Méthodes d'ensemble

TP3 - Classification des chiffres manuscrite avec Adaboost

LE Do Thanh Dat, YOU Borachhun

1. Importer des libraires et les données d'apprentissage

In [70]:
```
from sklearn.datasets import load_digits
import matplotlib.pyplot as plt
import numpy as np

X, y = load_digits(return_X_y=True)
X.shape
```

Out[70]:
```
(1797, 64)
```

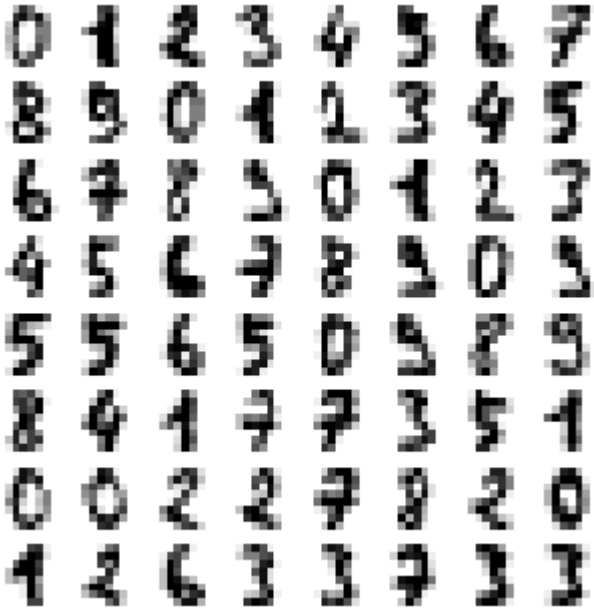
2. Visualiser quelques exemples

In [71]:
```
fig, ax = plt.subplots(nrows=1, ncols=1, figsize=(4, 4))

n_img_per_row = 8
img = np.zeros((10 * n_img_per_row, 10 * n_img_per_row))
for i in range(n_img_per_row):
    ix = 10 * i + 1
    for j in range(n_img_per_row):
        iy = 10 * j +1
        img[ix:ix + 8, iy:iy + 8] = X[i * n_img_per_row + j].reshape((8, 8))

ax.imshow(img, cmap=plt.cm.binary)
ax.set_xticks([])
ax.set_yticks([])
ax.axis('off')
```

Out[71]:
```
(-0.5, 79.5, 79.5, -0.5)
```



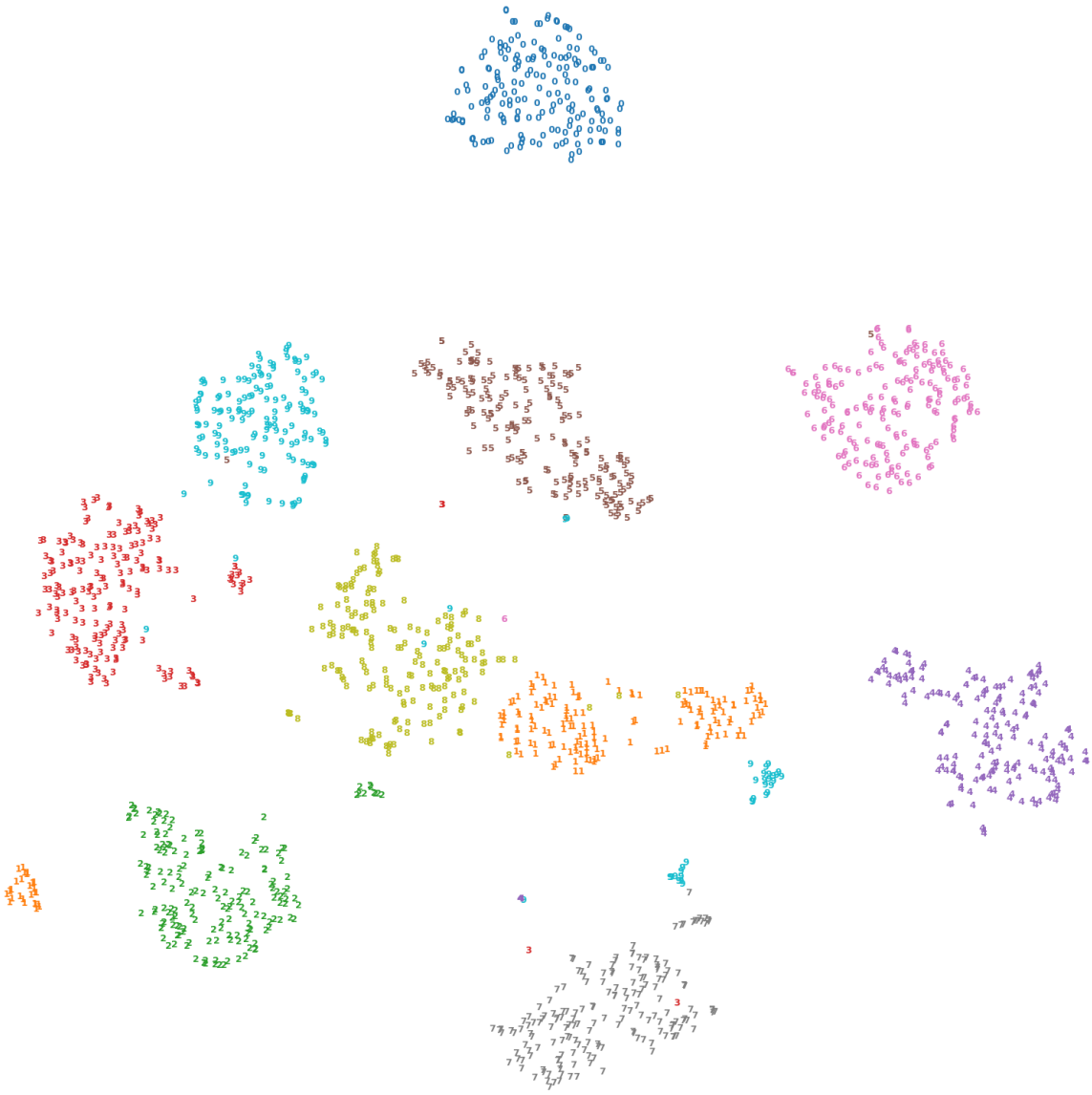
3. Réduction de dimensionnalité avec t-SNE

In [72]:
```
from sklearn.manifold import TSNE
Xemb = TSNE(n_components=2).fit_transform(X)

%matplotlib inline

fig, ax = plt.subplots(nrows=1, ncols=1, figsize=(15, 15))
xMin, xMax = np.min(Xemb, axis=0), np.max(Xemb, axis=0)
Xemb = (Xemb - xMin) / (xMax - xMin)
for i in range(Xemb.shape[0]):
    plt.text(Xemb[i, 0], Xemb[i, 1], str(y[i]), color=plt.cm.tab10(y[i] / 10.),
             fontdict={'weight': 'bold', 'size': 9})

ax.set_xticks([])
ax.set_yticks([])
ax.axis('off')
fig.tight_layout()
```



4. Définir base d'apprentissage et test

In [73]:
```
from sklearn.model_selection import train_test_split
Xtrn, Xtst, ytrn, ytst = train_test_split(Xemb, y, test_size=0.2, stratify=y)
```

Le paramètre ***stratify*** fait une division de sorte que la proportion de la cible dans l'ensemble d'entraînement et l'ensemble de test soit la même que la proportion de la cible dans l'ensemble de données d'origine.

Par exemple, l'ensemble de données d'origine a la cible **y** avec 3 classes $[0, 1, 2]$ avec le rapport 60:20:20, respectivement. Ensuite, si le paramètre ***stratify = y***, la proportion de 3 classes $[0, 1, 2]$ dans l'ensemble d'apprentissage et l'ensemble de test sera de 60:20:20 (la même proportion que les données d'origine)

Le paramètre ***stratify*** est important pour une classification parce qu'il préserve la distribution de toutes les classes dans l'ensemble d'entraînement et de test. Ce qui permet de réduire les biais du modèle et d'améliorer la précision et l'équité des prédictions du modèle.

5. Définir un apprenant faire pour l'Adaboost

In [74]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.metrics import accuracy_score

depth = np.arange(1, 10, 1)
trn_err = []
val_err = []

for i in depth:
    #using decision stumps
    stump = DecisionTreeClassifier(max_depth = i)
    ensemble = AdaBoostClassifier(estimator=stump)
    ensemble.fit(Xtrn, ytrn)

    #calculating error
    trn_err.append(1-accuracy_score(ytrn, ensemble.predict(Xtrn)))
    val_err.append(1-accuracy_score(ytst, ensemble.predict(Xtst)))
```

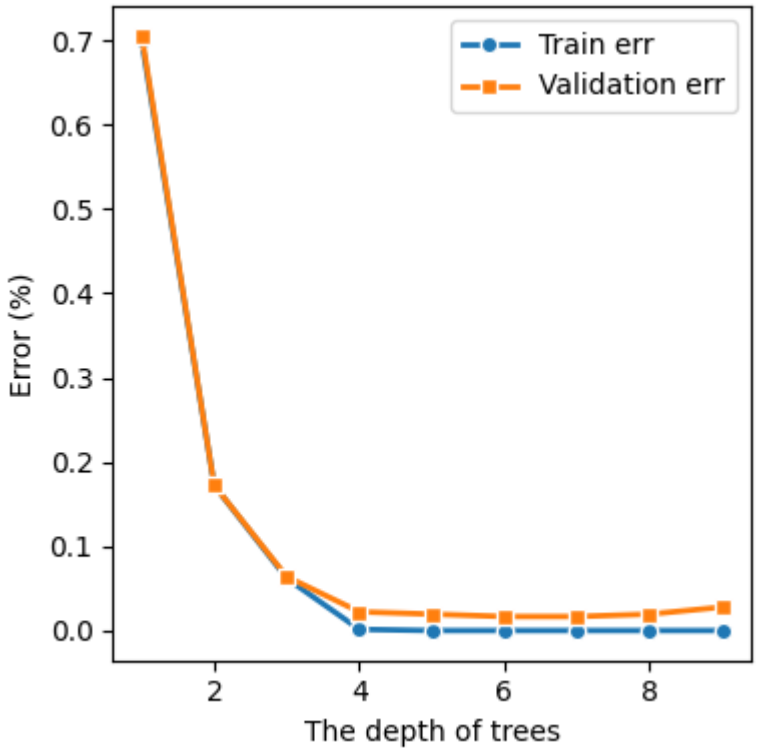
In [75]:
```
print(trn_err)
print(val_err)
```

```
[0.6993736951983298, 0.17188587334725125, 0.06193458594293666, 0.00139178844815
59062, 0.0, 0.0, 0.0, 0.0, 0.0]
[0.7055555555555555, 0.17222222222222228, 0.06388888888888888, 0.02222222222222
2254, 0.019444444444444486, 0.01666666666666672, 0.01666666666666672, 0.0194444
44444444486, 0.02777777777777779]
```

In [76]:
```
# Visualisation the results
fig, ax = plt.subplots(nrows=1, ncols=1, figsize=(4, 4))

ax.plot(depth, trn_err, marker='o', markeredgecolor='w', linewidth=2)
ax.plot(depth, val_err, marker='s', markeredgecolor='w', linewidth=2)
ax.legend(['Train err', 'Validation err'])
ax.set_xlabel('The depth of trees')
ax.set_ylabel('Error (%)')

fig.tight_layout()
```



Concernant le graphique ci-dessus, la performance augmente lorsque l'on augmente la profondeur d'arbre. Cependant, si la profondeur est trop élevée, les performances diminueront dans l'ensemble de validation. C'est à cause du sur-apprentissage (overfitting).

6. Définir les hyperparamètres à optimiser

On propose des hyperparamètres tels que **n_estimator** (nombre d'arbres), **learning_rate**, **max_depth** (la profondeur de l'arbre), **criterion** ('gini' ou 'entropy' pour l'arbre de décision), et **algorithm** ('SAMME' ou 'SAMME.R')

In [92]:
```
parameters_to_search = {'n_estimators': [200, 300, 400],
                        'learning_rate': [0.6, 0.8, 1.0],
                        'estimator__max_depth': [1, 2, 3, 4, 5],
                        'estimator__criterion' : ['gini', 'entropy'],
                        'algorithm': ['SAMME', 'SAMME.R']}
```

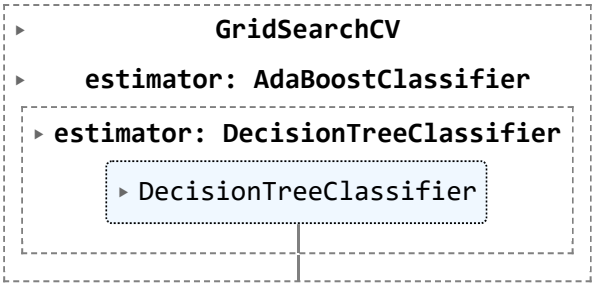
7. Modélisation avec GridSearch

In [93]:
```
ensemble = AdaBoostClassifier(estimator=DecisionTreeClassifier())

#Scoring function
from sklearn.metrics import balanced_accuracy_score, make_scorer
scorer = make_scorer(balanced_accuracy_score, greater_is_better=True)

#gridsearch modelling
from sklearn.model_selection import GridSearchCV
search = GridSearchCV(ensemble, param_grid=parameters_to_search,
                      scoring=scorer, cv=5, n_jobs=-1, refit=True)  #5-fold cross
search.fit(Xtrn, ytrn)
```

Out[93]:



8. Visualisation des résultats

In [94]:
```
#Best parameter setting
best_combo = search.cv_results_['params'][search.best_index_]
best_score = search.best_score_
print('The best parameter settings are {0}, with score = {1}.'.format(best_combo

#The best model is available is search.best_estimator_ and can be used for makin
ypred = search.best_estimator_.predict(Xtst)

from sklearn.metrics import confusion_matrix
print('Confusion matrix: \n {0}'.format(confusion_matrix(ytst, ypred)))
```

```
The best parameter settings are {'algorithm': 'SAMME.R', 'estimator__criterio
n': 'gini', 'estimator__max_depth': 5, 'learning_rate': 0.8, 'n_estimators': 40
0}, with score = 0.9915763546798029.
Confusion matrix:
 [[36  0  0  0  0  0  0  0  0  0]
 [ 0 35  0  0  0  0  0  0  1  0]
 [ 0  0 35  0  0  0  0  0  0  0]
 [ 0  0  0 36  0  0  0  1  0  0]
 [ 0  0  0  0 36  0  0  0  0  0]
 [ 0  0  0  0  0 36  1  0  0  0]
 [ 0  0  0  0  0  0 35  0  1  0]
 [ 0  0  0  0  0  0  0 35  0  1]
 [ 0  0  0  0  0  0  0  0 34  1]
 [ 0  0  0  1  0  0  0  0  0 35]]
```

Après avoir utilisé GridSearchCV, on obtient les meilleurs réglages de paramètres pour ce modèle. Lors de l'utilisation de ces paramètres pour prédire sur l'ensemble de test, on obtient la matrice de confusion. On peut voir que tous les éléments sur la diagonale principale de cette matrice de confusion sont élevés (34-36) et que les autres éléments de cette matrice sont 0 ou 1. Alors, la performance de ce modèle est élevée.

9. Visualisation des frontières de décision

In [96]:
```
# Visualize the decision boundary
xMin, xMax = Xemb[:, 0].min(), Xemb[:, 0].max() + 0.05
yMin, yMax = Xemb[:, 1].min(), Xemb[:, 1].max() + 0.05
xMesh, yMesh = np.meshgrid(np.arange(xMin, xMax, 0.05),
                           np.arange(yMin, yMax, 0.05))

fig, ax = plt.subplots(nrows=1, ncols=1, figsize=(15, 15))
zMesh = search.best_estimator_.predict(np.c_[xMesh.ravel(), yMesh.ravel()])
zMesh = zMesh.reshape(xMesh.shape) * 1.0
boundary = ax.contourf(xMesh, yMesh, zMesh, np.array([-1, 0, 1, 2, 3, 4, 5, 6, 7
                       cmap=plt.cm.tab10, alpha=0.5)
```

```
for i in range(X.shape[0]):
    plt.text(Xemb[i, 0], Xemb[i, 1], str(y[i]), color=plt.cm.tab10(y[i] / 10.),
             fontdict = {'weight': 'bold', 'size': 9})

ax.axis('off')

# fig.colorbar(boundary)
fig.tight_layout()
```