Import des librairies

```
# pip install
!pip install chart_studio tqdm
```

```
    Requirement already satisfied: chart_stud
    Requirement already satisfied: tqdm in /u
    Requirement already satisfied: plotly in 
    Requirement already satisfied: retrying>=
    Requirement already satisfied: requests i
    Requirement already satisfied: six in /us
    Requirement already satisfied: certifi>=2
    Requirement already satisfied: idna<3,>=2
    Requirement already satisfied: urllib3!=1
    Requirement already satisfied: chardet<4,
```

```
#import statements
import os
import logging
import pandas as pd
import numpy as np
from sklearn.datasets import fetch_lfw_people
import matplotlib.pyplot as plt
from collections import Counter
%matplotlib inline
from sklearn.model_selection import train_test_split, GridSearchCV, RandomizedSearc

from sklearn.preprocessing import scale
from sklearn import model_selection, tree
from sklearn.decomposition import PCA
from sklearn.ensemble import BaggingClassifier, RandomForestClassifier
from sklearn.linear_model import LinearRegression
from sklearn.cross_decomposition import PLSRegression, PLSSVD
from sklearn.metrics import mean_squared_error, f1_score, classification_report
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline

from scipy.stats import reciprocal, uniform

# Some libraries for PCA visualization
import seaborn as sns
#Make Plotly figure
import chart_studio.plotly as py
import plotly.graph_objs as go

# Libraries for SVM
from sklearn.svm import SVC, LinearSVC

# Libraries for Naive Bayes
from sklearn.naive_bayes import GaussianNB, MultinomialNB, ComplementNB

# Misc
```

```
import time
from tqdm import tqdm
import random
random.seed(1)

# Display progress logs on stdout
logging.basicConfig(level=logging.INFO, format='%(asctime)s %(message)s')
```

Helper functions

```
def plot_gallery(images, h, w, n_row=3, n_col=4):
  """Helper function to plot a gallery of portraits"""
  plt.figure(figsize=(1.8 * n_col, 2.4 * n_row))
  plt.subplots_adjust(bottom=0, left=.01, right=.99, top=.90, hspace=.35)
  for i in range(n_row * n_col):
      plt.subplot(n_row, n_col, i + 1)
      plt.imshow(images[i].reshape((h, w)), cmap=plt.cm.gray)
      plt.xticks(())
      plt.yticks(())


def plot_mean_stdev(X, n_features):
  """
  Trace la moyenne et l'ecart type de chaque feature du dataset passé en entrée.

  Parametres
  ----------
  X : pandas.DataFrame, requis
      Le jeu de donnée en question
  n_features: int, requis
      Le nombre de features à tracer.
      En pratique le nombre de features de X.
  """
  plt.plot([i for i in range(n_features)], np.std(X, axis=0), '.b', label="Standard
  plt.plot([i for i in range(n_features)], np.mean(X, axis=0), '.r', label="Mean")
  plt.legend()


def get_ncomponents(pca, cum_evr):
  """
  Retourne le nombre de composants principaux dont le cumul des ratios de variance

  Parametres
  ----------
  pca : sklearn.decomposition.PCA, requis
      L'objet pca initialisé sur un jeu de donné.
  cum_evr: float, requis
      Le ratio de variance expliqué souhaité.
      Doit etre compris entre 0 et 1.
  """
  assert (cum_evr <= 1 and cum_evr >= 0)
  cumulated_evr = np.cumsum(pca.explained_variance_ratio_)
  n = np.argmax(cumulated_evr >= cum_evr)
```

```
    n = np.argmax(cumulated_evr >= cum_evr)
    return n


def accuracy(y_true, y_pred):
    """Calcule la metrique accuracy d'un modele"""
    return sum(y_pred==y_true) / len(y_true)



def balance_classes(x,y,subsample_size=1.0):

    class_xs = []
    min_elems = None

    for yi in np.unique(y):
        elems = x[(y == yi)]
        class_xs.append((yi, elems))
        if min_elems == None or elems.shape[0] < min_elems:
            min_elems = elems.shape[0]

    use_elems = min_elems
    if subsample_size < 1:
        use_elems = int(min_elems*subsample_size)

    xs = []
    ys = []

    for ci,this_xs in class_xs:
        if len(this_xs) > use_elems:
            this_xs = this_xs.reindex(np.random.permutation(this_xs.index))

        x_ = this_xs[:use_elems]
        y_ = np.empty(use_elems)
        y_.fill(ci)

        xs.append(x_)
        ys.append(y_)

    xs = pd.concat(xs)
    ys = pd.Series(data=np.concatenate(ys),name='target', dtype=int)


    return xs,ys


# Function providing X and y data depending on some passed in parameters.
def provide_dataset(min_faces_per_person=0, size_factor=None, scale=True, split=Tru
    """
    Fournit le dataset sous plusieurs format en fonction des parametres passés en e

    Parametres
    ----------
    min_faces_per_person : int, optionnel
        Le min_faces_per_person de la fonction fetch_lfw_people.
    size_factor: None ou float, requis
        Le pourcentage du dataset retourné par la fonction fetch_lfw_people.
```

```
        Le pourcentage du dataset retourné par la fonction fetch_lfw_people.
        Si None toute la dataset est prise en compte.
        Si c'est un float, le pourcentage correspondant du dataset est pris en comp
        Si c'est un float, il doit etre compris entre 0 et 1.
    scale: booléen
        Renseigne si le dataset doit etre redimensionné avec une valeur moyenne de
    split: booléen
        Rensigne si le dataset doit etre divisé en jeu d'entrainement et en jeu de
        A noter que si le parametre scale est True, le dataset est divisé.
    resize: Le parametre resize de la fonction fetch_lfw_people.
    slice_: Le parametre slice_ de la fonction fetch_lfw_people.
    balance: Booléen, optionnel
        Indique s'il faut équilibrer les classes ou pas.
  """

  assert size_factor==None or (size_factor <= 1 and size_factor>=0)
  if slice_== None:
    dataset = fetch_lfw_people(data_dir, return_X_y=True, min_faces_per_person=min_
  else:
    dataset = fetch_lfw_people(data_dir, return_X_y=True, min_faces_per_person=min_
  if size_factor:
      data_size = int(size_factor * len(dataset[0]))
      bool_mask = [True] * data_size + [False] * (len(dataset[0]) - data_size)
      random.shuffle(bool_mask)
      dataset = (dataset[0][bool_mask], dataset[1][bool_mask])

  data_df = pd.DataFrame(dataset[0])
  data_df["target"] = dataset[1]

  X = data_df.drop("target", axis=1)
  y = data_df["target"]
  if balance:
    X, y = balance_classes(X, y)

  if scale or split:
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random

  if scale:
    scaler = StandardScaler()
    # On redimensionne les jeux d'entrainement et de test.
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test.astype(np.float32))
    return X_train_scaled, X_test_scaled, y_train, y_test
  elif split:
    return X_train, X_test, y_train, y_test
  return X, y


def fit_evaluate(model, X_train_scaled, X_test_scaled, y_train, y_test, timeit=Fals
  """
  Entraine et évalue la performance d'un modèle sur un dataset.
  Retourne les resultats d'évaluation sur le training et testing set
  et potentiellement les temps d'execution et de prediction en fonction des paramet

  Paramètres
  ----------
```

```
    model: modèle sklearn, requis
        Le modèle que l'on souhaite entrainer.
    X_train_scaled: pd.DataFrame, requis
        Le jeu d'entrainement.
    X_train_scaled: pd.DataFrame, requis
        Le jeu de test.
    y_train: pd.Series, requis
        Le target d'entrainement.
    y_test: pd.Series, requis
        Le target de test.
    timeit: Booléen, optionnel
        Indique si on veut chronometrer le temps d'entrainement et de prédiction.
        Si True, la fonction fit_evaluate retourne en plus le temps d'entrainement et
        Le temps d'entrainement est le temps total d'entrainement du modele sur tout
        Le temps de prediction est le temps de prediction moyen du modele pour une in
    """

    # Entrainement
    # Mesure du temps d'entrainement
    training_time_start = time.time()
    model.fit(X_train_scaled, y_train)
    training_time_end = time.time()

    training_time = training_time_end - training_time_start

    y_pred_train = model.predict(X_train_scaled)
    pred_time_start = time.time()
    y_pred_test = model.predict(X_test_scaled)
    pred_time_end = time.time()

    pred_time = pred_time_end - pred_time_start
    # avg prediction time for a single instance
    avg_pred_time = pred_time / len(X_test_scaled)

    acc_ =  accuracy(y_train, y_pred_train), accuracy(y_test, y_pred_test)
    f1_score_ = f1_score(y_train, y_pred_train, average='micro'), f1_score(y_test, y_

    if timeit:
      return acc_ , f1_score_, training_time, avg_pred_time

    return acc_ , f1_score_
```

Exercice 1

Chargement du dataset

```
data_dir = os.path.join(".")
dataset = fetch_lfw_people(data_dir)
```

```
    Downloading LFW metadata: https://ndownlo
```

```
2020-11-01 22:19:20,578 Downloading LFW me
Downloading LFW metadata: https://ndownlo
2020-11-01 22:19:21,617 Downloading LFW me
Downloading LFW metadata: https://ndownlo
2020-11-01 22:19:22,532 Downloading LFW me
Downloading LFW data (~200MB): https://nd
2020-11-01 22:19:23,662 Downloading LFW da
```

Dans un premier temps, on va utiliser la donnée brute des pixels et non les images car les SVM n'utilisent pas la correlation spatiale des pixels

```
# Dans un premier temps, on va utiliser la donnée brute des pixels et non les image
data_df = pd.DataFrame(dataset["data"])
data_df["target"] = dataset["target"]
data_df.head()
```

	0	1	2	
0	34.000000	29.333334	22.333334	22.000(
1	158.000000	160.666672	169.666672	168.333
2	77.000000	81.333336	88.000000	108.666(
3	11.333333	11.333333	11.666667	12.666(
4	32.333332	31.333334	31.333334	33.666(

5 rows × 2915 columns

Informations générales

Généralités

```
# Visualisons quelques unes des images en question.
_,h,w = dataset.images.shape

imgs = dataset['data'].reshape((-1, h, w))

plot_gallery(imgs, h, w)
plt.show()
```

Notons que les images sont en grayscale. En outre, elles sont centrées sur les visages. Il n'ya pas de bruit de background.

```
data_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 13233 entries, 0 to 13232
Columns: 2915 entries, 0 to target
dtypes: float32(2914), int64(1)
memory usage: 147.2 MB
```

```
data_df.describe()
```

	0	1	
count	13233.000000	13233.000000	13233.0000(
mean	73.645020	76.609215	82.0471(
std	50.931728	50.792950	51.22212
min	0.000000	0.000000	0.0000(
25%	33.000000	35.666668	40.0000(
50%	62.666668	67.000000	74.6666(
75%	105.666664	109.000000	116.6666(
max	254.000000	254.666672	255.0000(

8 rows × 2915 columns

On observe que notre dataset est composé 13233 instances exprimées par 2914 features de pixels exprimés en échelles de gris en prenant des valeurs comprises entre 0 et 255. La targuet en revanche, prend des valeurs comprises entre 0 et 5748.

Instances par classe

```
# Nombre d'instances par classe
instances_per_class = Counter(data_df["target"])


# Conversion en pandas.Series
instances_per_class_sr = pd.Series(instances_per_class).sort_index()


# Impression de statistiques autour du nombre d'instances par classe.
print(instances_per_class_sr.describe())
```

```
count    5749.000000
mean        2.301792
std         9.016410
min         1.000000
25%         1.000000
50%         1.000000
75%         2.000000
max       530.000000
dtype: float64
```

Nous voyons que 50% des personnes (les classes) aparaissent une fois et que et que 75% des personnes apparaissent moins de deux fois. Il y a cependant des personnes apparaissant un nombre elevé de fois comme le témoigne la valeur maximale décrit par nos statistiques qui s'èleve à 530 fois. Toutefois, la moyenne d'apparition, mu, est de 2 fois environs avec une déviation standard de, sigma, 9. On peut en déduire que 95% des personnes apparaissent au plus mu+2sigma = 20 fois et 99.7% au plus mu+3sigma = 30 fois.

C'est ce que nous allons vérifier dans la cellule suivante.

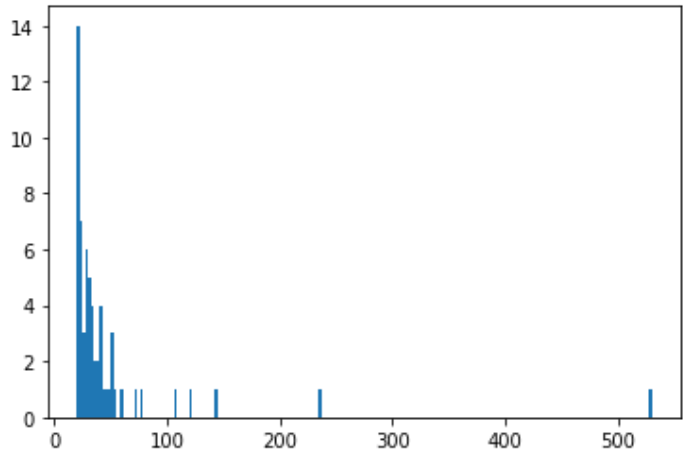
```
percentile = 99
print("{}% des personnes apparaissent au plus {} fois.".format(percentile, int(np.p
```

```
99% des personnes apparaissent au plus 20
```

```
sorted(instances_per_class.values())[-10:]
```

```
[53, 55, 60, 71, 77, 109, 121, 144, 236, !
```

```
# Tracé de l'histogramme des classes (personnes) pour celles apparaissant plus de 2
plt.hist(instances_per_class.values(), bins=200, range=(20,530))
plt.show()
```



En définitive, on voit bien que les classes ne sont pas equilibrées. Néanmoins, la majorité des personnes apparait peu de fois (1 à 2 fois). Seul 1% des personnes apparait plus de 20 fois. Cependant on estime que apparaitre 1 ou 2 fois ne permet pas au modele de bien apprendre. En plus des contraintes de ressources, il est necessaire de reduire le nombre de personnes à classifier, i.e eliminer les personnes apparaissant 1 ou 2 fois. Ensuite il faudra equiblibrer la frequence d'apparition des classes. On n'appliquera pas de technique d'upsampling du minority set car on fait face a un probleme avec déja enormement de dataset. Donc l'idee cest plutot de reduire les instances de personnes apparaissant beaucoup de fois.

Valeurs manquantes

```
# Valeurs manquantes
print(f"Total de valeurs manquantes: {data_df.isnull().sum().sum()}")
```

```
Total de valeurs manquantes: 0
```

Il n'y a pas de valeurs manquantes.

Corrélation

```
# Attention: cette cellule prends environ 2minutes30 pour tourner.
corr_matrix = data_df.corr()
```

```
corr_matrix[1][:5]
```

```
f, ax = plt.subplots(figsize=(12, 9))
sns.heatmap(corr_matrix, vmax=1, square=True);
```

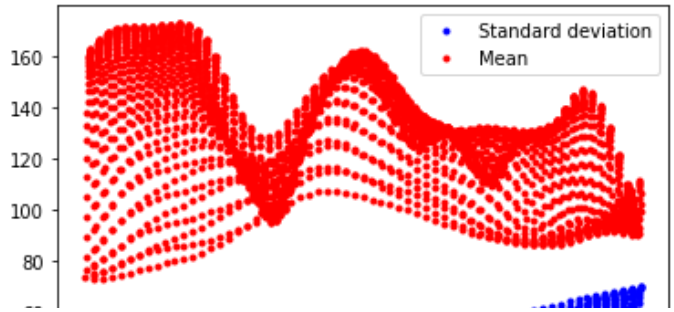
On observe que la diagonale de la matrice de corrélation contient des valeurs de coefficient de corrélation supérieure à 0.8 (Voir la légende à droite). Ceci s'interprete par le fait qu'il va y avoir une grande corrélation entre les features (dans notre cas les pixels) voisins spatialement dans l'image. En effet du fait que les pixels voisins d'une image sont très similaires (surtout pour une grande densité de pixels, i.e image avec une grande résolution) elles vont avoir tendance à varier dans le même sens d'ou une grande corrélation. Ainsi, une prise en compte de la corrélation entre features n'est pas pertinente pour ce type de donnée (image).

Analyse d'échelle

```
X = data_df.drop("target", axis=1)
y = data_df["target"]
print(X.shape, y.shape)
```

```
(13233, 2914) (13233,)
```

```
n_features = X.shape[1]
plot_mean_stdev(X, n_features)
```



Le tracé montre des valeurs de std variant entre environ 80 et 170. Les features sont sur une échelle différente. Pour deux raisons, nous allons réduire l'échelle de notre entrée de 0-255 à 0-1. Premièrement, les features sont intrinsèquement à des échelles différentes. Les pixels sont compris entre 0 et 255. Puisque nous avons affaire à des images, les pixels de chaque image sont aléatoires (ils dépendent de l'image qui sont aléatoires). Deuxièmement, avoir un modèle avec des features à grande échelle implique de grandes valeurs différentielles pendant la formation, ce qui peut conduire très probablement à des valeurs de différentielles exponentielles et à une non convergence de l'apprentissage.

Avant l'entraînement, nous redimensionnerons les valeurs de nos données d'entraînement à la plage 0:1.

Analyse PCA

La grande dimmensionalité de la donnée d'entrée suggère d'appliquer une PCA pour reduire cette dimensionalité.

On lance une PCA sur la totalité des composantes dans un premier temps pour pouvoir estimer la quantité de composantes nécessaire à l'obtention de bons résultats

```
# Tourne sous 15 sec max
pca = PCA()
X_reduced = pca.fit_transform(scale(X.astype(float)))
X_reduced.shape
```

```
(13233, 2914)
```

```
# Observons quelques valeurs des composants principaux.
pd.DataFrame(pca.components_.T).loc[:4,:5]
```

	0	1	2	3	
0	-0.006263	-0.011823	-0.023346	-0.020410	-:
1	-0.006807	-0.011871	-0.024935	-0.019557	-:
2	-0.007408	-0.012074	-0.027195	-0.018455	
3	-0.008141	-0.012527	-0.029697	-0.017895	

```
pd.DataFrame(pca.components_.T).describe()
```

	0	1	2
count	2914.000000	2914.000000	2914.000000
mean	-0.017590	-0.000581	-0.001185
std	0.005813	0.018519	0.018490
min	-0.025732	-0.031052	-0.040360
25%	-0.021877	-0.018331	-0.015563
50%	-0.018932	-0.000561	0.003743
75%	-0.015137	0.017762	0.014358
max	0.002275	0.030235	0.023660

8 rows × 2914 columns

Nous avons plus de 5000 features. Réaliser une cross-validation sur l'ensemble de ces features ne serait pas réalisable en un temps raisonnable. Une autre approche est de calculer le nombre de composantes nécessaire à expliquer 80% de la variance de notre jeu de données.

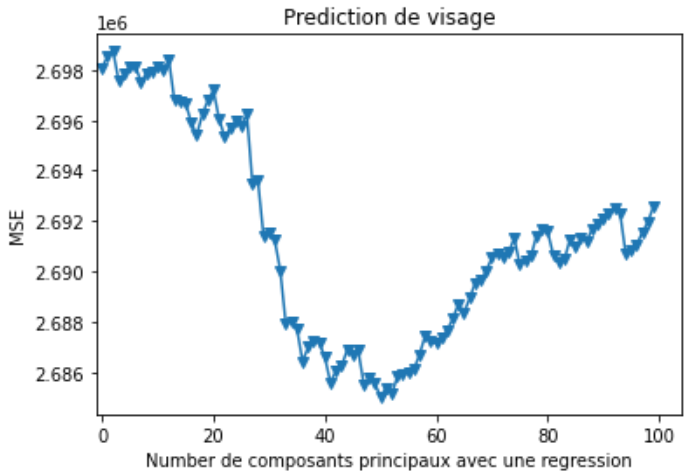
a alassane watt Resolve
16:01 Today
(edited 16:07 Today)
Peut etre mieux commenter cette partie

```
cum_evr = 0.9
n_components = get_ncomponents(pca, cum_evr)
print("Les {} premiers composant principaux expliquent à hauteur de {}% notre jeu d
```

```
Les 90 premiers composant principaux expl
```

Cette analyse nous montre que les 34 premiers principal components explique plus de 80 % de la données. Ceci suggère de projeter les données sur l'espace généré par les 34 premiers principal components et de comparer les résultats sans projection. On va tenter de vérifier cela en réalisant une régression simple avec cross-validation en ajoutant progressivement les 100 premieres composantes principales.

```
n = len(X_reduced)
kf_10 = model_selection.KFold( n_splits=10, shuffle=True, random_state=1)
regr = LinearRegression()
mse = []

# Calcule le MSE avec seulement le intercept (pas de composant principal avec une r
score = -1*model_selection.cross_val_score(regr, np.ones((n,1)), y.ravel(), cv=kf_1
mse.append(score)

# Calcule le MSE en faisant un CV pour les 1000 premiers composants principaux en a
for i in np.arange(1, 100):
    score = -1*model_selection.cross_val_score(regr, X_reduced[:,:i], y.ravel(), cv
    mse.append(score)

# Visualisons les resultats.
plt.plot(mse, '-v')
plt.xlabel('Number de composants principaux avec une regression')
plt.ylabel('MSE')
plt.title('Prediction de visage')
plt.xlim(xmin=-1);
```



On observe que le score optimal est obtenu pour 50 composantes principales, et qu'un score très acceptable (relativement au reste bien sur) est atteint pour 40 composantes. On fait donc le choix d'utiliser ces 40 premières composantes par la suite.

Réalisons maintenant la PCA sur les 40 premières composantes et observons les visages propres correspondant à nos composantes principales

```
n_components = 40
_,h,w = dataset.images.shape
pca = PCA(n_components)
X_reduced = pca.fit_transform(scale(X.astype(float)))

eigenfaces = pca.components_.reshape((n_components, h, w))

plot_gallery(eigenfaces, h, w)

plt.show()
```

On constate visuellement que la projection sur l'espace généré par les 40 composantes principales conserve plus ou moins bien les traits de visage.

Division du dataset

La fonction provide_dataset définie dans la partie **Helper Functions** permet de diviser le dataset.

Exercice 2

Linear SVC

Forward selection

Entrainer une SVM sur la donnée brute ne peut pas etre fait en un temps raisonnable sur nos machines. Nous allons donc essayer de sélectionner les meilleurs features de la donnée avec la technique du *Forward Selection*. On utilisera la technique d'ACP dans l'exercice 3.

```
X, y = provide_dataset(min_faces_per_person=50, scale=False, split=False, balance=T
X.shape
```

```
    2020-11-01 22:23:45,585 NumExpr defaultin
    (624, 2914)
```

```
def processSubset(feature_set, X_train, y_train, X_test, y_test):
  model = LinearSVC(random_state=42)
  model.fit(X_train[feature_set], y_train)
  y_pred_test = model.predict(X_test[feature_set])
  f1_score_ = f1_score(y_pred_test, y_test, average='micro')
  return {"model":model, "f1_score":f1_score_, 'features': feature_set}
```

```
def forwardCV(features, X_train, y_train, X_test, y_test):

  results = []

  scaler = StandardScaler()
  # On redimensionne les jeux d'entrainement et de test.
  X_train_scaled = pd.DataFrame(scaler.fit_transform(X_train))
  X_test_scaled = pd.DataFrame(scaler.transform(X_test.astype(np.float32)))
  # Pull out features we still need to process
  remaining_features = [d for d in X_train_scaled.columns if d not in features]
```

```
    for d in remaining_features:
        results.append(processSubset(features+[d], X_train_scaled, y_train, X_test_sc

    # Wrap everything up in a nice dataframe
    models = pd.DataFrame(results)

    # Choose the model with the highest RSS
    best_model = models.loc[models['f1_score'].argmin()]

    # Return the best model, along with some other useful information about the model
    return best_model



#Pour des contraintes de temps on considere au maximum le meilleur groupe de 50 fea
n_features_considered = 50
np.random.seed(seed=12)
train = np.random.choice([True, False], size = len(y), replace = True)
test = np.invert(train)


models_train = pd.DataFrame(columns=["f1-score", "model", "features"])

features = []

for i in tqdm(range(1,n_features_considered+1)):
    models_train.loc[i] = forwardCV(features, X[train], y[train], X[test], y[test])
    features = models_train.loc[i]["features"]

      0%|          | 0/50 [00:00<?, ?it/s]/Use
    The current behaviour of 'Series.argmin' 
    instead.
    The behavior of 'argmin' will be correcte
    minimum in the future. For now, use 'serie
    'np.argmin(np.array(values))' to get the 
    row.
      2%|▏         | 1/50 [01:07<54:56, 67.28


plt.plot(models_train["f1-score"])
plt.xlabel('# Features')
plt.ylabel('f1-score')
plt.plot(models_train["f1-score"].idxmin()+1, models_train["f1-score"].min(), "or")


# lin_clf = LinearSVC(random_state=42)
# lin_clf.fit(X_train_scaled, y_train)


# y_pred_train = lin_clf.predict(X_train_scaled)
# print("F1 Score training set: {}".format(f1_score(y_train, y_pred_train, average=
# print("Accuracy training set: {}".format(accuracy(y_train, y_pred_train)))

# y_pred_test = lin_clf.predict(X_test_scaled)
# print("F1 Score testing set: {}".format(f1_score(y_test, y_pred_test, average='mi
# print("Accuracy testing set: {}".format(accuracy(y_test, y_pred_test)))
```

Etude de l'influence de quelques parametres

Influence de min_faces_per_person

On fait varier min_faces_per_person entre 20 et 100.
Avant cela on définit quelques helper functions qu'on réutilise

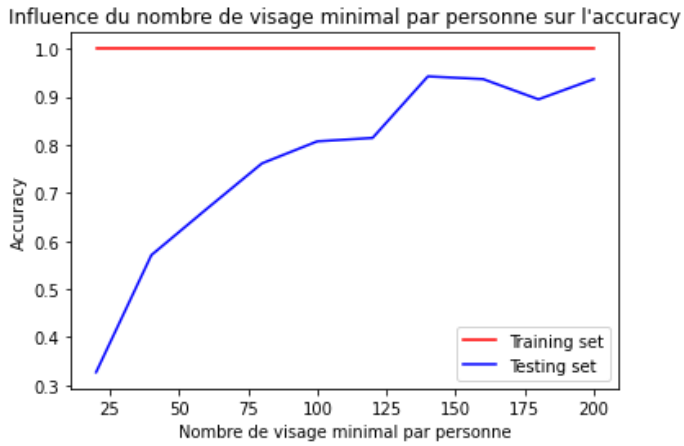
```
def mfpp_influence(min_faces_per_person_values):
    """
    Détermine l'influence du paramètre min_faces_per_person.
    Retourne les mètriques Accuracy et F1-score du jeu d'entrainement et de test
    pour toutes les valeurs de min_faces_per_person_values passé en entrée.

    Paramètres
    ----------
    min_faces_per_person_values: list, requis
        Les valeurs de min_faces_per_person.
    """
    accuracies = []
    f1_scores = []
    for m in min_faces_per_person_values:
        X_train_scaled, X_test_scaled, y_train, y_test = provide_dataset(min_faces_
        lin_clf = LinearSVC(random_state=42)
        acc_, f1_score_ = fit_evaluate(lin_clf, X_train_scaled, X_test_scaled, y_tr
        accuracies.append(acc_)
        f1_scores.append(f1_score_)
    return accuracies, f1_scores
```

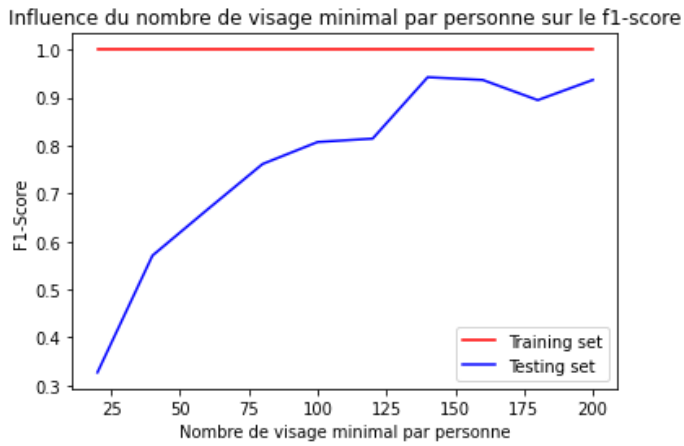
```
min_faces_per_person_values = np.linspace(20, 200, 10)
accuracies, f1_scores = mfpp_influence(min_faces_per_person_values)
```

```
/usr/local/lib/python3.6/dist-packages/sk

Liblinear failed to converge, increase the

/usr/local/lib/python3.6/dist-packages/sk

Liblinear failed to converge, increase the

/usr/local/lib/python3.6/dist-packages/sk

Liblinear failed to converge, increase the
```

```
# On trace l'accuracy pour le training et testing set.
```

```
plot1 = plt.figure(1)
plt.plot(min_faces_per_person_values, [acc[0] for acc in accuracies], '-r', label="
plt.plot(min_faces_per_person_values, [acc[1] for acc in accuracies], '-b', label="

plt.xlabel('Nombre de visage minimal par personne');
plt.ylabel('Accuracy')
plt.title('Influence du nombre de visage minimal par personne sur l\'accuracy')
plt.legend()

# On trace le f1-score pour le training et testing set.
plot2 = plt.figure(2)
plt.plot(min_faces_per_person_values, [f1[0] for f1 in f1_scores], '-r', label="Tra
plt.plot(min_faces_per_person_values, [f1[1] for f1 in f1_scores], '-b', label="Tes

plt.xlabel('Nombre de visage minimal par personne');
plt.ylabel('F1-Score')
plt.title('Influence du nombre de visage minimal par personne sur le f1-score')
plt.legend()

plt.show()
```

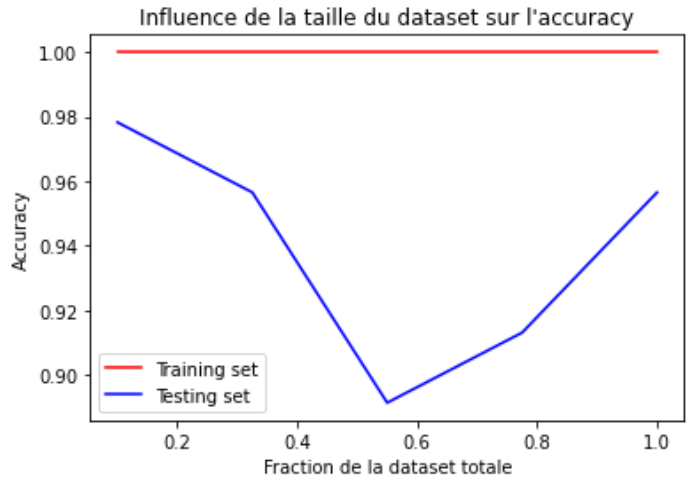


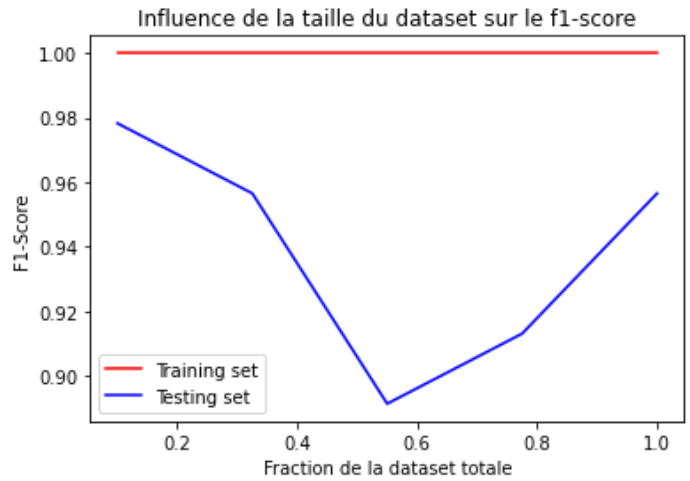


Influence de la taille du dataset

On fixe min_faces_per_person et on joue avec la taille du dataset.

```
def dataset_size_influence(dataset_size_factors, min_faces_per_person=100):
    """
    Détermine l'influence de la taille du jeu de données sur le modèle SVM linéaire
    Retourne les mètriques Accuracy et F1-score du jeu d'entrainement et de test
    pour toutes les valeurs de dataset_size_factors passé en entrée.

    Paramètres
    ----------
    dataset_size_factors: list, requis
        Les valeurs de la fraction de dataset à considérer.
    min_faces_per_person: int, optionnel
        Le parametre min_faces_per_person de la fonction fetch_lfw_people
    """
    accuracies = []
    f1_scores = []
    for f in dataset_size_factors:
        X_train_scaled, X_test_scaled, y_train, y_test = provide_dataset(min_faces_
        lin_clf = LinearSVC(random_state=42)
        acc_, f1_score_ = fit_evaluate(lin_clf, X_train_scaled, X_test_scaled, y_tr
        accuracies.append(acc_)
        f1_scores.append(f1_score_)
    return accuracies, f1_scores


random.seed(1)
dataset_size_factors = np.linspace(0.1, 1, 5)
accuracies, f1_scores = dataset_size_influence(dataset_size_factors)


# On trace l'accuracy pour le training et testing set.
plot1 = plt.figure(1)
plt.plot(dataset_size_factors, [acc[0] for acc in accuracies], '-r', label="Trainin
plt.plot(dataset_size_factors, [acc[1] for acc in accuracies], '-b', label="Testing

plt.xlabel('Fraction de la dataset totale');
plt.ylabel('Accuracy')
plt.title('Influence de la taille du dataset sur l\'accuracy')
plt.legend()

# On trace le f1-score pour le training et testing set.
plot2 = plt.figure(2)
plt.plot(dataset_size_factors, [f1[0] for f1 in f1_scores], '-r', label="Training s
plt.plot(dataset_size_factors, [f1[1] for f1 in f1_scores], '-b', label="Testing se

plt.xlabel('Fraction de la dataset totale');
plt.ylabel('F1-Score')
plt.title('Influence de la taille du dataset sur le f1-score')
plt.legend()

plt.show()
```





On observe donc que les paramètres optimaux en termes de compromis taille / performance sont un paramètre de min_faces_per_person de 150 environs et un size_factor de seulement 0,3. Ceci est tout à fait compréhensible pour le premier paramètre, étant donné que ce sont les visages qui sont peu nombreux qui compliquent le plus l'apprentissage

Exercice 3

```
# On utilisera donc ces paramètres afin d'alléger le temps de calcul.
X_train_scaled, X_test_scaled, y_train, y_test = provide_dataset(min_faces_per_pers
```

```
# On a vu précédemment que les 40 premières composantes principales expliquaient pl
pca2 = PCA(n_components)
X_train_reduced = pca2.fit_transform(X_train_scaled.astype(float))
X_test_reduced = pca2.transform(X_test_scaled.astype(float))
```

```
model = LinearSVC(random_state=42)
acc_pca2, f1_score_pca2, training_time_pca2, avg_pred_time_pca2 =  fit_evaluate(mod

print("F1 Score training set: {}".format(f1_score_pca2[0]))
print("Accuracy training set: {}".format(acc_pca2[0]))
print("F1 Score testing set: {}".format(f1_score_pca2[1]))
print("Accuracy testing set: {}".format(acc_pca2[1]))
print("Training time: {}".format(training_time_pca2))
print("Average prediction time: {}".format(avg_pred_time_pca2))
```

```
F1 Score training set: 1.0
Accuracy training set: 1.0
F1 Score testing set: 0.9285714285714286
Accuracy testing set: 0.9285714285714286
Training time: 0.0048711299896240234
Average prediction time: 5.321843283517026
```

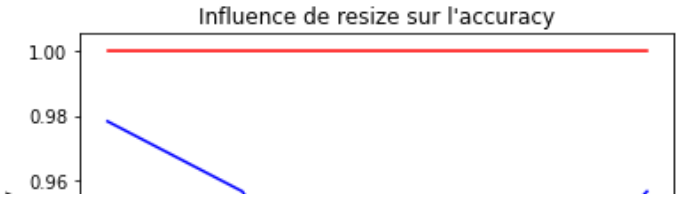
```
# Influence de slice_ = None, resize_ = 1
X_train_scaled, X_test_scaled, y_train, y_test = provide_dataset(min_faces_per_pers


pca3 = PCA(n_components)
X_train_reduced = pca3.fit_transform(X_train_scaled.astype(float))
X_test_reduced = pca3.transform(X_test_scaled.astype(float))


model = LinearSVC(random_state=42)
acc_pca3, f1_score_pca3, training_time_pca3, avg_pred_time_pca3 =  fit_evaluate(mod

print("F1 Score training set: {}".format(f1_score_pca3[0]))
print("Accuracy training set: {}".format(acc_pca3[0]))
print("F1 Score testing set: {}".format(f1_score_pca3[1]))
print("Accuracy testing set: {}".format(acc_pca3[1]))
print("Training time: {}".format(training_time_pca3))
print("Average prediction time: {}".format(avg_pred_time_pca3))
```

```
F1 Score training set: 1.0
Accuracy training set: 1.0
F1 Score testing set: 0.8000000000000002
Accuracy testing set: 0.8
Training time: 0.004204750061035156
Average prediction time: 4.601478576660156
```

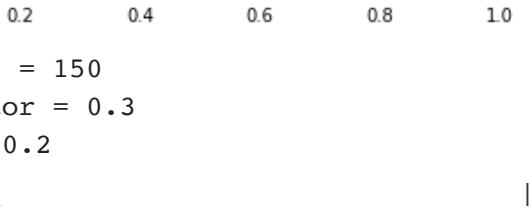
On observe que l'impact de ces deux paramètres est radical sur la performance. Cela mérite qu'on cherche à optimiser la valeur de resize

```
resize_values = np.linspace(0.1, 1, 5)
accuracies = []
f1_scores = []
for r in resize_values:
```

```
        X_train_scaled, X_test_scaled, y_train, y_test = provide_dataset(min_faces_per_
        lin_clf = LinearSVC(random_state=42)
        acc_, f1_score_ = fit_evaluate(lin_clf, X_train_scaled, X_test_scaled, y_train,
        accuracies.append(acc_)
        f1_scores.append(f1_score_)



    # On trace l'accuracy pour le training et testing set.
    plot1 = plt.figure(1)
    plt.plot(resize_values, [acc[0] for acc in accuracies], '-r', label="Training set")
    plt.plot(resize_values, [acc[1] for acc in accuracies], '-b', label="Testing set")

    plt.xlabel('Valeur de resize');
    plt.ylabel('Accuracy')
    plt.title('Influence de resize sur l\'accuracy')
    plt.legend()

    # On trace le f1-score pour le training et testing set.
    plot2 = plt.figure(2)
    plt.plot(resize_values, [f1[0] for f1 in f1_scores], '-r', label="Training set")
    plt.plot(resize_values, [f1[1] for f1 in f1_scores], '-b', label="Testing set")

    plt.xlabel('Valeur de resize');
    plt.ylabel('F1-Score')
    plt.title('Influence de resize sur le f1-score')
    plt.legend()

    plt.show()
```



On observe que l'exactitude sur le testing set souffre fortement lorsque le facteur resize augmente. Sa valeur optimale semble se trouver vers 0,35 On définit nos valeurs optimales pour plus tard:




```
best_min_faces = 150
best_size_factor = 0.3
best_resize = 0.2
```


Exercice 4

Naive Bayes

Nous ne pouvons pas utiliser de données centrées ici, car le Naive Bayes ne supporte pas les données négatives Pas d'ACP ici donc. Pour conserver un temps d'exécution correct, nous allons fixer le nombre de visages par personne à 100, ce qui réduira l'échantillon.

Ici, la supposition à priori d'une indépendance entre les features prises deux à deux est fausse, sachant que tout le concept de nos images est qu'elles forment des visages par association de pixels.

```
X_train, X_test, y_train, y_test = provide_dataset(min_faces_per_person=best_min_fa

print("Number of mislabeled faces out of total")
gnb = GaussianNB()
y_pred = gnb.fit(X_train, y_train).predict(X_test)
print("Gaussian: {} faces : {} misslabeled ({} %)".format(X_test.shape[0], (y_test

mnb = MultinomialNB()
y_pred = mnb.fit(X_train, y_train).predict(X_test)
print("Multinomial: {} faces : {} misslabeled ({} %)".format(X_test.shape[0], (y_te

cnb = ComplementNB()
y_pred = cnb.fit(X_train, y_train).predict(X_test)
print("Complement: {} faces : {} misslabeled ({} %) \n".format(X_test.shape[0], (y_
```

```
print("Affichons plus en détail le score du Naive Bayes le plus performant: \n ")
print(classification_report(y_test, y_pred))
```

```
Number of mislabeled faces out of total
Gaussian: 46 faces : 13 misslabeled (28.0
Multinomial: 46 faces : 13 misslabeled (2
Complement: 46 faces : 13 misslabeled (28

Affichons plus en détail le score du Naiv

              precision    recall  f1-sco

           0       0.53      0.71      0.
           1       0.85      0.72      0.

    accuracy                           0.
   macro avg       0.69      0.72      0.
weighted avg       0.75      0.72      0.
```

Accuracy de 0,72 pour le ComplementNB

SVM with Polynomial Kernel

Dans un premier temps, préparons le jeu de données qui va nous servir pour expérimenter avec les SVM à noyaux. Cette fois, on peut utiliser une ACP

```
X_train_scaled, X_test_scaled, y_train, y_test = provide_dataset(min_faces_per_pers
pca = PCA(n_components)
X_train_reduced = pca.fit_transform(X_train_scaled.astype(float))
X_test_reduced = pca.transform(X_test_scaled.astype(float))
```

D'abord, faisons une grid search sur une portion de la donnée afin de déterminer les hyperparamètres optimaux.

```
param_grid = {"gamma": reciprocal(0.0001, 0.1), "C": uniform(1, 100), "degree": uni
poly_kernel_svm = SVC(kernel="poly", degree=2, coef0=1)

rnd_search_cv = RandomizedSearchCV(poly_kernel_svm, param_grid, n_iter=40, verbose=
rnd_search_cv.fit(X_train_reduced, y_train)

rnd_search_cv.best_estimator_.fit(X_train_reduced, y_train)
y_pred = rnd_search_cv.best_estimator_.predict(X_test_reduced)

print(classification_report(y_test, y_pred))
```

```
Fitting 5 folds for each of 40 candidates
[Parallel(n_jobs=2)]: Using backend LokyBa
              precision    recall  f1-sco

           0       0.91      0.83      0.8
           1       0.94      0.97      0.9

    accuracy                           0.9
   macro avg       0.93      0.90      0.9
weighted avg       0.93      0.93      0.9

[Parallel(n_jobs=2)]: Done 200 out of 200
```

Accuracy de 0,90

SVM with RBF Kernel

Même chose pour le noyau RBF

```
param_grid = {"gamma": reciprocal(0.0001, 0.1), "C": uniform(1, 1000)}
rbf_kernel_svm = SVC(kernel='rbf')

rnd_search_cv = RandomizedSearchCV(rbf_kernel_svm, param_grid, n_iter=20, verbose=1
rnd_search_cv.fit(X_train_reduced, y_train)

rnd_search_cv.best_estimator_.fit(X_train_reduced, y_train)
y_pred = rnd_search_cv.best_estimator_.predict(X_test_reduced)

print(classification_report(y_test, y_pred))
```

```
[Parallel(n_jobs=2)]: Using backend LokyBa
Fitting 5 folds for each of 20 candidates
              precision    recall  f1-sco

           0       1.00      0.83      0.9
           1       0.94      1.00      0.9

    accuracy                           0.9
   macro avg       0.97      0.92      0.9
weighted avg       0.96      0.96      0.9

[Parallel(n_jobs=2)]: Done 100 out of 100
```

Accuracy de 0,93

Arbres de Décision

Expérimentons d'abord sur le jeu de données brut
sans restreindre la taille de l'arbre.

```
X_train_scaled, X_test_scaled, y_train, y_test = provide_dataset(min_faces_per_pers

tree_clf = tree.DecisionTreeClassifier()
tree_clf.fit(X_train, y_train)
y_pred = tree_clf.predict(X_test)

print(classification_report(y_test, y_pred))
```

```
                  precision    recall  f1-sco

               0       0.36      0.31      0.
               1       0.66      0.70      0.

        accuracy                           0.
       macro avg       0.51      0.51      0.
    weighted avg       0.55      0.57      0.
```

Accuracy de 0,51

Réalisons maintenant une cross-validation avec GridSearch sur un dataset plus petit afin d'identifier les hyperparamètres optimaux

```
pgrid = {"max_depth": [1, 2, 3, 4, 5, 6, 7],
      "min_samples_split": [2, 3, 5, 10, 15, 20]}

grid_search = GridSearchCV(tree.DecisionTreeClassifier(), param_grid=pgrid, cv=10,
grid_search.fit(X_train, y_train)
y_pred = grid_search.best_estimator_.predict(X_test)

# On récupère les paramètres optimaux
best_max_depth, best_min_samples_split = grid_search.best_params_["max_depth"], gri

print(classification_report(y_test, y_pred))
```

```
                  precision    recall  f1-sco

               0       0.25      0.06      0.
               1       0.64      0.90      0.

        accuracy                           0.
       macro avg       0.45      0.48      0.
    weighted avg       0.51      0.61      0.
```

Accuracy de 0,65

Bagging sur des arbres de décision

Utilisons désormais le Bagging pour augmenter la précision de notre modèle tout en réduisant sa variance. Pour cela, le bagging met en commun les apprentissages de nombreux weak learners

Trois hyperparamètres jouent ici: le nombre de samples max, le nombre de features max et le nombre d'estimateurs On va dans un premier temps sélectionner les valeurs optimales des deux premiers paramètres par gridsearch avant De selectionner le nombre d'estimateur optimal en visualisant graphiquement la précision pour des valeurs croissantes. On devrait voir qu'après un certain nombre d'estimateurs cette valeur se stabilise.

```
grid_parameters ={"max_samples": [0.2, 0.4, 0.6, 0.8],"max_features": [0.2, 0.4, 0.
grid_search = GridSearchCV(BaggingClassifier(tree.DecisionTreeClassifier(max_depth=
grid_search.fit(X_train, y_train)
y_pred = grid_search.best_estimator_.predict(X_test)

print(classification_report(y_test, y_pred))

# Récupérons maintenant les paramètres optimaux et utilisons les pour déterminer le
best_max_samples, best_max_features = grid_search.best_params_["max_samples"], grid

x = [10*(i+1) for i in range(30)]
accuracy=[0  for i in range(30)]

for n in x:
    bag_clf = BaggingClassifier(tree.DecisionTreeClassifier(max_depth=best_max_dept
    bag_clf.fit(X_train, y_train)
    y_pred = bag_clf.predict(X_test)
    accuracy[int((n/10)-1)]=bag_clf.score(X_test,y_test)

plt.plot(x,accuracy)
plt.show()
```

```
                  precision    recall  f1-sco

               0       0.50      0.12      0.
               1       0.67      0.93      0.

        accuracy                           0.
       macro avg       0.58      0.53      0.
    weighted avg       0.61      0.65      0.
```

On voit que la convergence de l'accuracy se fait dès 50 estimateurs et oscille autour d'une valeur moyenne de 0,72

Entrainons donc notre modèle optimal:

```
bag_clf = BaggingClassifier(tree.DecisionTreeClassifier(),max_samples=best_max_samp
bag_clf.fit(X_train, y_train)
y_pred = bag_clf.predict(X_test)
print(classification_report(y_test, y_pred))
```

```
                  precision    recall  f1-sco

               0       0.40      0.25      0.
               1       0.67      0.80      0.

        accuracy                           0.
       macro avg       0.53      0.53      0.
    weighted avg       0.57      0.61      0.
```

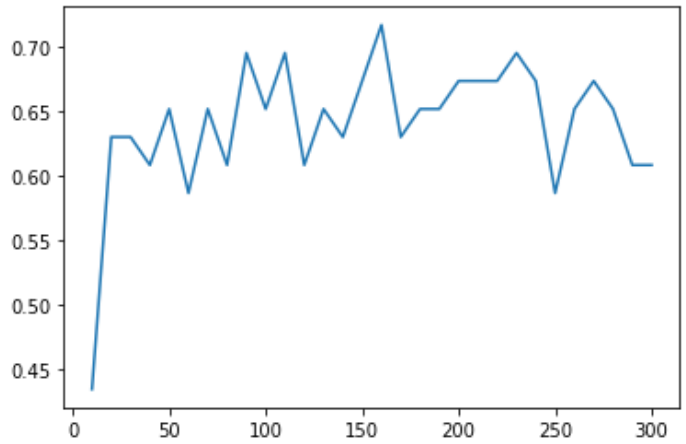
Accuracy de 0.72

Forêt Aléatoire

Etant donné que les forêts aléatoires utilisent un bagging d'arbres de décision randomisés, nous devrions obtenir des Résultats légèrement meilleurs ici. Affichons l'accuracy pour un nombre d'estimateurs variant entre 0 et 300 afin de déterminer l'optimum

```
x = [10*(i+1) for i in range(30)]
accuracy=[0  for i in range(30)]

for n in x:
    rf_clf = RandomForestClassifier(n_estimators=n)
```

```
    rf_clf.fit(X_train, y_train)
    y_pred = rf_clf.predict(X_test)
    accuracy[int((n/10)-1)]=rf_clf.score(X_test,y_test)

plt.plot(x,accuracy)
plt.show()
```

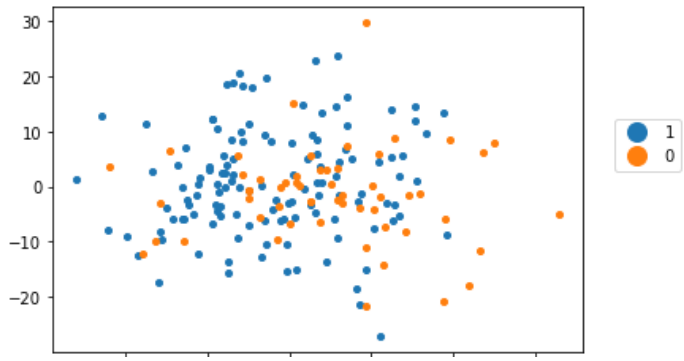


Exercice 5 : Tests

KMeans

On va désormais tenter de classifier notre jeu de données avec la méthode K-Means. Dans un premier temps, on va réduire en 2D pour visualiser. On prend donc nos 2 premières composantes principales.

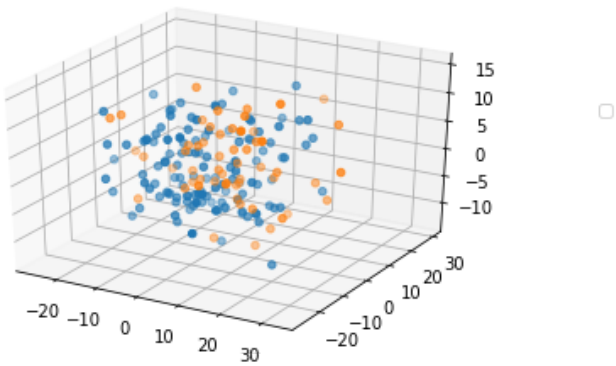
```
X_train_scaled, X_test_scaled, y_train, y_test = provide_dataset(min_faces_per_pers

pca_2D = PCA(2)
X_train_reduced = pca.fit_transform(X_train_scaled.astype(float))
X_test_reduced = pca.transform(X_test_scaled.astype(float))

classes = y_train.unique()

for c in classes:
    Xc_train_reduced = X_train_reduced[y_train == c]
    plt.plot(Xc_train_reduced[:, 0], Xc_train_reduced[:, 1], '.', markersize=8, lab
    plt.legend(numpoints=1,loc=1, bbox_to_anchor=(1.2, 0.7), markerscale=3)
```



On voit bien que sans surprise, aucune classe n'est linéairement séparable. On va essayer d'augmenter légèrement la dimension.

```
from mpl_toolkits.mplot3d import Axes3D

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
pca_3D = PCA(3)
X_train_reduced = pca.fit_transform(X_train_scaled.astype(float))
X_test_reduced = pca.transform(X_test_scaled.astype(float))

classes = y_train.unique()

for c in classes:
    Xc_train_reduced = X_train_reduced[y_train == c]
    ax.scatter(Xc_train_reduced[:, 0], Xc_train_reduced[:, 1], Xc_train_reduced[:,
    ax.legend(numpoints=1,loc=1, bbox_to_anchor=(1.2, 0.7), markerscale=3)
```

```
2020-11-01 22:36:29,390 No handles with la
2020-11-01 22:36:29,397 No handles with la
```



Il semble y avoir un peu plus d'espoir de faire un clustering visualisable ici

On va tenter d'appliquer le bagging au SVM, qui est pour le moment notre meilleur modèle en termes de

```
X_train_reduced = pca.fit_transform(X_train_scaled.astype(float))
X_test_reduced = pca.transform(X_test_scaled.astype(float))

param_grid = {"gamma": reciprocal(0.0001, 0.1), "C": uniform(1, 100), "degree": uni
poly_kernel_svm = SVC(kernel="rbf", coef0=1)

rnd_search_cv = RandomizedSearchCV(poly_kernel_svm, param_grid, n_iter=40, verbose=
rnd_search_cv.fit(X_train_reduced, y_train)

gamma, C, degree = rnd_search_cv.best_params_["gamma"], rnd_search_cv.best_params_[

rnd_search_cv.best_estimator_.fit(X_train_reduced, y_train)
y_pred = rnd_search_cv.best_estimator_.predict(X_test_reduced)

print(classification_report(y_test, y_pred))

grid_parameters ={"max_samples": [0.2, 0.4, 0.6, 0.8],"max_features": [0.2, 0.4, 0.
grid_search = GridSearchCV(BaggingClassifier(SVC(kernel="rbf", gamma=gamma, C=C, de
grid_search.fit(X_train_reduced, y_train)
y_pred = grid_search.best_estimator_.predict(X_test_reduced)

print(classification_report(y_test, y_pred))

# Récupérons maintenant les paramètres optimaux et utilisons les pour déterminer le
best_max_samples, best_max_features = grid_search.best_params_["max_samples"], grid

x = [10*(i+1) for i in range(30)]
accuracy=[0  for i in range(30)]

for n in x:
    bag_clf = BaggingClassifier(SVC(kernel="rbf", gamma=gamma, C=C, degree=degree,
    bag_clf.fit(X_train_reduced, y_train)
    y_pred = bag_clf.predict(X_train_reduced)
    accuracy[int((n/10)-1)]=bag_clf.score(X_test_reduced,y_test)

plt.plot(x,accuracy)
plt.show()
```

```
Fitting 5 folds for each of 40 candidates
[Parallel(n_jobs=8)]: Using backend LokyB
[Parallel(n_jobs=8)]: Done  60 tasks
[Parallel(n_jobs=8)]: Done 200 out of 200
              precision    recall  f1-sco

           0       0.85      0.85      0.
           1       0.94      0.94      0.

    accuracy                           0.
   macro avg       0.89      0.89      0.
weighted avg       0.91      0.91      0.

              precision    recall  f1-sco

           0       0.92      0.85      0.
           1       0.94      0.97      0.

    accuracy                           0.
   macro avg       0.93      0.91      0.
weighted avg       0.93      0.93      0.
```

Entrainons donc notre modèle optimal:

```
      |    \  |  \ / \ /    | | \ /           |
```

```
bag_clf = BaggingClassifier(SVC(kernel="rbf", gamma=gamma, C=C, degree=degree, coef
bag_clf.fit(X_train_reduced, y_train)
y_pred = bag_clf.predict(X_test_reduced)
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-sco

           0       1.00      0.85      0.
           1       0.94      1.00      0.

    accuracy                           0.
   macro avg       0.97      0.92      0.
weighted avg       0.96      0.96      0.
```

Accuracy de 0.96