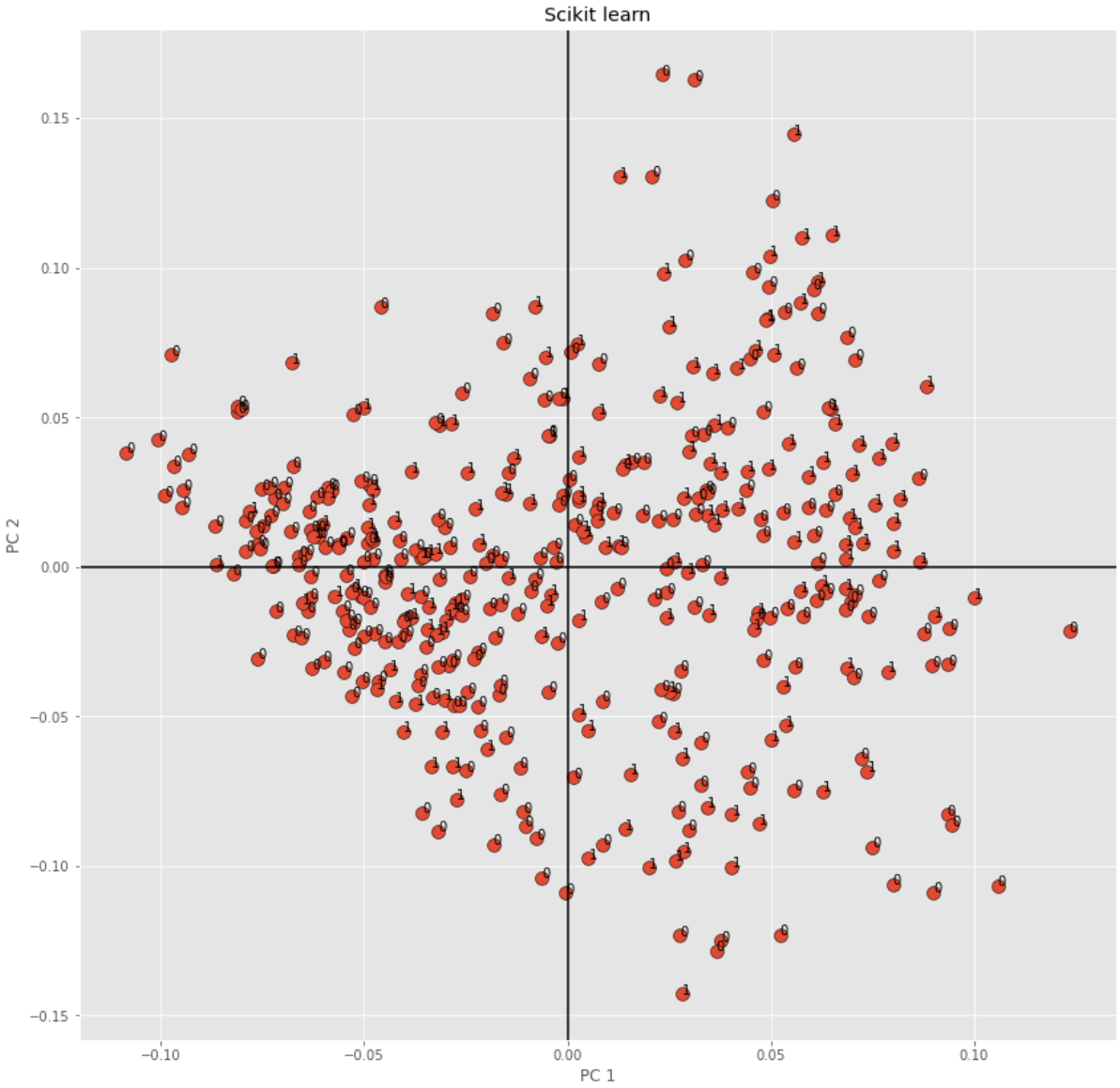
Entrée [79]:
```
#afficher les vecteurs propres:
print(eigvecs)
```

```
[[-0.04993762  0.01344243  0.02746446 ... -0.02503593  0.03208601
  -0.05824632]
 [-0.04993762 -0.00794124 -0.01930108 ...  0.0318496   0.00893399
  -0.0320663 ]
 [-0.04993762 -0.03523335 -0.01618895 ... -0.04403214  0.01188307
  -0.06702621]
 ...
 [-0.04993762 -0.02966272 -0.0208505  ... -0.02942392  0.03055559
  -0.0692945 ]
 [-0.04993762  0.01917306 -0.07721988 ... -0.0132397   0.01316937
  -0.06164489]
 [-0.04993762  0.04194632 -0.01661491 ...  0.01951251  0.04775664
  -0.07772056]]
```

Entrée [80]:
```
#obtenir le i vecters propres qui correspondent aux i valeurs propres:
X_pc = np.column_stack((eigvecs[:,-i] for i in range(1,n_components+1)))
X_pc
```

Out[80]:
```
array([[-0.05824632,  0.03208601, -0.02503593, ...,  0.03576365,
         -0.10232574,  0.0241981 ],
       [-0.0320663 ,  0.00893399,  0.0318496 , ..., -0.00746804,
        -0.06363934, -0.0832369 ],
       [-0.06702621,  0.01188307, -0.04403214, ..., -0.01216251,
        -0.01672659,  0.00434564],
       ...,
       [-0.0692945 ,  0.03055559, -0.02942392, ..., -0.02306535,
         0.01942192,  0.00533849],
       [-0.06164489,  0.01316937, -0.0132397 , ..., -0.04805985,
        -0.02768005,  0.01364359],
       [-0.07772056,  0.04775664,  0.01951251, ..., -0.04246679,
         0.01749308,  0.03587075]])
```

On peut ensuite facilement calculer les projections donnant les composantes principales.



On peut dire qu’on a la même interprétation que l'Analyse en Composantes Principales .

- **Appliquer SVM sur les variables réduites issues de l'ACP à noyaux:**

Nous avons obtenu comme résultats de ce modèle:

```
Entrée [79]: print('Training score:',(SV.score(X_train,y_train)*100).round(1),'%')
             print('test score:',(SV.score(X_test,y_test)*100).round(1),'%')

             Training score: 87.5 %
             test score: 65.4 %

Entrée [80]: #22 de gap==>overfitting
             #Training score: 87.5 %
             #test score: 65.4 %
             #on va essayer de resoudre le probleme:
```

c'est à dire 22.1% de gap entre les deux scores ce qui veut dire que le modèle est over fit.

Le test score a légèrement augmenté de 64.2%(SVM sans réduction de dimensions) à 65.4% ce qui est bien et ce qui nous indique l'intérêt d'appliquer l'ACP à noyaux dans le modèle.

La performance du modèle a bien été améliorée avec les 33 dimensions c'est-à-dire, nous avons pu améliorer le modèle en réduisant les variables à 33 avec la méthode ACP à noyaux.

Mais comme il y a un overfitting (gap 22.1%), le modèle ne va pas être performant avec des nouvelles données. Nous allons essayer de résoudre ce problème en déterminant les hyperparamètres du SVM.
Les mêmes hyperparamètres qu'auparavant seront maintenus.

Après la réglage des hyperparamètres, nous obtenons les résultats suivantes:

```
Entrée [87]: print('Training score:',(grid.score(X_train,y_train)*100).round(1),'%')
             print('test score:',(grid.score(X_test,y_test)*100).round(1),'%')

             Training score: 64.1 %
             test score: 63.0 %

Entrée [88]: #Training score: 64.1 %
             #test score: 63.0 %
             #underfit!
             #et le modele n'a pas amelioré!
```

Comme le train score est bas c'est-à-dire que le modèle n' a pas pu apprendre des données.
On remarque que le modèle est underfit (sous appris) c'est-à-dire les deux scores sont mauvais et donc le modèle ne peut plus être amélioré en fixant les hyperparamètres de la SVM.

- **Appliquer SVM sur les variables issues de la méthode Forêt Aléatoire:**

Comme nous l'avons déjà indiqué, nous avons sélectionné les 23 gènes les plus importants et qui influencent au maximum notre variable cible pour les utiliser en entrée de notre modèle SVM et voir les résultats de prédiction.

Nous avons obtenu comme résultats de ce modèle:

```
Entrée [526]: print('Training score:',(SV.score(X_train,y_train)*100).round(1),'%')
              print('test score:',(SV.score(X_test,y_test)*100).round(1),'%')

              Training score: 78.4 %
              test score: 72.8 %

Entrée [527]: #random=18
              #Training score: 78.4 %
              #test score: 72.8 %
              #gap=5.6%
```

5.6% de gap entre les deux scores ce qui veut dire que le modèle a de bonnes performances et n'est pas overfit (n'est pas surpris) car le gap est inférieur à 10.

Le test score à beaucoup augmenté de 64.2%(SVM sans réduction de dimensions) à 72.8% ce qui est très bien et ce qui nous indique l'intérêt d'appliquer la méthode de features selection avec forêt aléatoire dans le modèle SVM.

La performance du modèle SVM a bien été améliorée avec les 23 variables c'est -à -dire, nous avons pu améliorer le modèle en réduisant les variables à 23 avec cette méthode.

Nous allons essayer de réduire encore le gap entre les scores (5.6%) et d'améliorer la performance du modèle en déterminant les hyperparamètres du SVM.
Les mêmes hyperparamètres qu'auparavant seront maintenus.

Après la réglage des hyperparamètres, nous obtenons les résultats suivantes:

```
Entrée [530]: print('Training score:',(grid.score(X_train,y_train)*100).round(1),'%')
              print('test score:',(grid.score(X_test,y_test)*100).round(1),'%')

              Training score: 70.0 %
              test score: 71.6 %

Entrée [110]: #random=18
              #Training score:70.0%
              #test score: 71.6 %
              #gap=1.6%
```

Comme le gap entre les deux scores est 1.6% et les deux scores sont assez grands, nous pouvons dire que la performance du modèle est très bien Ce qui nous indique que c'est le meilleur modèle obtenu jusqu'à maintenant!

En conclusion, la méthode de features selection avec random forest est la plus efficace pour notre base de données puisque le modèle était de très haute performance.

VI. Comparaison des résultats obtenus avec la modèle svm en réduction de dimensions (avec les 3 méthodes de réductions) et sans réduction

- **Modèle SVM en utilisant comme méthode réduction l'ACP**

Score SVM avant réduction de dimension	Score SVM après réduction de dimension	Score après application du Tuning Hyperparamètres
Score train : 85.3%	Score train : 80.3%	Score train : 80.3%
Score test : 64.2%	Score test : 69.1 %	Score test : 69.1%

Avant la réduction de dimension, notre modèle, avec un train score très élevé de 85.3% et un mauvais test score est overfit, donc il n'est pas efficace et ne peut pas être performant avec des nouvelles données. Ceci s'explique par le fait que notre modèle est trop complexe et l'écart entre le test et l'apprentissage soit important soit 21.1% .Donc il faudrait le simplifier. D'où le fait qu'on ait effectué une réduction de dimension.
Ce qui nous a clairement donné de meilleurs résultats. En effet , pour ce modèle SVM, après avoir appliqué l'ACP, nous voyons une amélioration nette au niveau de l'écart. Pour celui-ci , nous observons un écart de 11.2 % soit environ 2 fois moins que le précédent.

On voit clairement que le test score est nettement meilleur après avoir effectué la réduction de dimension puisqu'il a augmenté de 5%.
Notre modèle à une petite tendance vers l'overfitting.
Pour éviter d'avoir l'overfitting, et de vérifier si notre modèle a besoin d'être améliorée, nous allons utiliser le Tuning Hyperparamètres . Le réglage des hyperparamètres est un élément essentiel du contrôle du comportement d'un modèle d'apprentissage. Si nous n'accordons pas correctement nos hyperparamètres, nos paramètres de modèle estimés produisent des résultats sous-optimaux, car ils ne minimisent pas la fonction de perte. Cela signifie que notre modèle fait plus d'erreurs. En pratique, les indicateurs clés comme la précision ou la matrice de confusion seront moins bons.

Après l'avoir utilisé , nous remarquons que les scores obtenus sont exactement les mêmes que ceux obtenus avant le tuning . Ce qui veut dire que notre modèle ne s'est pas amélioré et qu'il ne peut être amélioré.

- **<u>Modèle SVM en utilisant comme méthode réduction l'ACP à Noyaux</u>**

Score avant réduction de dimension	Score après réduction de dimension	Score après Tuning Hyperparamètres
Score train : 85.3 %	Score train : 87.5%	Score train : 64.1 %
Score test : 64.2%	Score test : 65.4 %	Score test : 63 %

Avant la réduction de dimension ,nous avons un écart entre l'apprentissage et le test de 21.1% ,Alors que après réduction de dimension avec ACP à noyaux l'écart des scores de notre modèle est 22.1% ça veut dire qu'il est également overfit.
Nous remarquons aussi qu'avec la réduction de dimension avec l'ACP noyaux, la performance c'est-à-dire le test score n'a que légèrement augmenté(1.2%).

En terme d'efficacité , nous pouvons clairement déduire que l'ACP est beaucoup plus adapté au modèle de prédiction du SVM pour une réduction de dimension.
Nous allons ici aussi essayer d'éviter l'overfitting , en utilisant le Tuning à nouveau . Celui-ci nous donne un score d'apprentissage de 64.1 % et un score test de 63% . En effet , le score d'apprentissage est très bas , ce qui veut dire que notre modèle n'a pas appris . On dira qu'il est Under fit car les deux scores ne sont pas bons . Alors on en conclut que ce modèle ne peut pas être amélioré.

- **<u>Modèle SVM en utilisant comme méthode de réduction Forêt Aléatoire</u>**

Score avant réduction de dimension	Score après réduction de dimension	Score Tuning Hyperparamètres
Score Train : 85.3%	Score Train: 78.4 %	Score train : 70%
Score Test : 64.2%	Score Test : 72.8 %	Score test : 71.6%

Pour ce modèle , l'écart entre les deux scores est de 5.6% , ce qui est très bien . Ce modèle-ci est également bien , il a de bons résultats et surtout il n'est pas over fit (car l'écart entre les test score et la training score est inférieure à 10).

Nous constatons aussi que la test score comparé au modèle sans réduction de dimension à bien augmenté et il est d'environ 72.8%, ce qui est bien satisfaisant. Ce modèle nous est très utile pour la réduction de dimension.
Nous pouvons en conclure que ces résultats sont excellents et ils sont ni overfitting ni underfitting mais le juste milieu .Donc, il est "parfait".

Afin de perfectionner notre modèle , nous allons prendre le Tuning Hyperparamètres du SVM . Il faut également souligner que ce sont les mêmes hyperparamètres qui seront maintenues.
Celui-ci , nous a donné des scores satisfaisants en termes d'écart entre test score et training score. 70 % comme score d'apprentissage et 71.6% comme test . L'écart entre ces deux scores est presque minime .
mais en comparaison avec le modèle SVM avant de faire le tuning, on pourra dire que les test score et la training score sont diminués!
Donc , nous pouvons en conclure que les résultats de la SVM obtenus avec la méthode Forêt Aléatoire sont les meilleures.

En termes de précision et d'efficacité, nous pouvons clairement dire que le Random Forest, est la méthode de réduction qui nous a apporté les meilleurs résultats. Elle est la méthode la plus adaptée à notre jeu de données avec ces performances remarquables. En deuxième lieu, se trouve la méthode de l'ACP qui avait une bonne performance aussi mais moins élevée en comparaison avec la performance de la méthode Forêt aléatoire.