



---

## Static and Dynamic Hand Gestures

---

*Auteur*

Youva ADDAD

*Enseignant*

Olivier SCHWANDER

# Table des matières

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Analyse et Exploration des données</b>	<b>3</b>
2.1	Analyse globale . . . . .	3
2.2	Analyse des predictors . . . . .	5
2.3	Visualisation a l'aide d'une PCA et d'une TSNE . . . . .	6
<b>3</b>	<b>Protocole expérimental</b>	<b>9</b>
3.1	Pré-Traitemet . . . . .	10
3.2	Découpage des Données . . . . .	10
3.3	Construction des Modèles . . . . .	10
3.4	Sélection de caractéristique . . . . .	10
3.5	Evaluation . . . . .	11
<b>4</b>	<b>Modèles d'apprentissage :</b>	<b>11</b>
4.1	Baseline : . . . . .	11
4.2	Amélioration des baselines . . . . .	12
4.3	Réduction des features . . . . .	13
<b>5</b>	<b>GridSearch et hyperparametres tuning :</b>	<b>13</b>
<b>6</b>	<b>Conclusion</b>	<b>15</b>

# 1 Introduction

L'objectif de ce projet est de prendre en main le dataset UC2017 Static and Dynamic Hand Gestures afin de réaliser un projet de machine learning en entier.

Les données **UC2017 Static and Dynamic Hand** sont obtenues à partir d'un gant de données CyberGlove II et d'un tracker de position Polhemus Liberty avec 6 degrés de liberté (DOF), ils sont utilisés pour capturer la forme, la position et l'orientation de la main au fil du temps. Cet ensemble de données comprend deux types de gestes : les SG et les DG. La différence entre les deux est que les SG sont instantanés, des valeurs des données des capteurs à un instant où la main décrit un geste statique. Les DG sont des séries temporelles de poses et d'orientations de longueur variable qui correspondent à un modèle spécifique dans le temps.

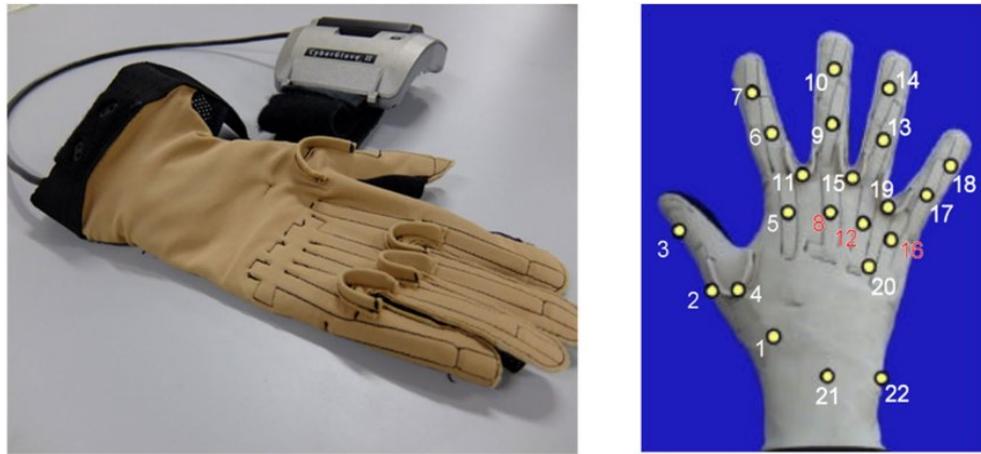


FIGURE 1 – Cyberglove II et emplacement des 22 capteurs.

Le CyberGlove<sup>1</sup> fournit des signaux numériques proportionnels à l'angle de flexion de chacun des 22 capteurs donnant une mesure approximative de la forme de la main. Le capteur du tracker est fixé au gant au poignet et mesure sa position et son orientation par rapport à un cadre fixé au sol. L'orientation est la rotation entre le cadre fixe et le cadre du capteur, donnée comme les angles d'Euler (X,Y,Z) ce qui rajoute 6 autres prédicteur. Mais avec les angles d'Euler, il y'a une dégénérescence en certains points de l'hypersphère, ce qui conduit au problème du blocage de cardan (perte d'un degré de liberté). Pour éviter cela l'utilisation de quatre<sup>2</sup> coordonnées euclidiennes w, x, y et z est nécessaire (Quaternions). Avec cette représentation il y'aura bien 29 predictors.

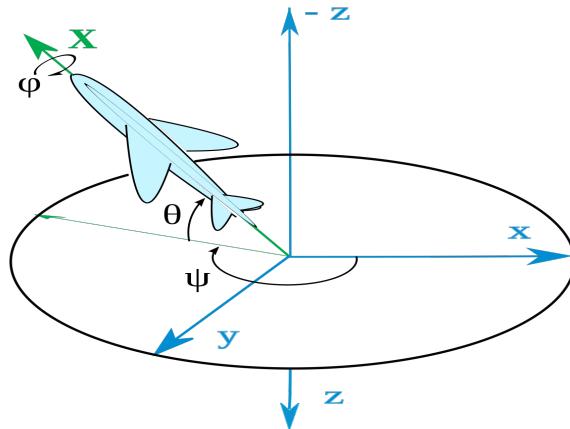


FIGURE 2 – Transformation d'angles d'Euler à quaternions



FIGURE 3 – Représentation des 24 gestes statique

Formellement, nous devons réaliser une tâche de **classification multi-classe** par apprentissage supervisé, où chaque classe correspond à un geste de la figure 3. Nous devons pour cela tout d'abord analyser les données et les interpréter afin de faire des pré-traitement dessus si besoin. Ensuite nous allons passer à la partie entraînement/test des modèles avec les différentes mesures.

À noter que nous allons travailler seulement avec les données statique (UC2017 Static Hand Gestures) .

## 2 Analyse et Exploration des données

### 2.1 Analyse globale

Il est important de visualiser et d'analyser les données acquises pour s'assurer qu'il n'y a pas de problèmes perceptibles.

Notre dataset composée de 24 classes et de 2400 échantillons au total réalisé avec huit sujets pour un total de 100 répétitions par classe et chaque donnée est composée de 29 feature. Tous les sujets sont droitiers et exécutent les gestes de la main gauche.

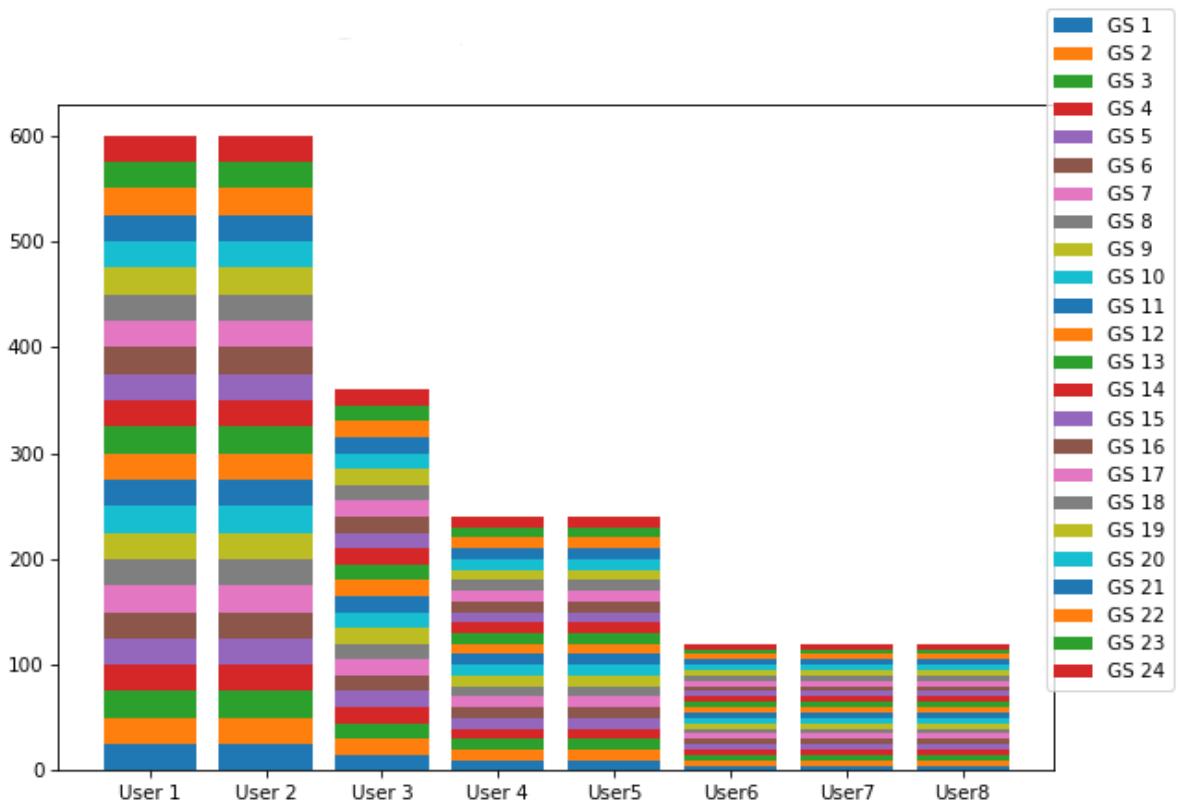


FIGURE 4 – Le nombre de gestes réalisé par chaque utilisateur pour chaque geste

Nous remarquons sur la figure 4 que les données sont parfaitement bien équilibrées et bien reparties entre chaque classe (geste), chaque geste est reproduit par un lot de 100 reparties entre tous les utilisateurs. Par contre il y a un déséquilibre entre le nombre de geste réalisé par les utilisateurs (les utilisateurs 1 et 2 représente 50% de notre ensemble de données).

Pour les predictors de 1 à 22 correspondent bien aux 22 capteurs (angles) du gant en rajoute 7 predictors du tracker qui correspondent à des coordonnées quaternions, de la forme : les 3 positions suivies des quaternions.

## 2.2 Analyse des predictors

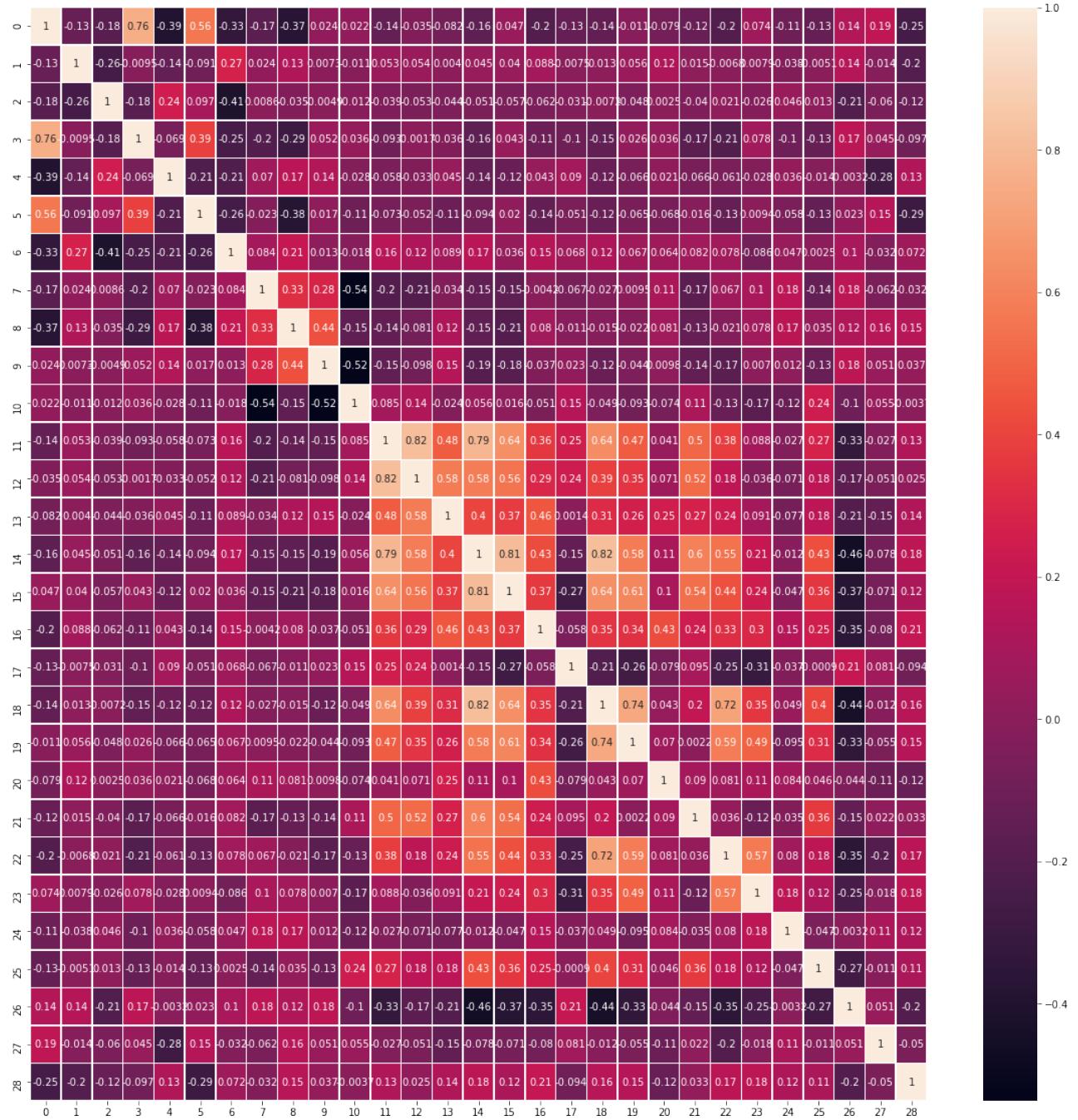


FIGURE 5 – Matrice de corrélation des prédicteurs

Dans cette section nous avons cherché à savoir quel est le lien entre chaque geste et existe-t-il des associations entre features, dans la figure plus la couleur vire vers le blanc plus les caractéristiques associées sont corrélées, nous pouvons remarqué que la partie du centre de 11 à 16 ont peut être un lien entre elle mais c'est dur de tirer des informations exploitable de cette dernière.

## 2.3 Visualisation a l'aide d'une PCA et d'une TSNE

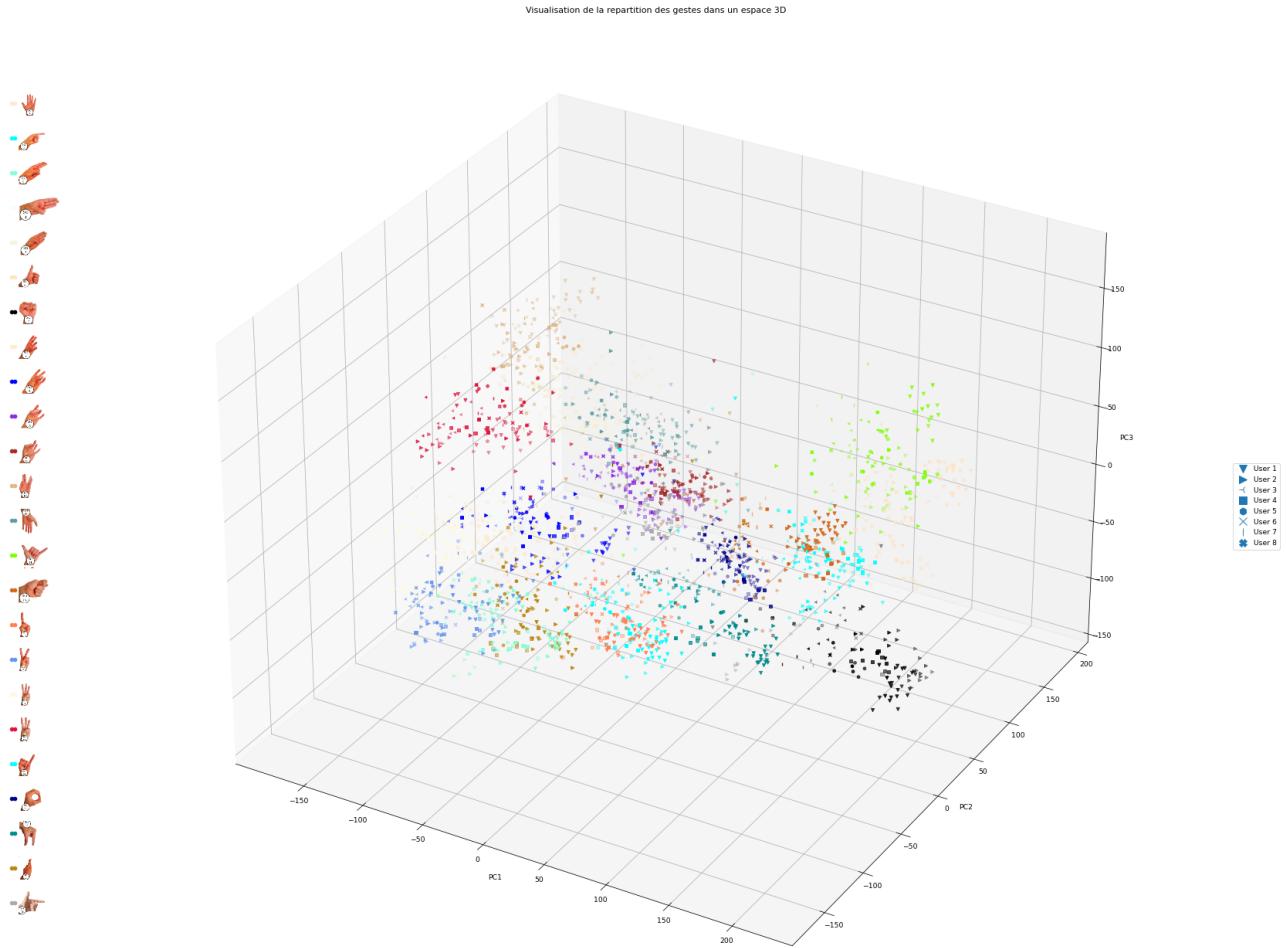


FIGURE 6 – Représentation 3D de tous les gestes et user

Le but de cette figure était de voir si nous pouvions distinguer les gestes entre eux. Si nous regardons l'ensemble de la répartition des données, il est difficile de discerner des clusters qui correspondraient aux classes, ce qui est probablement causer par la dimensionnalité élevée des données. De plus il existe une dispersion intra-classe considérable, en particulier entre les différents utilisateurs figure 4.

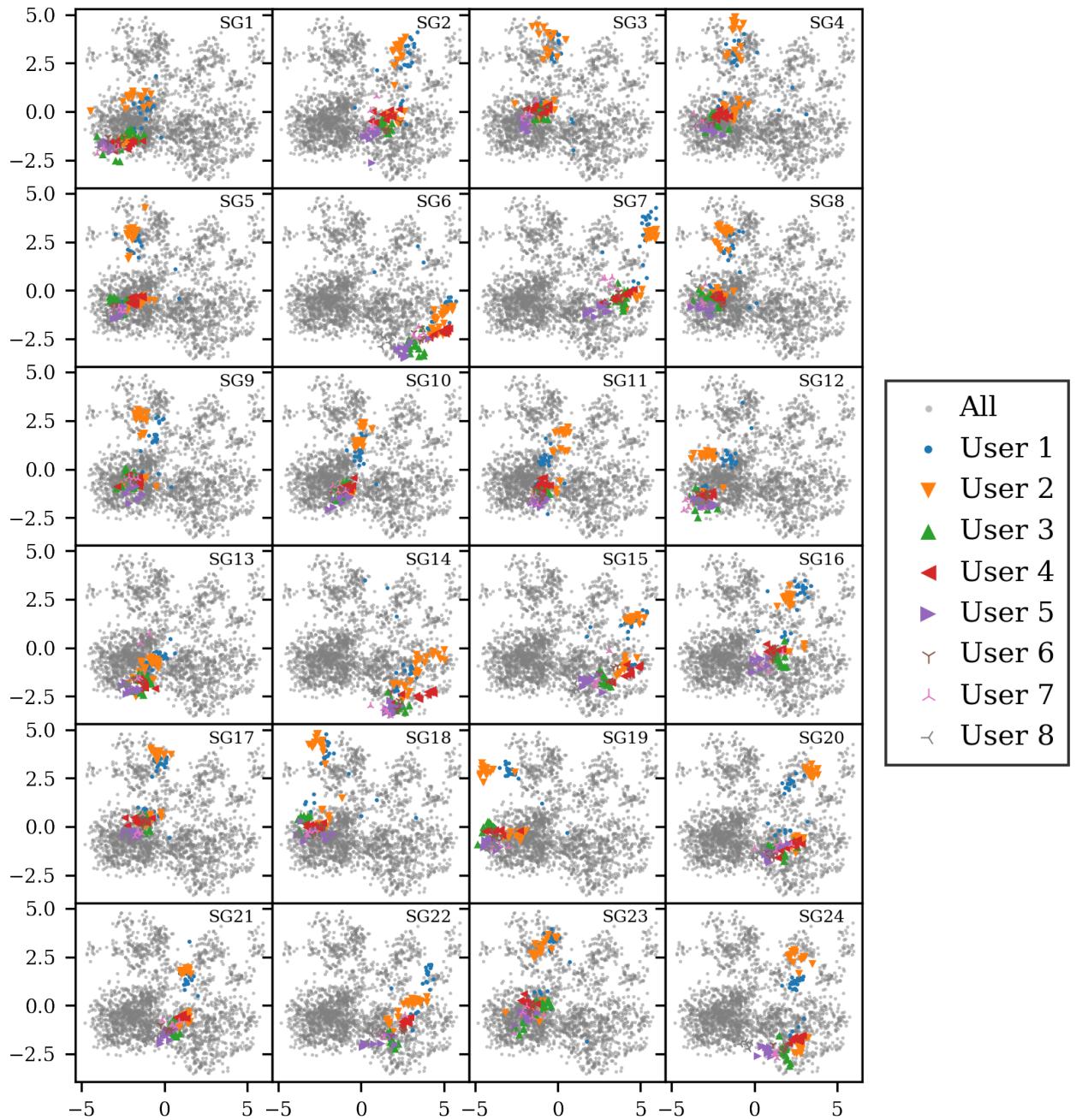


FIGURE 7 – Répartition de chaque user pour un geste avec un PCA

Dans la figure 7 nous avions voulu isoler chaque geste et les users ayant effectué ce geste dans un plan 2D en arrière-plan (la zone grise) correspond à tous les gestes.

De cette figure nous avions ainsi pu remarquer que les gestes proches visuellement sont proches dans l'espace de représentation. Nous distinguons par exemple les groupes suivant : {3, 4, 5}, {8, 9}, {10, 11}, {2, 16, 20} et {18, 19}.

les gestes similaires ont des predictors qui se ressemblent, ce qui ce remarque dans la PCA, ceci pourra potentiellement induire en erreur un algorithme d'apprentissage supervisé en faisant la distinction entre ces gestes.



(a) Geste N°3



(b) Geste N°4



(c) Geste N°5

FIGURE 8 – Premier groupe de similarité dans la PCA



(a) Geste N°8



(b) Geste N°9

FIGURE 9 – Deuxième groupe de similarité dans la PCA



(a) Geste N°10



(b) Geste N°11

FIGURE 10 – Troisième groupe de similarité



(a) Geste N°2



(b) Geste N°16



(c) Geste N°20

FIGURE 11 – Quatrième groupe de similarité dans la PCA



(a) Geste N°18



(b) Geste N°19

FIGURE 12 – Dernier groupe de similarité

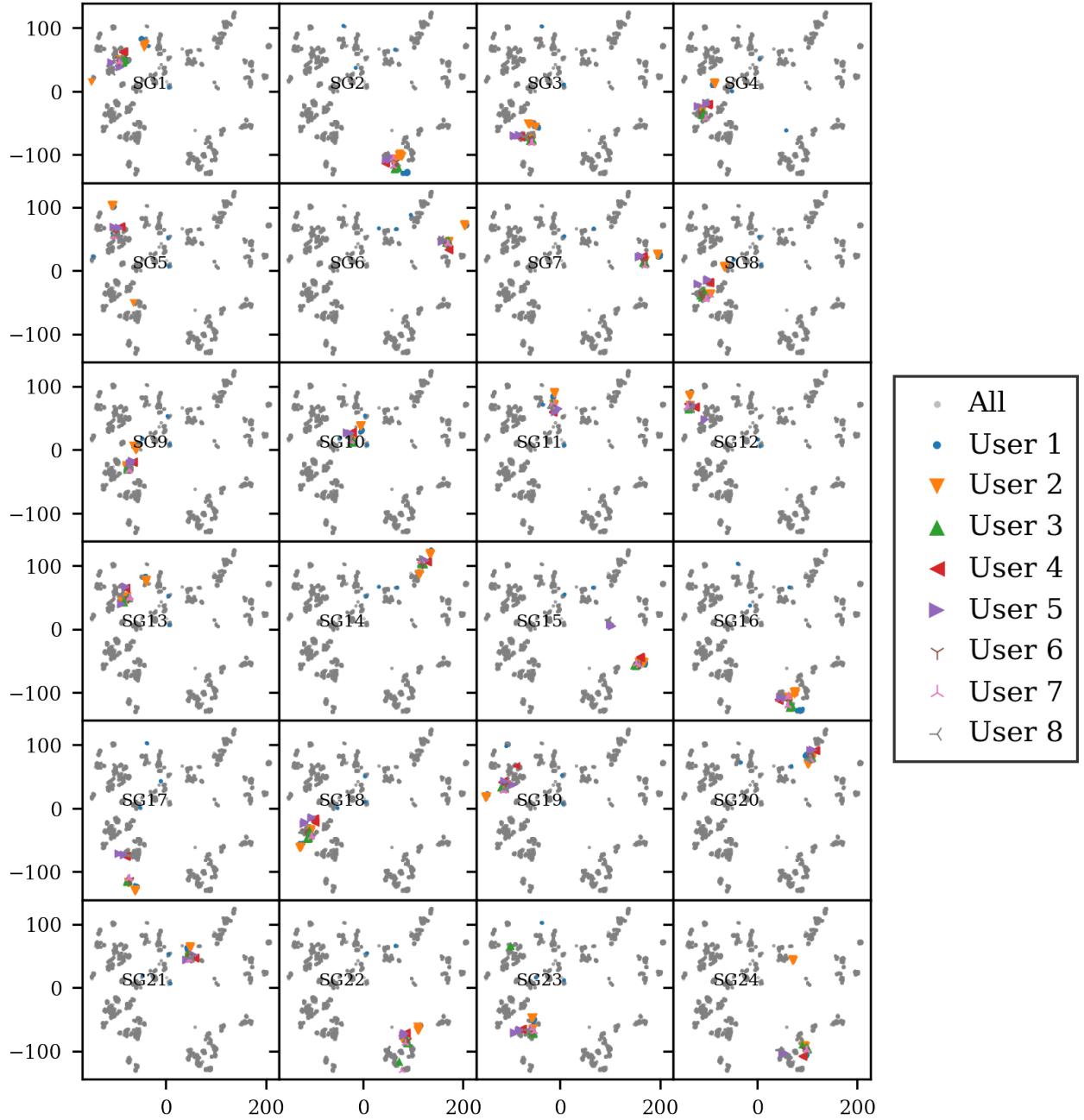


FIGURE 13 – Répartition de chaque user pour un geste avec une TSNE

Nous pouvons faire la même remarque avec une TSNE, les gestes suivant ce ressemble  $\{1, 13\}$ ,  $\{2, 16\}$ ,  $\{8, 18\}$  et  $\{14, 20\}$ , mais nous pouvons remarqué que les gestes  $\{2, 16\}$  sont répartie de la même manière que ce soit pour la TSNE où la PCA. Nous avons fait le choix de rajouté la TSNE afin de voir comment les données ce comporte avec une transformation non linear et comme nous avons 29 predictors le choix de la TSNE était un choix intéressant.

### 3 Protocole expérimental

Dans cette tâche qui est la classification de gestes, aucune analyse mathématique ne peut déterminer si un algorithme d'apprentissage automatique fonctionnera bien. Donc notre but est d'estimer plusieurs modèles afin de trouver celui qui se généralise au mieux sur des données qu'il n'a jamais rencontrées. Des études expérimentales sont donc nécessaires, dans cette section nous allons discuter de :

- **Traitement** : Pre-processing (Normalisation).
- **Paramètres** : hyperparamètres, réglage de paramètres.
- **Selection** : Modèle, feature/
- **Validation et test** : Cross Validation, train test, GridSearch.
- **Métriques et évaluation** : Accuracy, Rappel, Précision, Fmeasure.

### 3.1 Pré-Traitement

Tout d'abord nous avons normalisé les données, en centrant et réduisant les données, le principal avantage de la centration-réduction est de rendre comparables des variables qui ne le seraient pas directement parce qu'elles ont des moyennes et/ou des variances trop différentes, la normalisation pour l'ensemble de test et (possiblement de val) est effectué sur la moyenne et la variance de l'ensemble de train afin de ne pas biaiser le modèle.

$$X_{\text{standard}} = \frac{X - \mu}{\sigma} \quad (1)$$

### 3.2 Découpage des Données

Nous avons découpé nos données en un ensemble de train et de test, comme nous disposons seulement de 2400 exemples cela laisserai moins de données pour l'évaluation et la validation sinon, comme nous avons utilisé un cross validation avec 5 Fold directement sur les données de train cela suffira pour valider le modèle. Nous avons pris un pourcentage de 80% pour le train et de 20% pour le test, de plus afin d'assurer des résultats cohérents et stables, tous les gestes sont présents dans chacun des ensembles avec la même proportion (**stratified**), de train et de test et nous avons en outre tester nos modèles sur des utilisateurs non présents dans les données de train, nous avons utilisé l'utilisateur 6 pour cela, ceci est dans le but de vérifier les performances sur les nouveaux utilisateurs et tester la généralisation (l'utilisateur 6 est donc rajouter en entier sur l'ensemble de test, la proportion devient 75% pour le train et 25% pour le test) .

### 3.3 Construction des Modèles

Étant dans une tâche de classification multi-classe, et Étant donner le peu de données (2400) à notre disposition il sera inutile d'utiliser des méthodes avec des réseaux de neurone. Mais pour pouvoir vraiment comprendre puis améliorer les performances de notre modèle, nous devons d'abord établir une base de référence (une borne inférieure en terme d'accuracy) pour les données dont nous disposons, pour cela nous avons utilisé les baselines suivantes avec découpage train et test sans répartition des utilisateurs :

- Une stratégie **Random** qui consiste à prédire aléatoirement un label, chaque classe à la même probabilité d'être sélectionné.
- Une stratégie **le plus fréquent** on prédit toujours l'étiquette de classe la plus fréquente  $y = \max(\{P(y_i) : i = 1, \dots, n\})$ .
- Une stratégie **Stump** modèle composé d'un arbre de décision à un niveau.
- Une stratégie **Var<sub>i</sub>** où l'apprentissage est effectué sur une seule caractéristique de nos données.

Ayant borné notre problème, nous avons utilisé des algorithmes de supervision plus expressifs, de part la suite nous allons prendre le modèle offrons la meilleur généralisation :

- Gaussian Naive Bayes (NB)
- Decision Tree
- K-Nearest Neighbors (KNN)
- Support Vector Machines (SVM)
- Random Forests
- GradientBoostingClassifier

### 3.4 Sélection de caractéristique

Dans cette partie nous avons jugé que les features fournissent par le gant glove c-à-d les angles sont tous pertinents, le gant fournit des signaux numériques proportionnels à l'angle de flexion de chacun des 22 capteurs attachés des articulations de la main, donc chacun des angles est pertinent pour la façon dont un geste est effectué. Nous avons aussi jugé que les positions relatives des mains n'était pas un critère important, de même pour les quaternions nous avons jugé qu'ils sont importants pour la bonne classification.

### 3.5 Evaluation

Pour l'évaluation de nos modèles, nous avons utilisé l'accuracy comme métrique principale, en effet nous sommes sur un cas d'une bonne répartition des données (shuffle + stratify sur les données), l'accuracy dans ce cas sera la plus significative. Par ailleurs comme souligner précédemment nous avons utilisé une cross validation avec 5 folds afin de nous assurer de la bonne cohérence de l'accuracy.

## 4 Modèles d'apprentissage :

### 4.1 Baseline :

Nous avons tester nos données sur les modèles baseline cité précédemment, le modèles aléatoire fait a peu près 1/24 de bonne classification égal a l'algorithme qui prend le label le plus fréquent (most frequent). Dans le modèle a une features,nous avons appris un arbre de décision sur cette dernière, le modèle réalise une accuracy de  $\sim 30\%$  avec le predictor 13. En ce qui concerne le stump il réalise une accuracy en test de 5.6% devant les random et les most frequent mais moins bon que le modèle a une variable, dans la figure 15 le X[12] correspond a notre predictor numéro 12. Nous avons donc borné notre problème, nous savons que nous nous pouvons pas faire pire que 23% de bonne classification.

Les figures suivantes représentent des comparaisons entre les modèles, la figure 14 et 15 représente l'accuracy des variables sur le modéle variable unique et l'arbre stump respectivement :

Model	Train	Test
Random	4.32	6.46
Most Frequent	4.53	2.71
Stump	9.0	5.6
Variable Unique	33.2	22.9

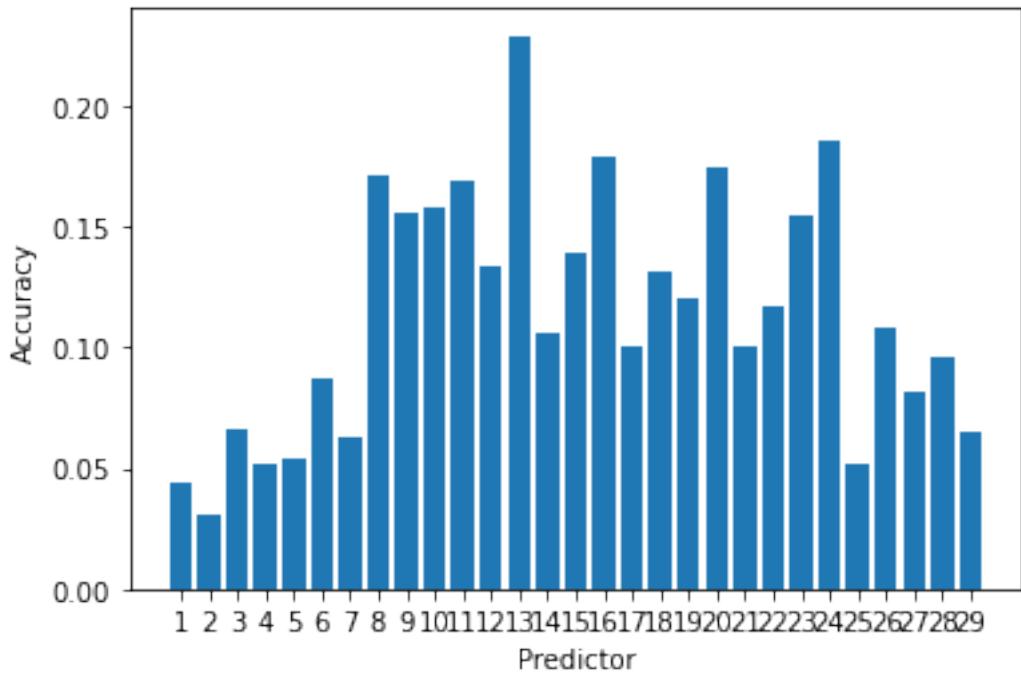


FIGURE 14 – Accuracy d'une variable avec arbre de décision

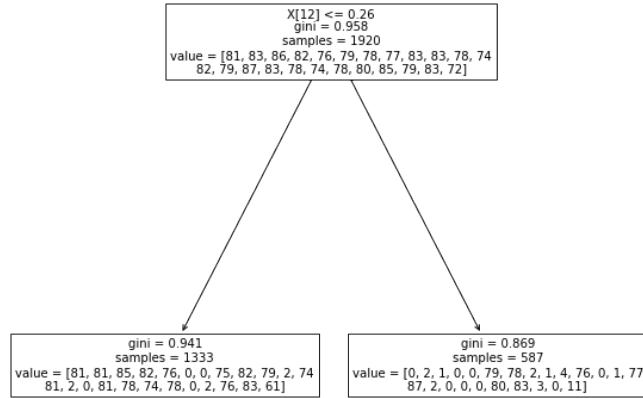


FIGURE 15 – Visualisation de l’arbre stump

## 4.2 Amélioration des baselines

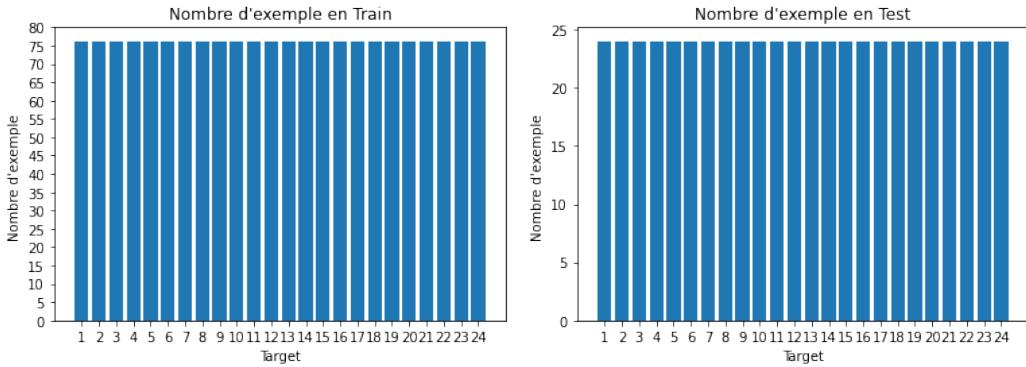
Nous avons entraîné des modèles plus expressifs cité précédemment sur l’ensemble des predictors (les 29 features), il s’agit ici de voir à quel point les predictors sont importants, mais sachant que plus de features est souvent synonyme d’overfitting nous avons donc pour but d’essaier de sur-apprendre afin de borné supérieurement l’accuracy en train.

Dans tous les cas tous les modèles utilisent une cross validation de 5 Fold, un ensemble d’évaluation en test et en train, la stratégie de la présence d’un utilisateur (user 6) sur le test et pas sur le train est ainsi utilisé aussi, enfin les données seront stratified (il y aura le même nombre de target partout).

Pour les arbres de décision et les random forest nous ne mettons aucune restriction sur la profondeur des arbres de même pour les méthodes de boosting. Les KNN utilisent un  $k=5$  et 3, le SVM un poly kernel de degré 2 où un rbf. Afin de ne pas biaiser la validation la normalisation est effectuée à chaque tour du boucle du KFold (les données de test sont normalisées avec la moyenne et la variance des données de train).

Model	Accuracy (%)		
	Train (Sans user 6)	Validation (5 Folds)	Test (Avec le user 6)
Naive Bayes	92.32	90.02	90.45
3NN	96.71	92.65	93.22
5NN	95.39	89.42	91.67
Arbre de décision	100	90.73	88.19
Gradient Boosting	100	92.32	92.70
Random Forest	100	95.88	95.13
SVM (kernel=poly, degree=2)	98.73	95.12	95.14
SVM (kernel=rbf)	98.95	95.06	95.65

Nous pouvons statuer que tous les modèles font de bonne performances, les arbres de décisions, gradient boosting ainsi que les random forest font du surapprentissage, une zéro erreur en train mais considérable en validation et test. 3NN et meilleur que 5NN en performance, le SVM avec un kernel rbf fait la meilleure performance en test ce qui nous prouve qu’il généralise bien sur des nouveaux utilisateurs.



(a) Nombre d'exemple dans le train après le split et le retrait du User 6, 1824 Exemples 76 pour chaque target  
(b) Nombre d'exemple dans le test après le split et le rajout du User 6, 576 Exemples 24 pour chaque target

FIGURE 16 – Nombre d'exemple présent en train et en test

### 4.3 Réduction des features

Maintenant que nous avons une borne supérieure de notre tâche, nous allons réduire le nombre de features afin d'avoir un gain de temps, nous avons souligné le fait que la position de la main n'était pas pertinente dans notre cas, nous allons donc enlever les 3 premiers predictors, nous allons de plus utilisé un SVM avec un kernel linear et une pénalité "l1" ce qui conduit à des vecteurs des poids qui sont sparsé, nous allons d'abord entraîné le modèle précédent avec les mêmes condition que précédemment, ensuite nous allons réduire les features.

Model	Accuracy (%)		
	Train (Sans user 6)	Validation (5 Folds)	Test (Avec le user 6)
Linear SVM	96.43	93.47	94.61
Naive Bayes	91.94	89.36	89.93
3NN	97.15	93.86	93.92
5NN	95.83	92.98	93.22
Arbre de décision	100	90.13	88.19
Gradient Boosting	100	92.54	91.66
Random Forest	100	95.77	94.96
SVM (kernel=poly, degree=2)	98.51	94.79	94.61
SVM (kernel=rbf)	98.68	95.06	94.79

Nous avons certes un gain de temps, mais les performances ne sont pas meilleur que précédemment donc il y aura un dilemme temps versus performances, mais dans l'ensemble les performance sont presque identique, par ailleurs les performances restent bonnes et cette fois le Random forest fait mieux que le SVM en test.

## 5 GridSearch et hyperparametres tuning :

Comme les modèles ayant eu les meilleures performances sont le Random Forest avec 100 arbres dans la forêt ainsi que le SVM avec un kernel 'rbf' coupler a un C=1, nous avons proposé d'optimiser ces deux derniers. Dans le cas du SVM nous utilisons toujours le kernel rbf et nous optimisons sur C avec les valeurs [0.001,0.1,0.5,1,2,...,50] avec toutes les features (les 29 predictors utilisé ici sauf mention contraire). Dans le cas du Random Forest nous utilisons [10,20,...,150] pour le nombre d'arbre et [0,10,...60,None] pour la longueur des arbres (les 29 predictors utilisé ici sauf mention contraire).

Model	Accuracy (%)		
	Train (Sans user 6)	Validation (5 Folds)	Test (Avec le user 6)
SVM (kernel=rbf, C=7 )	99.61	96.32	96.18
Random Forest (max_d = 30,n_e = 130)	100	95.77	95.48
SVM (kernel=rbf, C=11)	99.67	96.10	95.83
Random Forest(max_d = 40,n_e = 110)	100	95.72	95.13

Les deux premières performances sont obtenu avec les 29 predictors , n\_e fait référence aux nombre d'arbre et max\_d fait référence a la profondeur maximum de l'arbre, les 2 suivantes correspondent a la performances

obtenu sur 26 predictors en ayant enlever les positions, entre paranthése les meilleurs modéle.

Les performances sont quasiment les mêmes en test, elle ce chevauche (29 predictors vs 26 predictors), malgré cela le meilleur modéle est le SVM avec un kernel rbf et un C égal a 7.

	<b>precision</b>	<b>recall</b>	<b>f1-score</b>	<b>support</b>
1	<b>0.96</b>	<b>1.00</b>	<b>0.98</b>	23
2	<b>0.88</b>	<b>1.00</b>	<b>0.93</b>	21
3	<b>0.96</b>	<b>1.00</b>	<b>0.98</b>	23
4	<b>1.00</b>	<b>0.92</b>	<b>0.96</b>	26
5	<b>1.00</b>	<b>0.96</b>	<b>0.98</b>	25
6	<b>0.92</b>	<b>1.00</b>	<b>0.96</b>	22
7	<b>0.96</b>	<b>0.92</b>	<b>0.94</b>	25
8	<b>0.88</b>	<b>0.95</b>	<b>0.91</b>	22
9	<b>0.96</b>	<b>0.88</b>	<b>0.92</b>	26
10	<b>0.92</b>	<b>0.92</b>	<b>0.92</b>	24
11	<b>0.92</b>	<b>1.00</b>	<b>0.96</b>	22
12	<b>1.00</b>	<b>0.96</b>	<b>0.98</b>	25
13	<b>0.96</b>	<b>0.96</b>	<b>0.96</b>	24
14	<b>1.00</b>	<b>1.00</b>	<b>1.00</b>	24
15	<b>0.96</b>	<b>1.00</b>	<b>0.98</b>	23
16	<b>1.00</b>	<b>0.92</b>	<b>0.96</b>	26
17	<b>1.00</b>	<b>0.92</b>	<b>0.96</b>	26
18	<b>0.96</b>	<b>0.88</b>	<b>0.92</b>	26
19	<b>0.96</b>	<b>1.00</b>	<b>0.98</b>	23
20	<b>1.00</b>	<b>1.00</b>	<b>1.00</b>	24
21	<b>1.00</b>	<b>1.00</b>	<b>1.00</b>	24
22	<b>0.96</b>	<b>0.96</b>	<b>0.96</b>	24
23	<b>0.96</b>	<b>1.00</b>	<b>0.98</b>	23
24	<b>1.00</b>	<b>0.96</b>	<b>0.98</b>	25
<b>accuracy</b>			<b>0.96</b>	576
<b>macro avg</b>	<b>0.96</b>	<b>0.96</b>	<b>0.96</b>	576
<b>weighted avg</b>	<b>0.96</b>	<b>0.96</b>	<b>0.96</b>	576

FIGURE 17 – Rapport sur le SVM kernel='rbf' et C=7 avec les 29 predictors

	<b>precision</b>	<b>recall</b>	<b>f1-score</b>	<b>support</b>
1	<b>0.92</b>	<b>0.96</b>	<b>0.94</b>	23
2	<b>0.88</b>	<b>0.95</b>	<b>0.91</b>	22
3	<b>0.96</b>	<b>1.00</b>	<b>0.98</b>	23
4	<b>1.00</b>	<b>0.86</b>	<b>0.92</b>	28
5	<b>1.00</b>	<b>0.96</b>	<b>0.98</b>	25
6	<b>0.92</b>	<b>1.00</b>	<b>0.96</b>	22
7	<b>1.00</b>	<b>0.96</b>	<b>0.98</b>	25
8	<b>0.83</b>	<b>0.87</b>	<b>0.85</b>	23
9	<b>0.96</b>	<b>1.00</b>	<b>0.98</b>	23
10	<b>0.92</b>	<b>0.96</b>	<b>0.94</b>	23
11	<b>0.92</b>	<b>0.92</b>	<b>0.92</b>	24
12	<b>1.00</b>	<b>1.00</b>	<b>1.00</b>	24
13	<b>0.96</b>	<b>0.92</b>	<b>0.94</b>	25
14	<b>1.00</b>	<b>0.96</b>	<b>0.98</b>	25
15	<b>1.00</b>	<b>1.00</b>	<b>1.00</b>	24
16	<b>0.96</b>	<b>0.96</b>	<b>0.96</b>	24
17	<b>1.00</b>	<b>0.96</b>	<b>0.98</b>	25
18	<b>0.92</b>	<b>0.96</b>	<b>0.94</b>	23
19	<b>0.96</b>	<b>1.00</b>	<b>0.98</b>	23
20	<b>1.00</b>	<b>0.89</b>	<b>0.94</b>	27
21	<b>1.00</b>	<b>1.00</b>	<b>1.00</b>	24
22	<b>0.96</b>	<b>1.00</b>	<b>0.98</b>	23
23	<b>0.96</b>	<b>1.00</b>	<b>0.98</b>	23
24	<b>1.00</b>	<b>0.96</b>	<b>0.98</b>	25
<b>accuracy</b>			<b>0.96</b>	<b>576</b>
<b>macro avg</b>	<b>0.96</b>	<b>0.96</b>	<b>0.96</b>	<b>576</b>
<b>weighted avg</b>	<b>0.96</b>	<b>0.96</b>	<b>0.96</b>	<b>576</b>

FIGURE 18 – Rapport sur le Random Forest avec les 29 predictors

## 6 Conclusion

Nous concluons nos expérimentations en disons tout d’abord que le meilleur modèle ce généralisant le mieux avec des utilisateurs qu’il n’a jamais rencontrer est le SVM avec un kernel rbf qui obtient un score en accuracy de 96.18% en test, le random forest est non loin avec 95.48%. La réduction de la dimensionnalité a certes pour effet de reduire le temps d’excution mais cela ce répercute sur les performances, il y aura donc un dilemme performances versus temps d’exécution.