Ecole Nationale d'Ingénieurs de Sfax, Université de Sfax

Exposé Projet IA

Reconnaissance des gestes de la main

Equipe :

Mariam OMRANE, Dorra CHAARI, Khalil KOLSI

Mars 2019

Plan

- Context et Objectif
- Description Base de données
- Résultats
- Discussion

Context

Notre projet était motivé par la capabilité des algorithmes d'intelligence artificielle à reconnaître et classer certains types de gestes de main en se basant sur une base de donnée bien définie et bien organisée.

Context

Les gestes sont les suivants :

- Zoom in: deux mains s'éloignant horizontalement
- Zoom out: deux mains qui se rapprochent horizontalement
- Move left: seule main se déplaçant à gauche
- Move right: seule main se déplaçant à droite
- Move down: seule main en descendant
- Move up: seule main en remontant

Objectif

L’objectif de notre projet est de savoir lequel des algorithmes d'intelligence artificielle qu’on a étudié et le plus précis et présente le moins d'erreurs pour l'utiliser à manipuler une lecteure de vidéo en reconnaissant les gestes des mains en temps réel et affecter une action à chaque geste.

Objectif

Puisque on cherche à distinguer les différentes gestes du main et les classifier , on a choisi d'utiliser les différents algorithmes de classification pour faire de la prédiction dans l'apprentissage supervisé et l'algorithme K-means pour faire de la prédiction dans l'apprentissage non supervisé.

Description Base de données

La base de données nommée **Hand_Gesture_Recognition_Dataset** contient **300 échantillons** de **6 gestes**.

Chaque geste est effectué chacun à 10 reprises par 5 volontaires, aboutissant à **50 échantillons pour chaque geste**.

Description Base de données

Un geste consiste en une séquence d'images consécutives.

On considère **la moyenne** de la somme d'une séquence de 30 images comme une seule entrée.

Les features sont basées sur les positions du **bout des doigts**, le **centroïde de la paume** de la main et sa valeur de **profondeur**.

Description Base de données

On a donc **26 features** :

- Positions du centroïde gauche (LC_X et LC_Y)
- Positions du centroïde droit (RC_X et RC_Y)
- Positions des doigts gauches numérotées dans le sens des aiguilles d'une montre (LF1_X, LF2_X, LF3_X, LF4_X, LF5_X et LF1_Y, LF2_Y, LF3_Y, LF4_Y, LF5_Y)
- Positions des doigts droits numérotées dans le sens des aiguilles d'une montre (LR1_X, LR2_X, LR3_X, LR4_X, LR5_X et LR1_Y, LR2_Y, LR3_Y, LR4_Y, LR5_Y)
- Profondeur du centroïde droit (RCD)
- Profondeur du centroïde gauche (LCD)

Apprentissage supervisé

- Préparation des données:

```
data=dataset.iloc[:,:-2].values
target=dataset.iloc[:,-2].values
```

- Séparer training/testing set utilisant train_test_split :

```
data=dataset.iloc[:,:-2].values
target=dataset.iloc[:,-2].values
```

```
from sklearn.model_selection import train_test_split
data_train,data_test,target_train,target_test=train_test_split(data,target,test_size=0.2,random_state=0)
```

```
print("data_train shape: {}".format(data_train.shape))
print("target_train shape: {}".format(target_train.shape))
```

```
data_train shape: (240, 27)
target_train shape: (240,)
```

```
print("data_test shape: {}".format(data_test.shape))
print("target_test shape: {}".format(target_test.shape))
```

```
data_test shape: (60, 27)
target_test shape: (60,)
```

Classifieur Kppv

❖ Créer le classifieur 3-NN:

```
from sklearn import neighbors
knn = neighbors.KNeighborsClassifier(n_neighbors=3)
```

❖ Évaluation de performance avec la méthode mean_squared_error:

```
from sklearn import metrics
print("Mean Squared Error:",metrics.mean_squared_error(target_test,y_pred))
```

```
Mean Squared Error: 0.0
```

❖ Évaluation de performance avec la méthode score:

```
print(knn.score(data_test,target_test))
```

```
1.0
```

❖ Matrice de confusion:

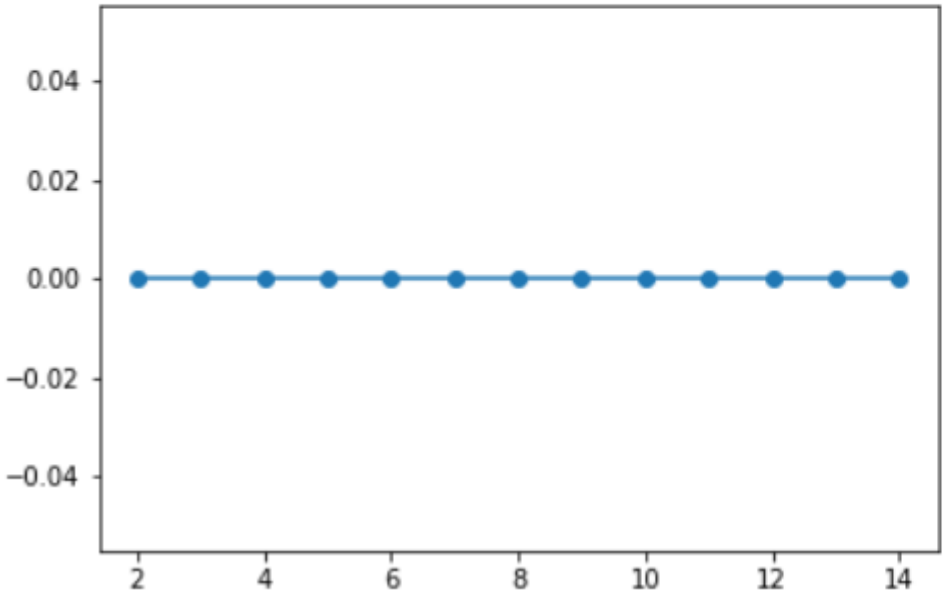
```
print(metrics.confusion_matrix(target_test,target_pred))
```

```
[[ 9  0  0  0  0  0]
 [ 0 11  0  0  0  0]
 [ 0  0  9  0  0  0]
 [ 0  0  0 11  0  0]
 [ 0  0  0  0 13  0]
 [ 0  0  0  0  0  7]]
```

Classifieur Kppv

- Tester le modèle pour tous les k de 2 à 15

```
errors = []
for k in range(2,15):
    knn = neighbors.KNeighborsClassifier(k)
    errors.append(100*(1 - knn.fit(data_train,target_train).score(data_test,target_test)))
plt.plot(range(2,15), errors, 'o-')
plt.show()
```



Classifieur Naive Bayes

- Créer le classifieur Gaussien:

```
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
```

- Prédiction des classes des éléments de test avec la fonction gnb.predict()

```
y_pred= gnb.predict(data_test)
y_pred
```

```
array([4, 3, 0, 4, 4, 2, 4, 4, 1, 2, 5, 0, 3, 4, 0, 1, 5, 1, 4, 3, 3, 2,
       2, 2, 4, 2, 3, 5, 4, 5, 1, 1, 0, 2, 4, 3, 3, 3, 5, 0, 4, 1, 1, 0,
       1, 2, 4, 1, 0, 0, 1, 4, 3, 2, 1, 5, 3, 0, 5, 3], dtype=int64)
```

- Évaluation de performance avec la méthode accuracy_score:

```
from sklearn import metrics
from sklearn.metrics import accuracy_score
print('accuracy = %f' %accuracy_score(target_test,target_pred))
```

```
accuracy = 1.000000
```

Classifieur Naive Bayes

- Créer le classifieur de Bernoulli:

```
from sklearn.naive_bayes import BernoulliNB
bnb = BernoulliNB()
```

- Prédiction des classes des éléments de test avec la fonction bnb.predict()

```
y_pred1= bnb.predict(data_test)
```

```
y_pred1
```

```
array([4, 5, 0, 4, 3, 2, 4, 4, 1, 2, 5, 0, 3, 4, 0, 1, 5, 1, 4, 3, 3, 2,
       2, 2, 4, 2, 3, 5, 4, 5, 1, 3, 0, 2, 4, 3, 3, 3, 5, 0, 4, 1, 3, 0,
       3, 2, 4, 1, 0, 0, 3, 4, 3, 2, 1, 5, 3, 0, 5, 3], dtype=int64)
```

- Évaluation de performance avec la méthode accuracy_score:

```
from sklearn import metrics
from sklearn.metrics import accuracy_score
print('accuracy = %f' %accuracy_score(target_test,y_pred1))
```

```
accuracy = 0.900000
```

Classifieur Naive Bayes

- Comparaison des rapports de classification des deux classifieurs

```
print(metrics.classification_report(target_test,y_pred))
```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	9
1	1.00	1.00	1.00	11
2	1.00	1.00	1.00	9
3	1.00	1.00	1.00	11
4	1.00	1.00	1.00	13
5	1.00	1.00	1.00	7
avg / total	1.00	1.00	1.00	60

Classifieur Gaussien

```
print(metrics.classification_report(target_test,y_pred1))
```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	9
1	1.00	0.64	0.78	11
2	1.00	1.00	1.00	9
3	0.67	0.91	0.77	11
4	1.00	0.92	0.96	13
5	0.88	1.00	0.93	7
avg / total	0.92	0.90	0.90	60

Classifieur de Bernoulli

=> On peut remarquer que le classifieur Gaussien est plus robuste que le classifieur de Bernoulli

Classifieur Naive Bayes

- Comparaison des matrices de confusion des deux classifieurs

```
print(metrics.confusion_matrix(target_test,y_pred))
```

```
[[ 9  0  0  0  0  0]
 [ 0 11  0  0  0  0]
 [ 0  0  9  0  0  0]
 [ 0  0  0 11  0  0]
 [ 0  0  0  0 13  0]
 [ 0  0  0  0  0  7]]
```

Classifieur Gaussien

```
[[ 9  0  0  0  0  0]
 [ 0  7  0  4  0  0]
 [ 0  0  9  0  0  0]
 [ 0  0  0 10  0  1]
 [ 0  0  0  1 12  0]
 [ 0  0  0  0  0  7]]
```

Classifieur de Bernoulli

Classifieur Naive Bayes

- Évaluation de performance avec la méthode mean_squared_error:

```
print("Mean Squared Error:",metrics.mean_squared_error(target_test,y_pred))
```

```
Mean Squared Error: 0.0
```

Classifieur Gaussien

```
print("Mean Squared Error:",metrics.mean_squared_error(target_test,y_pred1))
```

```
Mean Squared Error: 0.35
```

Classifieur de Bernoulli

Remarque: le classifieurs Multinomial Naïve Bayes est non applicable sur notre base puisqu'elle contient des valeurs négatives.

Classifieur SVM

❖ Créer le classifieur SVM:

```
from sklearn import svm
model = svm.SVC(kernel='linear')
```

❖ Évaluation de performance avec la méthode mean_squared_error:

```
from sklearn import metrics
print("Mean Squared Error:",metrics.mean_squared_error(target_test,y_pred))
```

```
Mean Squared Error: 0.0
```

❖ Évaluation de performance avec la méthode score:

```
print(model.score(data_test,target_test))
```

```
1.0
```

❖ Matrice de confusion:

```
print(metrics.confusion_matrix(target_test,target_pred))
```

```
[[ 9  0  0  0  0  0]
 [ 0 11  0  0  0  0]
 [ 0  0  9  0  0  0]
 [ 0  0  0 11  0  0]
 [ 0  0  0  0 13  0]
 [ 0  0  0  0  0  7]]
```

Reseau de Neurones

❖ Pré-traitement sur la base de données

```
sc=StandardScaler()
sc.fit(data_train)
```

```
data_train_std=sc.transform(data_train)
data_test_std=sc.transform(data_test)
```

❖ Création d'un Perceptron

```
ppn=Perceptron(max_iter=40,eta0=0.1,random_state=0)
ppn.fit(data_train_std,target_train)
```

```
Perceptron(alpha=0.0001, class_weight=None, eta0=0.1, fit_intercept=True,
      max_iter=40, n_iter=None, n_jobs=1, penalty=None, random_state=0,
      shuffle=True, tol=None, verbose=0, warm_start=False)
```

Reseau de Neurones

❖ Prédiction des classes des éléments de test avec la fonction pnb.predict()

```
y_pred=ppn.predict(data_test_std)
y_pred
```

```
array([5, 1, 3, 2, 1, 5, 3, 5, 3, 3, 4, 2, 2, 3, 5, 4, 2, 5, 3, 0, 3, 1,
       3, 5, 0, 1, 3, 1, 3, 5, 0, 5, 0, 0, 5, 4, 2, 1, 0, 5, 5, 4, 2, 0,
       4, 0, 2, 1, 1, 1, 5, 0, 4, 1, 4, 5, 5, 5, 0, 1, 4, 2, 4, 4, 0, 0,
       5, 5, 3, 0, 2, 5, 4, 4, 5, 4, 4, 2, 1, 2, 0, 1, 2, 3, 1, 1, 0, 0,
       3, 0], dtype=int64)
```

❖ Évaluation de performance avec la méthode accuracy_score:

```
print("Accuracy:%.2f"%accuracy_score(target_test,y_pred))
```

```
Accuracy:0.97
```

❖ Évaluation de performance avec la méthode mean_squared_error:

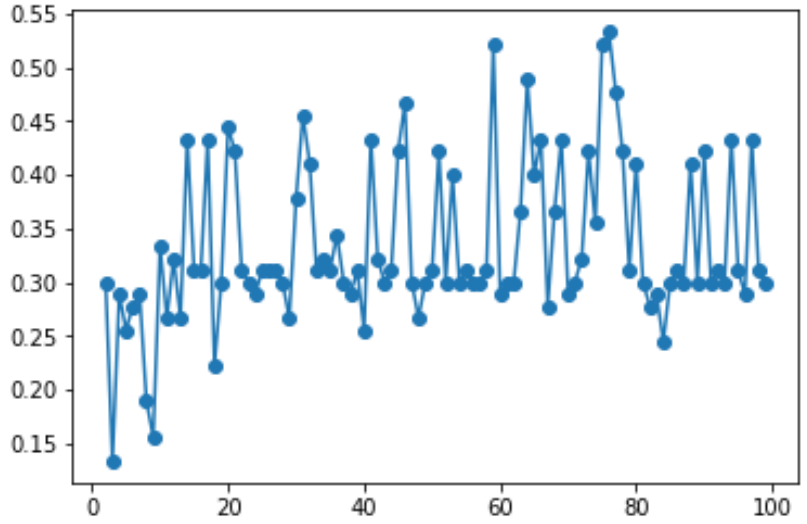
```
from sklearn import metrics
print("Mean Squared Error:",metrics.mean_squared_error(target_test,y_pred))
```

```
Mean Squared Error: 0.1
```

Reseau de Neurones

- Création et test de score avec le réseau de neurones:

```
hiddenLayer=[]
for k in range(2,100):
    mlp=MLPClassifier(hidden_layer_sizes=(k),solver="sgd",learning_rate_init=0.01,max_iter=500)
    mlp.fit(data_train,target_train)
    hiddenLayer.append(mlp.score(data_test,target_test))
plt.plot(range(2,100),hiddenLayer,'o-')
plt.show()
```



Apprentissage non supervisé

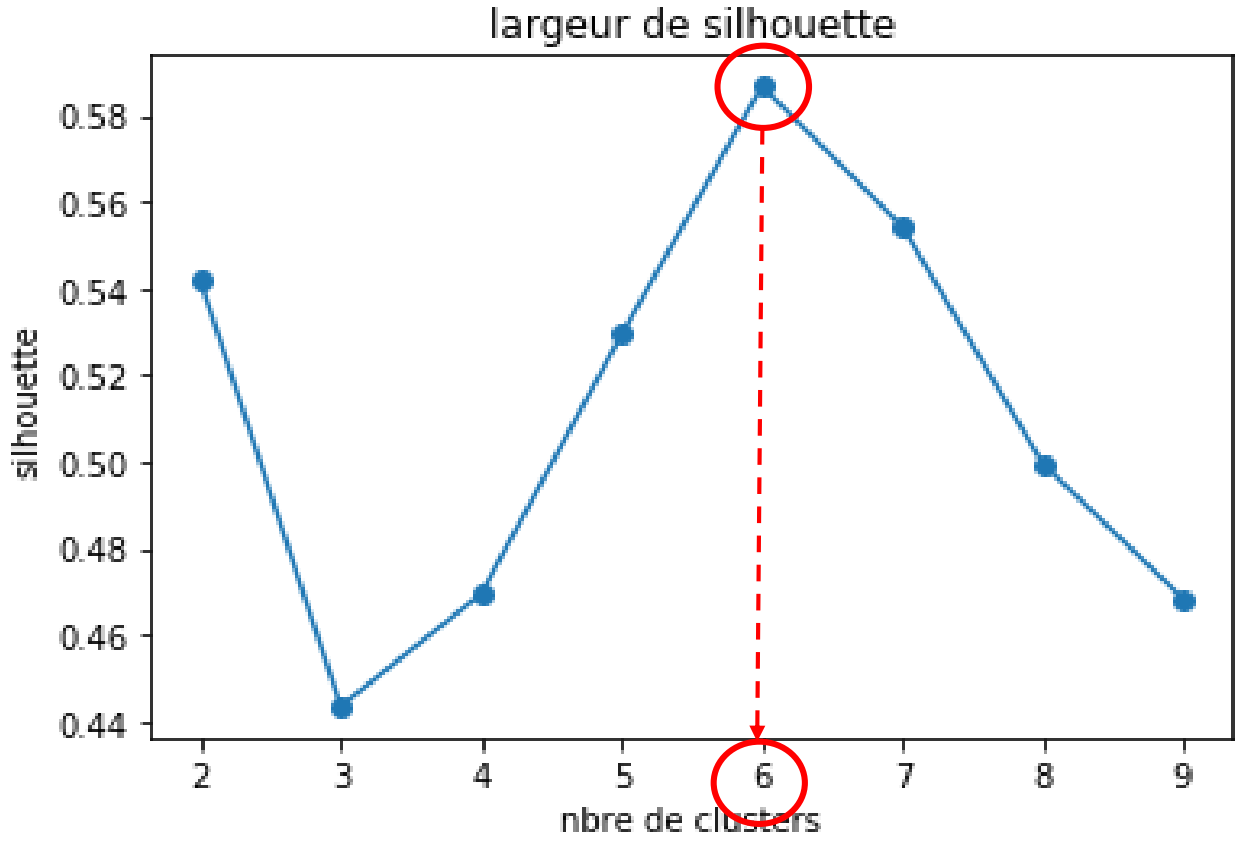
- Enlever les deux colonnes Gesture_Class et Gesture_Name:

```
data=dataset.iloc[:,:-2].values
data
```

```
array([[  0, -20,  -6, ...,   3,   3,   5],
       [  1, -17,  -3, ...,   7,  -2,   3],
       [  2, -12,   5, ...,   2,   2,  -1],
       ...,
       [297,  12, -11, ...,   0,   0,   0],
       [298,   7, -10, ...,   0,   0,   0],
       [299,  11, -16, ...,   0,   0,   0]], dtype=int64)
```

Classifieur Kmeans

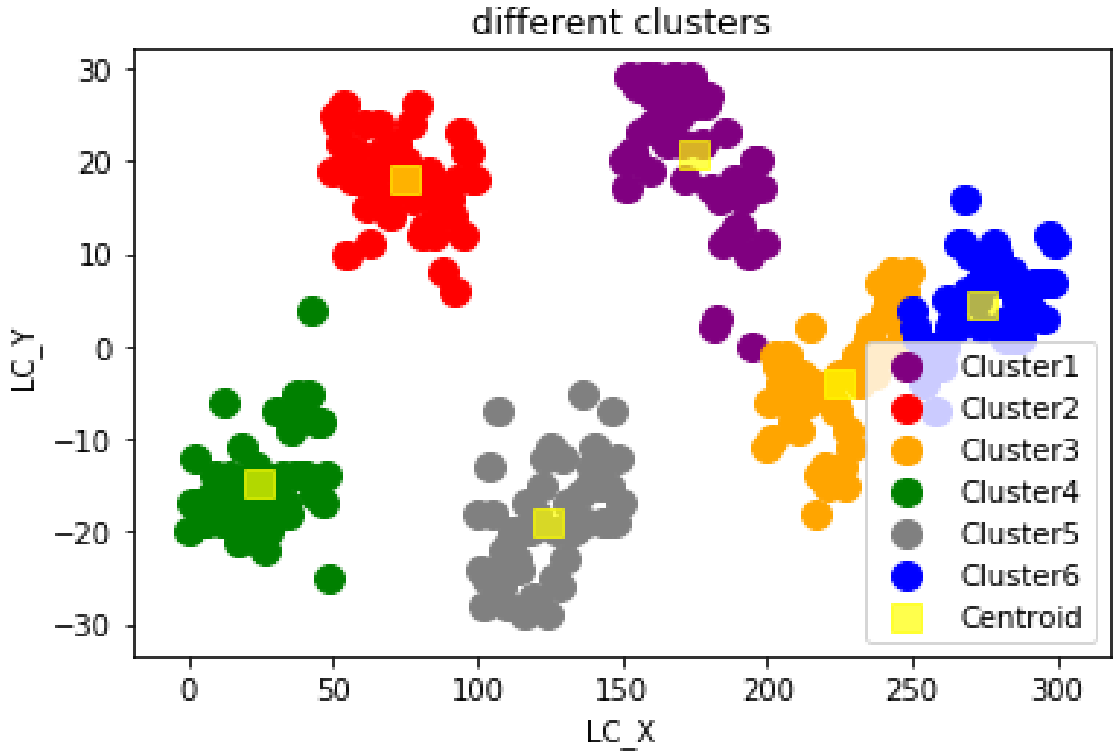
- La recherche du nombre optimal de clusters avec la méthode silhouette_score:



Classifieur Kmeans

- Kmeans clustering: Ajustement de kmeans au jeu de donnés avec k=6

```
km2=KMeans(n_clusters=6,init='k-means++',max_iter=300,n_init=10,random_state=0)
y_means=km2.fit_predict(data)
```



Classifieur Kmeans

- Mesurer la performance de notre modèle de clustering kmeans:

```
from sklearn.metrics import classification_report
target_original=dataset.iloc[:,-2]
target_predicted_relabel = np.choose(km2.labels_,[3,1,4,0,2,5]).astype(np.int64)
print(classification_report(target_original,target_predicted_relabel))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        50
           1       1.00      1.00      1.00        50
           2       1.00      1.00      1.00        50
           3       1.00      1.00      1.00        50
           4       1.00      1.00      1.00        50
           5       1.00      1.00      1.00        50

   micro avg       1.00      1.00      1.00       300
   macro avg       1.00      1.00      1.00       300
weighted avg       1.00      1.00      1.00       300
```

Discussion

Algorithme d'apprentissage supervisé	Precision du modéle	Erreur quadratique moyenne
Classifieur Kppv	1.0	0.0
Classifieur de Bernoulli	0.65	0.35
Classifieur Gaussien	1.0	0.0
Classifieur SVM	1.0	0.0
Perceptron	0.97	0.1
Réseau de Neurones	0.55	0.45
K-means	1.0	0.0

Discussion

- D’après le tableau récapitulatif déjà présenté, on peut ordonner les différents classificateurs utilisés selon leurs performance comme suit:
    1. Le classifieur Kppv ,Gaussien et SVM
    2. Perceptron
    3. Classifieur de Bernoulli
    4. Réseau de Neurones

  Remarque: le modèle réseau de neurone a une performance faible par rapport aux autres modèles car notre base est petite donc elle n’assure pas un bon apprentissage de ce modèle.

- Avec la méthode k-means de apprentissage non supervisé, on a été capable de prédire les différentes classes de notre base et de trouver des résultats presque égaux aux résultats des algorithmes supervisés.

Thank You

mariam.omrane@enis.tn
dorra.chaari@enis.tn
khalil.koulsi@enis.tn