ilxloeb7b

January 28, 2025

1 Lab1 (5IR - 2024): Human Activity Recognition (HAR) - Supervised learning

Group name: A

Names: BIENDOU, CHANFREAU

First names: BRIAN, CEDRIC

In this project, we will try to predict 8 different human activities:

1-Recumbent, 2-Sitting, 3-Standing, 4-Walking, 5-Biking, 6-Nordic Walking, 7-Cleaning_Aspirator or 8-Repassing

using different sensors (accelerometer, gyroscope, magnetometer or temperature) placed at different points on the body (hand, torso and ankle).

For the accelerometer, gyroscope and magnetometer sensors, measurements in ms^{-2}, rad/s and μT respectively are taken on the 3-axes at a sampling rate of 100Hz. Each measurement is labeled.

For each sensor and position, a matrix of the measurements collected is provided, along with the activity label (i.e. the ground truth).

After loading the data, this lab is divided into 3 parts: - Part I: Implementing DTW to create a k-NN recognition system; - Part II: Dimension reduction using PCA and classification using neural networks; - Part III: (Bonus) Dimension reduction using k-medoids for DTW + k-NN.

[!!!!!] What you have to do is indicated as Questions and #Todo. [!!!!!]

[1]:
```
# Necessary libraries: ipykernel matplotlib==3.7.3 pandas scikit-learn seaborn

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import math
from scipy import stats

from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.decomposition import PCA
```

```
from sklearn.neural_network import MLPClassifier

import seaborn as sns
from mpl_toolkits.mplot3d import Axes3D
```

2 Preprocessing data

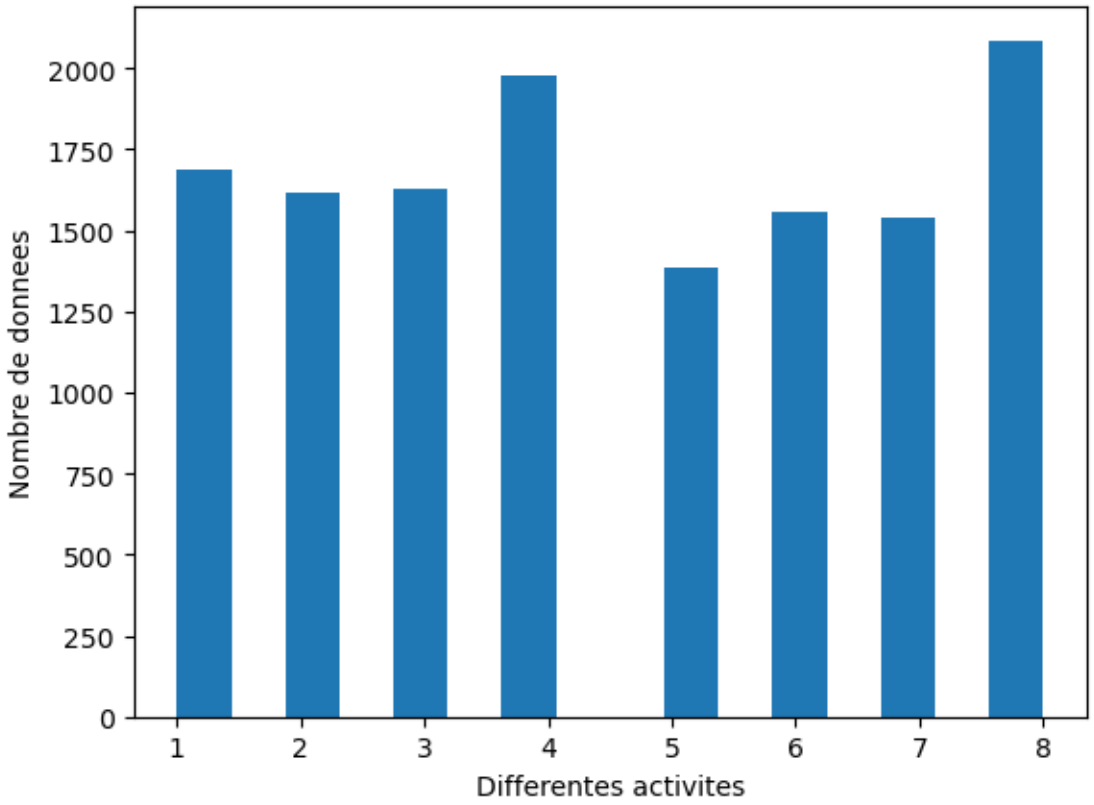
Read the data files for all sensors and the labels

[2]:
```
# Data loading
datas = []

# Sensors part for Temperature
datas.append(np.swapaxes(np.load('DataTemperature.npy'), 1, 2))

# Sensors part (Accelerometer, Gyroscope and Magnetometer) placed on the Hand
# Hand
datas.append(np.load('DataHandAcc.npz')['DataHandAcc'])

#TODO: do the same for the other sensors: accelerator/gyrometer/magnetometer ;␣
 ↪hand/torso/ankle
#pour le torse
datas.append(np.load('DataChestAcc.npz')['DataChestAcc'])
datas.append(np.load('DataChestGyro.npz')['DataChestGyro'])
datas.append(np.load('DataChestMagneto.npz')['DataChestMagneto'])

#pour la cheville
datas.append(np.load('DataAnkleAcc.npz')['DataAnkleAcc'])
datas.append(np.load('DataAnkleGyro.npz')['DataAnkleGyro'])
datas.append(np.load('DataAnkleMagneto.npz')['DataAnkleMagneto'])

#pour la main
datas.append(np.load('DataHandGyro.npz')['DataHandGyro'])
datas.append(np.load('DataHandMagneto.npz')['DataHandMagneto'])

# Labels
labels = np.load('labels.npz')['labels']
```

[3]:
```
RawData=np.concatenate(datas, axis=2)
```

[4]:
```
# Loading the ground truth: labels
fl=np.load('labels.npz')
labels=fl['labels']

# Repartition of data by class
plt.hist(labels,bins=16)
plt.xlabel('Differentes activites')
```

```
plt.ylabel('Nombre de donnees')
plt.show()
```



Scaling the data and smoothing the temporal series

[15]:
```
N, T, S = RawData.shape
X_reshaped = RawData.reshape(N, T*S)

scaler = StandardScaler()
X_scaled_2d = scaler.fit_transform(X_reshaped)
Data = X_scaled_2d.reshape(N, T, S)
```

```
6
500
13477
```

[18]:
```
# Smoothing function
def smooth_signal(X, k):
    for i in range(len(X)):
        Xi = X[i]
        downsample=math.ceil(len(Xi)/k)
```

```
        for j in range(downsample):
            Xj = Xi[j*k:min((j+1)*k, len(Xi)),:]
            X[i,j] = np.mean(Xj, axis=0)
    X = X[:,:downsample,:]
    return X

# Smooth training and testing data
SMOOTH_RATIO=100
X = smooth_signal(Data, SMOOTH_RATIO)

print(f"Examples: {N}")
print(f"Sensors: {S}")
print(f"Timestamp: {T}")
```

```
Examples: 13477
Sensors: 6
Timestamp: 500
```

Questions (look inside the data) - How many examples in the dataset ? il y'a 13 477 exemples qui est repartie sur 8 activités

- What is the difference between the variables RawData, Data, and X ?
  - RawData: represente les données brutes
  - Data: qui sont les Données Traités
  - X: Données Lissés (sous échantilloné)
- About Data (or RawData):
  - Considering the sensors give values with a frequency of 100Hz, what is the length of the measurement window for each example ?

    Durée de la fenêtre de mesure pour chaque exemple : 500/100 = 5,0 secondes
  - For each example, for each time step, what is the dimension of the data i.e. how many sensors ?

    qui represente le S donc 6 sensors
- About X: what is the new frequency of the measurement ?

  `X=100 fréquence=100 alors 100/100 = 1Hz`

3 Part I-a: Implementing the dynamic programming algorithm

1. Implement the Dynamic Time Warping algorithm (DTW) in a python function such that:

- the local distance used is a parameter which is a callable function with 2 inputs and 1 output: e.g. euclidian distance(a, b)
- local distances can be computed with any type of input local data: e.g. (a,b)=('A','T'), (a,b)=(12,22), (a,b)=((1,2,3), (4,5,6))
- there are two optionnal parameters: local and global constraints

The DTW algorithm is given and explained in the Lecture 3… Slides 12-16.

[92]:
```python
# TODO

# Define the Dynamic Time Warping algorithm
#   => header: def dtw(a, b, distance, w=(1,1,1), gamma=5):
def dtw(a, b, distance, w=(1,1,1), gamma=5):
    N = len(a)
    M = len(b)

    g = np.full((N + 1, M + 1), math.inf)
    g[0, 0] = 0

    for j in range(1, M + 1):
        g[0, j] = math.inf
    for i in range(1, N + 1):
        g[i, 0] = math.inf

    for i in range(1, N + 1):
        for j in range(1, M + 1):
            if abs(i - j) >= gamma:
                g[i, j] = math.inf
                continue
            localD = distance(a[i - 1], b[j - 1])
            g[i, j] = min(
                g[i - 1, j] + w[0] * localD,
                g[i - 1, j - 1] + w[1] * localD,
                g[i, j - 1] + w[2] * localD
            )

    S = g[N, M] / (N + M)
    return S


# Define distances functions
#   => header: def dist(a, b)
def euclidian_distance(a, b):
    return np.linalg.norm(np.array(a) - np.array(b))
```

2. Test your methods on the examples given in class. Typically:

- example 1:
  - DTW between (4, 6, 1) and (5, 1) => resulting score S=0.4.
  - local distance: the euclidian distance
  - local constraints: (1,1,1)
  - global constraint: 2
- example 2:
  - DTW between “ATGGTACGTC” and “AAGTAGGC” => resulting score S=1/6

- local distance: 0 if the characters are equal, 1 else
- local constraints: (1,1,1)
- global constraint: 4

[93]:
```
#TODO

# Example 1:
a = (4, 6, 1)
b = (5, 1)

dtw1 = dtw(a, b, euclidian_distance, w=(1,1,1), gamma=5)
print(dtw1)
# Example 2:
i = ('A', 'T', 'G', 'G', 'T', 'A', 'C', 'G', 'T', 'C')
j = ('A', 'A', 'G', 'T', 'A', 'G', 'G', 'C')

def char_dist(x, y):
    return 0 if x == y else 1

dtw2 = dtw(i, j, char_dist, w=(1,1,1), gamma=4)
print(dtw2)
```

```
0.4
0.16666666666666666
```

Question - What are the effects of the local and global constraints on the DTW computation ?

Local: Affecte les étapes simples dans le chemin de déformation

Global: Limite le chemin de déformation sur une région spécifique et empêche les alignement

4 Part I-b: DTW human activity recognition system

In this section, we want to use the k-Nearest Neighbor method (k-NN) combined with the DTW algorithm in order to construct a model for the classification of human acivity based on the previously loaded data.

The principle of our approach is the following: 1) For each class, compute the DTW score between the test temporal profile and the temporal profiles of the class. Test with data from only one sensor then include them all if possible. 2) Determine the class membership by k-nearest neighbors.

We provide here the class KNN overlaying the k-NN algorithm implemented inside the scikit-learn library (*sklearn.neighbors.KNeighborsClassifier*): - In scikit-learn, only one dimensional data is accepted for classification, - Here, we use multiple sensors, thus two dimensional data for each example which can be treated easily using the **DTW algorithm with the euclidian distance**.

In scikit-learn, KNeighborsClassifier takes as argument *metrics* which can be a callable function taking as input 2 arrays representing 1D vectors and returning one value indicating the distance between those vectors. **We provide a metric function using the DTW function you implemented which should be called *dtw(...)*.**

[94]:
```
def dtw_KNN(x1, x2):
    n=x1.shape[0]
    folds=Data.shape[2]
    x1 = x1.reshape(int(n/folds),folds)

    n=x2.shape[0]
    folds=Data.shape[2]
    x2 = x2.reshape(int(n/folds),folds)

    res=dtw(x2, x1, w=(1,1,1), gamma=5, distance=euclidian_distance)

    return res

class KNN:
    def __init__(self, n_neighbors=1, metric=dtw_KNN, n_jobs=-1):
        self.n_neighbors=n_neighbors
        self.metric=metric
        self.n_jobs=n_jobs
        self.model=n_jobs
        self.model = KNeighborsClassifier(n_neighbors, metric=self.metric,␣
 ↪n_jobs=-1)

    def fit(self, xtrain, y_train):
        m, n, k = xtrain.shape
        xtrain = xtrain.reshape(m,n*k)
        self.model.fit(xtrain, y_train)
        return self.model

    def predict(self, xtest):
        m, n, k = xtest.shape
        xtest = xtest.reshape(m,n*k)
        return self.model.predict(xtest)

    def score(self, xtest, ytest):
        m, n, k = xtest.shape
        xtest = xtest.reshape(m,n*k)
        return self.model.score(xtest, ytest)
```

Information: Here are the main commands: - The line *model = KNN(n_neighbors=1])* creates an object of type classifier based on the n_neighbors closest neighbors, - The instruction *model.fit(X, y)* uses the data to define the classifier (training), - The command *model.predict()* is used to classify the new examples, - The command *model.predict_proba()* allows to estimate the probability of the proposed classification, - The command *clf.score(xtest, ytest)* computes the global score of the classifier for a given dataset.

4.1 Training and test datasets

Randomly create the training and test datasets: - The training dataset must be of size 160. Be careful to select examples such that the labels are balanced (20 examples for each activity) - The test set of size 1000.

[ ]:
```
# TODO
# Pour que les résultats soient les mêmes à chaque fois
np.random.seed(42)

# on prend 20 exemples pour chaque classe
indices = [np.where(labels == i)[0] for i in range(1, 9)]
train_indices = np.hstack([np.random.choice(class_indices, 20, replace=False)␣
 ↪for class_indices in indices])

# on mélange un peu
np.random.shuffle(train_indices)

# on test avec 1000 exemples
test_indices = np.random.choice(np.setdiff1d(np.arange(len(labels)),␣
 ↪train_indices), 1000, replace=False)


X_train, X_test = X[train_indices], X[test_indices]
y_train, y_test = labels[train_indices], labels[test_indices]

print(f"Taille du jeu d'entraînement: {X_train.shape[0]} exemples")
print(f"Taille du jeu de test: {X_test.shape[0]} exemples")
```

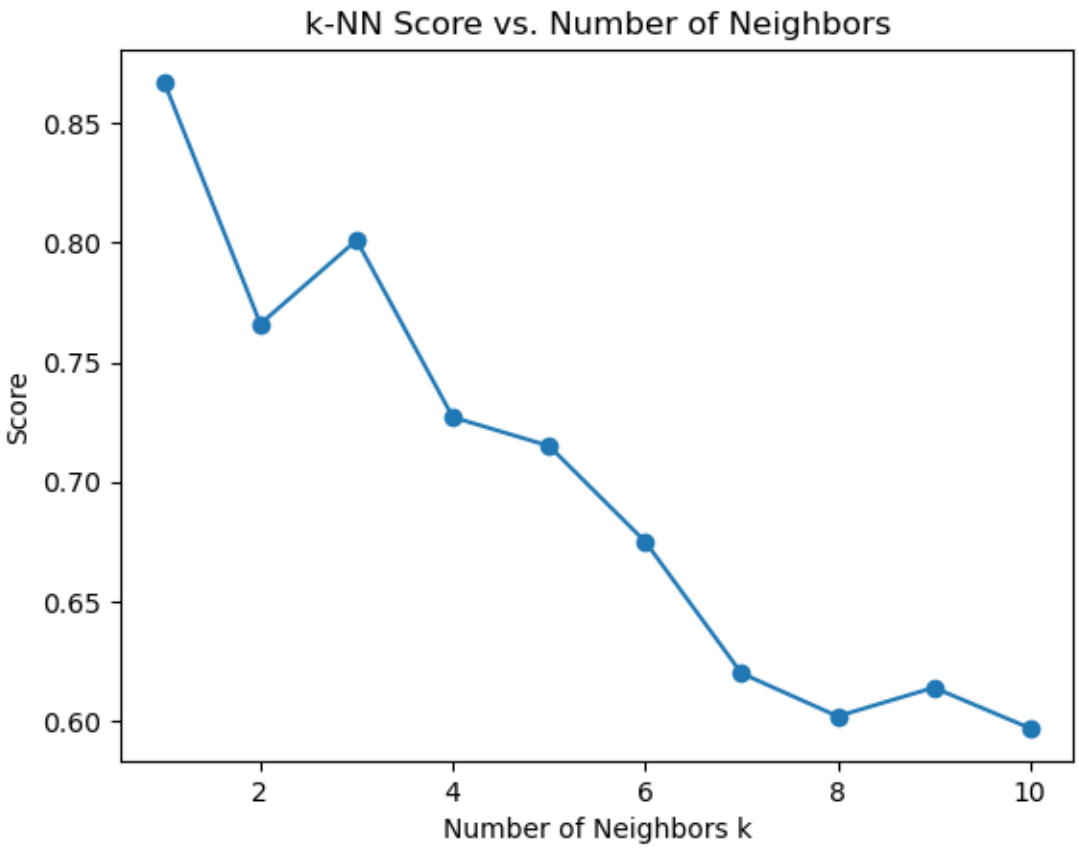
4.2 Create and train the k-NN model

Try several choices for the number of neighbors k (e.g. 1 to 10). and select the one giving best results on the test set. Ideally, plot the evolution of the score on the test set depending on k

[100]:
```
# TODO
# on essaie avec différentes valeurs de k
k_values = range(1, 11)
scrore = []

for k in k_values:
    #on entraine le modele
    model = KNN(n_neighbors=k)
    model.fit(X_train, y_train)

    #on calcul son score
    score = model.score(X_test, y_test)
    scrore.append(score)
    print(f"k={k}, Score={score}")
```

```
plt.plot(k_values, scrore, marker='o')
plt.xlabel('Number of Neighbors k')
plt.ylabel('Score')
plt.title('k-NN Score vs. Number of Neighbors')
plt.show()


#commentaire : k=1 donne les meilleurs résultats pour notre classification␣
 ↪d'activités humaines.
```

```
k=1, Score=0.867
k=2, Score=0.766
k=3, Score=0.801
k=4, Score=0.727
k=5, Score=0.715
k=6, Score=0.675
k=7, Score=0.62
k=8, Score=0.602
k=9, Score=0.614
k=10, Score=0.597
```



4.3 Evaluation

From the prediction on the test set obtained after DTW: - compute the confusion matrix that counts the number of right and wrong classifications. - compute the score on the training and test sets.

[101]:
```
from sklearn.metrics import confusion_matrix, classification_report

# on entraine le modèle
y_pred = model.predict(X_test)

# on calcule la matrice de confusion
conf_matrix = confusion_matrix(y_test, y_pred)
print('Matrice de confusion par DTW')
print(conf_matrix)

# on calcule le score de classification
class_report = classification_report(y_test, y_pred)
print('Score de classification')
print(class_report)
```

```
Matrice de confusion par DTW
[[ 98   2   4   2   0   1   0  19]
 [ 11  48   8   0   0   0   0  46]
 [  1  10  80   0   0   0   2  14]
 [  0   0  48 118   0   0   2   0]
 [  3  20   0   0  61   4   0  23]
 [  7   2  16  20   9  52   1   0]
 [  3   2  43  16   0   3  51   6]
 [  2  18   8   0  15   1  11  89]]
Score de classification
              precision    recall  f1-score   support

           1       0.78      0.78      0.78       126
           2       0.47      0.42      0.45       113
           3       0.39      0.75      0.51       107
           4       0.76      0.70      0.73       168
           5       0.72      0.55      0.62       111
           6       0.85      0.49      0.62       107
           7       0.76      0.41      0.53       124
           8       0.45      0.62      0.52       144

    accuracy                           0.60      1000
   macro avg       0.65      0.59      0.60      1000
weighted avg       0.65      0.60      0.60      1000
```

Questions: - When creating the *sklearn.neighbors.KNeighborsClassifier* model in KNN(), what is the purpose of the n_jobs=-1 parameter ? - il accélère le calcul de manière parallèle - Usually, ~80% of the data is used as train set, why do we have to choose a very small number of examples for this purpose here ? - la DTW est tres lent a calculer, donc plus de donner implique plus de temps de calcul - What is the best choice for number of neighbors ? - k=1 donne un meilleur score, soit environ 85%, et on remarque que les performances diminue lorsque k devient plus grand - Analyse the results (accuracy, execution time, ...) - le temps de calcul beaucoup trop long soit plus de 13 min pour les 11 k, une précision partiellement correcte. - Briefly, what are the advantages and disadvantages of the k-NN method: optimality ? computation time ? scalability ? - marche bien avec peu de données, pas d'entrainement nécessaire, mais c'est lent pour prédire, et pas pratique pour bcp de données

5 Part II: Comparison of dynamic programming with a neural network classification method after PCA dimension reduction

In this section, we will compare the results of the classification using k-NN and DTW with those of another classification method: dimension reduction using PCA combined with neural networks (Multi-Layer Perceptron).

We'll be using the functions for calculating PCA and MLPClassifier via the *scikit-learn* python library.

5.1 Training and test datasets

Starting from the data in the variable X: - For each example, transform the data to be 2D, i.e. we want to concatenate the measurement of all sensors (one can use numpy.reshape). - Randomly create the training and test datasets with 80% of the data in the training dataset (there is a function for this..).

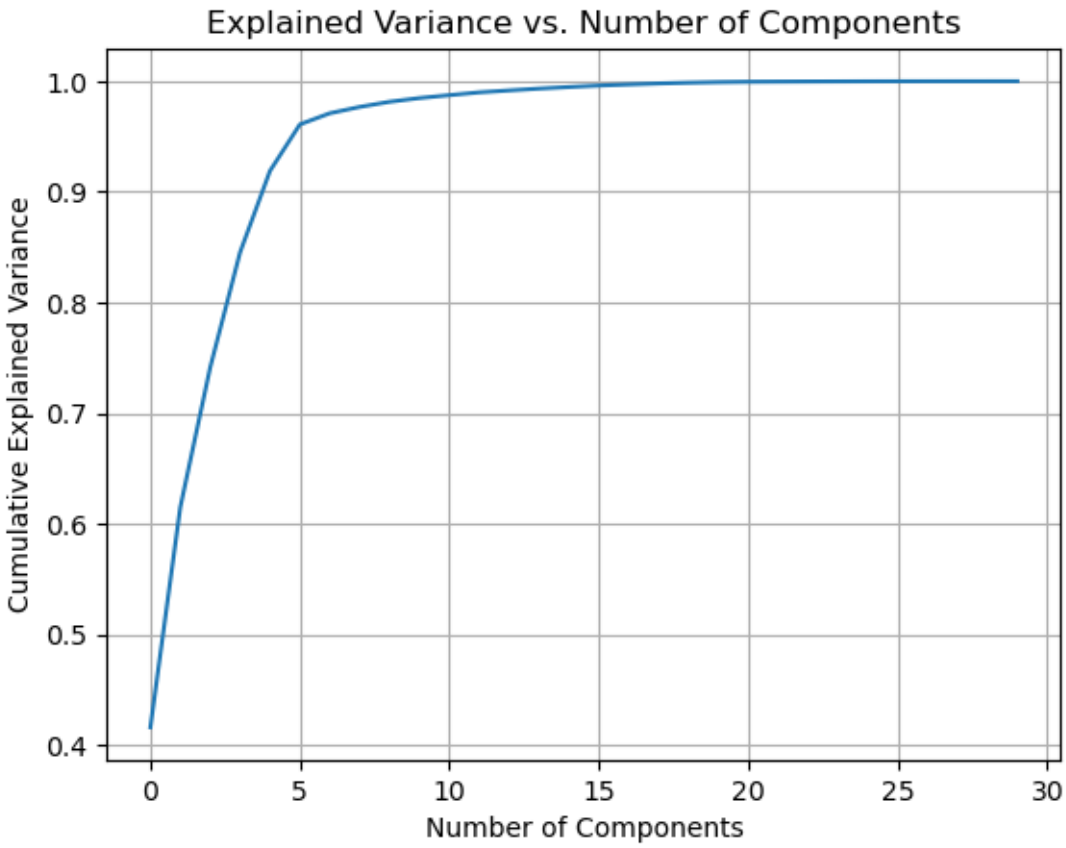
[102]:
```
# TODO
# on transforme les données en 2D
N, T, S = X.shape
X_reshaped = X.reshape(N, T * S)

# ici on sépare les données en jeu d'entrainement et jeu de test
X_train, X_test, y_train, y_test = train_test_split(X_reshaped, labels,␣
 ↪test_size=0.2, random_state=42)

print(f"Training set size: {X_train.shape[0]}")
print(f"Test set size: {X_test.shape[0]}")
```

```
Training set size: 10781
Test set size: 2696
```

5.2 Determining the best number of components for PCA

We want to determine the ideal number of components for the dimension reduction, i.e. a reduced number of variables reprensenting well the data, using using the *PCA* function in the *scikit-learn* library: - *pca=PCA()*: Create a PCA model - *pca.fit(x)*: From the training data, calculate the p principal axes of the PCA by extracting the p eigenvectors, denoted X_1, X_2,..., X_p, associated with the p largest eigenvalues of the variance-covariance matrix variance-covariance matrix Σ_{App}. These eigenvectors will form the new database. - Get the explained variance ratio of each principal component (see the documentation) and plot these - Determine how many principal components must be kept to explain 99% of the variance.

[103]:
```
#TODO
# on crée et applique la PCA
pca = PCA()
pca.fit(X_train)

#la variance expliquée par chaque composant
explained_variance_ratio = pca.explained_variance_ratio_


plt.plot(np.cumsum(explained_variance_ratio))
plt.xlabel('Number of Components')
plt.ylabel('Cumulative Explained Variance')
plt.title('Explained Variance vs. Number of Components')
plt.grid()
plt.show()

# nombre de composantes pour 99%
cumulative_variance = np.cumsum(explained_variance_ratio)
n_components = np.argmax(cumulative_variance >= 0.99) + 1
print(f"Number of components to explain 99% of the variance: {n_components}")
```




```
Number of components to explain 99% of the variance: 13
```

5.3 PCA pre-processing

Using the previously determined number of components: - *pca=PCA(n_components=[...])*: Create the PCA model with right number of components - *pca.fit(x)*: the model must be trained on the training data again - *pca.transform(x)*: Project the data from the training and test database into this new database by multiplying each vector vector by the base $P = [X_1 X_2 ... X_p]$.

[104]:
```
#TODO

pca = PCA(n_components=n_components)
pca.fit(X_train)

# Transformation
X_train_pca = pca.transform(X_train)
X_test_pca = pca.transform(X_test)

print(f"Training set size after PCA: {X_train_pca.shape}")
print(f"Test set size after PCA: {X_test_pca.shape}")
```

```
Training set size after PCA: (10781, 13)
Test set size after PCA: (2696, 13)
```

Questions - What is the number of components kept ? - on a 13 composants - Consider this: the data for each example is a 1D vector where each value (feature) is the measurement of 1 sensor at 1 time step. In your opinion, does it make sense to use this as features and apply dimension reduction with PCA directly ? Do you expect good results for the upcoming classification ? - on remarque qu'on permet l'aspect temporel des donnés

5.4 Classification using neural networks

We will use the library **sklearn.model_selection** to develop ANN of type Multi Layer Perceptron (MLP). We use the class: MLPClassifier.

This model optimizes the cross entropy function (loss function) and a gradient based method. The main parameters of this class are: - **hidden_layer_sizes** is a tuple that specifies the number of neurons in each hidden layer; from the entrance to the exit. For example, a unique hidden layer of 55 neurons, hidden_layer_sizes = (55); for three hidden layers of size respectively 50, 12 and 100 neurons, hidden_layer_sizes = (50, 12, 100). - **activation** defines the activation function for hidden layers: {"identity", "logistic", "tanh", "relu"}, default "relu" - 'identity', no-op activation, linear bottleneck, returns f(x) = x - 'logistic', logistic sigmoid function, returns f(x) = 1 / (1 + exp(-x)). - 'tanh', hyperbolic tan function, returns f(x) = tanh(x). - 'relu', rectified linear unit function, returns f(x) = max(0, x) - **max iter**, default =200, indicates the max number of iterations of the solver.

The main commands are: - *mlp = MLPClassifier(hidden_layer_sizes=(10,5,), max_iter=300, activation='relu')* - *mlp.fit(xtrain, ytrain)* - *ypred = mlp.predict(xtest)*

Accuracy of the model can eventually be computed using the library **sklearn.metrics** and the commands: - Score: accuracy_score(y_test, y_pred) - Report: classification_report(y_test, y_pred))

The usual sklearn learning and testing functions are as follows: (**fit**, **predict**, **score**).

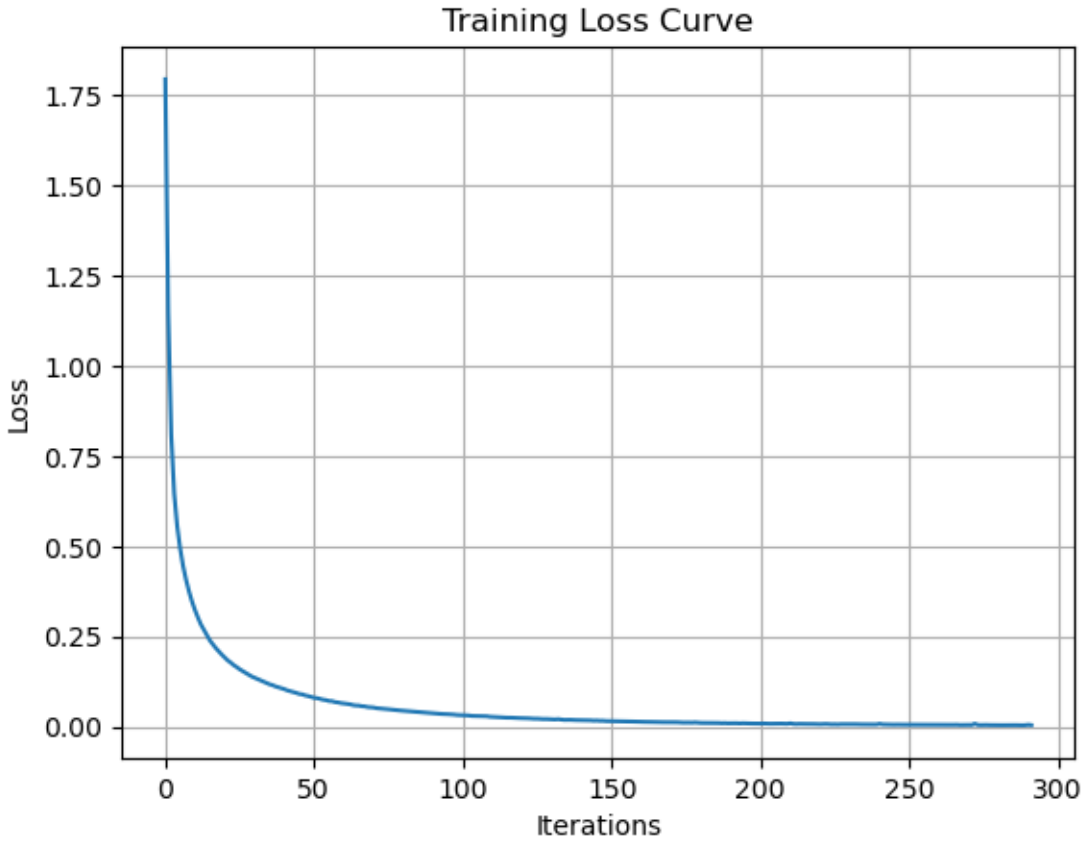
5.5 Create and train the MLP model

- Try several choices for the number of layers and the activation function.
- Plot the evolution of the training score during the training (can be obtained from the model, see the documentation).

[105]:
```
#TODO
# entrainement du modèle
mlp = MLPClassifier(hidden_layer_sizes=(50, 25), max_iter=300,␣
 ↪activation='relu', random_state=42)
mlp.fit(X_train_pca, y_train)

# on prédit les classes
y_pred_mlp = mlp.predict(X_test_pca)

#l'accuarcy
accuracy = accuracy_score(y_test, y_pred_mlp)
print(f"Accuracy of MLP model: {accuracy}")
```

```
plt.plot(mlp.loss_curve_)
plt.xlabel('Iterations')
plt.ylabel('Loss')
plt.title('Training Loss Curve')
plt.grid()
plt.show()
```

```
Accuracy of MLP model: 0.9777448071216617
```



5.6 Evaluation

From the prediction on the test set obtained after DTW: - compute the confusion matrix that counts the number of right and wrong classifications. - compute the score on the training and test sets.

[106]:
```
# TODO
# on calcule la matrice de confusion
conf_matrix_mlp = confusion_matrix(y_test, y_pred_mlp)
```

```
print('Confusion Matrix for MLP')
print(conf_matrix_mlp)

#on calcule le score de classification
class_report_mlp = classification_report(y_test, y_pred_mlp)
print('Classification Report for MLP')
print(class_report_mlp)
```

```
Confusion Matrix for MLP
[[345   1   2   0   0   0   0   1]
 [  6 306   4   0   0   0   2   2]
 [  3   8 322   0   0   0   0   4]
 [  0   0   0 384   1   0   0   0]
 [  0   2   0   0 261   1   0   0]
 [  0   0   0   0   1 352   0   0]
 [  1   0   0   2   0   0 292   6]
 [  0   0   9   0   0   0   4 374]]
Classification Report for MLP
              precision    recall  f1-score   support

           1       0.97      0.99      0.98       349
           2       0.97      0.96      0.96       320
           3       0.96      0.96      0.96       337
           4       0.99      1.00      1.00       385
           5       0.99      0.99      0.99       264
           6       1.00      1.00      1.00       353
           7       0.98      0.97      0.97       301
           8       0.97      0.97      0.97       387

    accuracy                           0.98      2696
   macro avg       0.98      0.98      0.98      2696
weighted avg       0.98      0.98      0.98      2696
```

Questions: - What is the best model of MLP you found ? What did you try ? - le meilleur modele MLP trouvé a 300 iteration maximum, avec un score plutot optimal de 98% - Analyse the results (accuracy, execution time, Underfitting/Overfitting, ...) - on a une bonne accuracy de 98%, une très bonne convergence avec un temps d'execution très rapide - Briefly, what are the advantages and disadvantages of the MLP compared to k-NN method: optimality ? computation time ? scalability ? - le MLP est beaucoup meilleur que le KNN car - plus rapide en prédiction et demande moin de mémoire - par contre la configuration est plus complexe, necessite un entrainement - In the end which machine learning method gave the best results for the classification of sensors data ? Was it expected and why ? - a MLP a donné un meilleur resultat qui est de 98% comparé a 86% pour le KNN