9v6okdfz3

March 10, 2025

[1]:
```
#!pip install kagglehub#!pip install matplotlib
#!pip install -U scikit-learn
#!pip install matplotlib
#!pip install -U scikit-learn
#!pip install numpy
#!pip install pandas
#!pip install scikit-image
#!pip install imblearn
```

[2]:
```
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from tqdm import tqdm
from skimage import io, transform, feature
from sklearn.model_selection import train_test_split, StratifiedKFold
from sklearn.svm import SVC
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.utils.multiclass import unique_labels
from imblearn.over_sampling import SMOTE
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import GridSearchCV

# Configurations
dataset_root = "/Users/markus/.cache/kagglehub/datasets/pulavendranselvaraj/
↪oasis-dataset/versions/1/input"
classes = ["Non Demented", "Very mild Dementia", "Mild Dementia", "Moderate␣
↪Dementia"]
```

[3]:
```
# Chargement des données
X, y = [], []
for class_name in classes:
    class_path = os.path.join(dataset_root, class_name)
    for filename in tqdm(os.listdir(class_path), desc=f"Loading {class_name}"):
        try:
            img_path = os.path.join(class_path, filename)
            img = io.imread(img_path, as_gray=True)
```

```
            img = transform.resize(img, (64, 64), anti_aliasing=True)␣
↪#Redimensionnement
            img = img / 255.0 #Normalisation
            fd = feature.hog(img, orientations=8, pixels_per_cell=(16, 16),
                            cells_per_block=(1, 1), channel_axis=None)␣
↪#Extraction des caractéristiques HOG
            X.append(fd)
            y.append(class_name)
        except Exception as e:
            print(f"Error loading {img_path}: {e}")
```

```
Loading Non Demented: 100%|          | 3000/3000 [00:38<00:00, 78.68it/s]
Loading Very mild Dementia: 100%|      | 3000/3000 [00:37<00:00, 79.10it/s]
Loading Mild Dementia: 100%|          | 3000/3000 [00:38<00:00, 77.54it/s]
Loading Moderate Dementia: 100%|        | 488/488 [00:06<00:00, 77.21it/s]
```

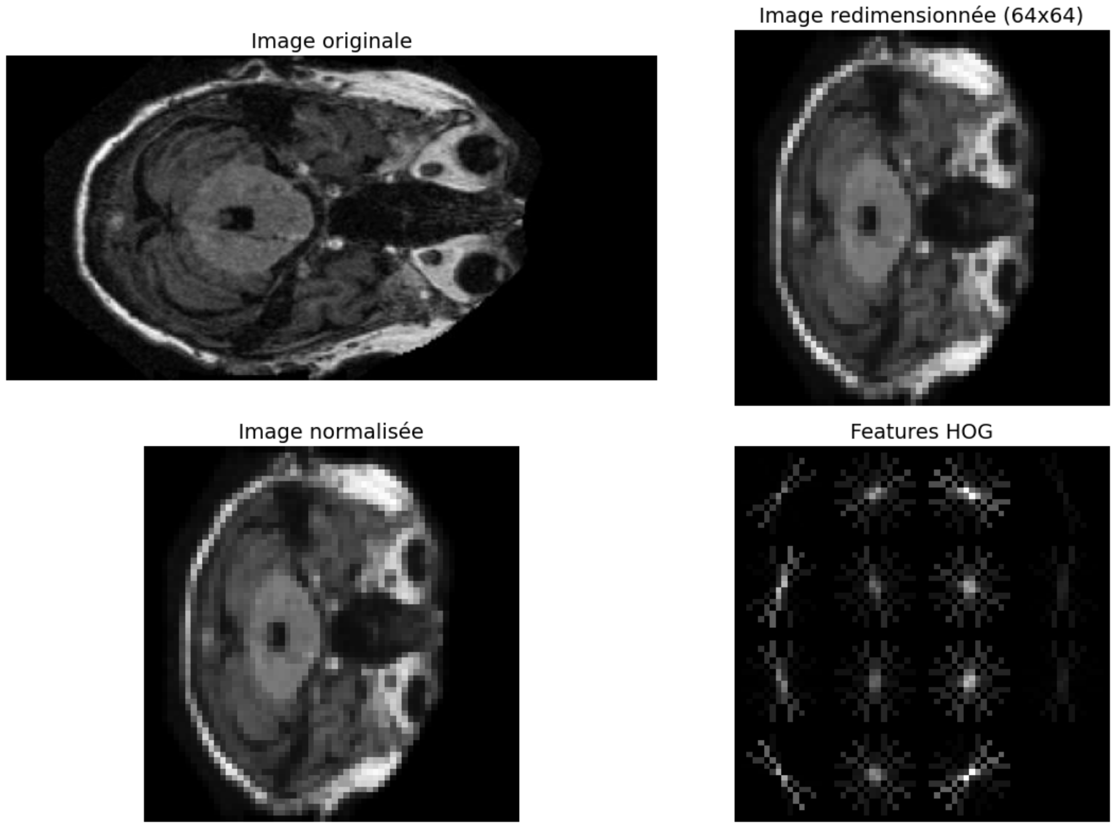
[4]:
```
# Vérification de classes chargées
print(f"\n Données chargées")
print(f"Labels uniques : {np.unique(y)}")
print(f"Nombre d'échantillons : {len(X)}")
```

```
 Données chargées
Labels uniques : ['Mild Dementia' 'Moderate Dementia' 'Non Demented' 'Very mild
Dementia']
Nombre d'échantillons : 9488
```

[5]:
```
# Chemin de l'image testée
img_path = "/Users/markus/.cache/kagglehub/datasets/pulavendranselvaraj/
↪oasis-dataset/versions/1/input/Mild Dementia/OAS1_0028_MR1_mpr-1_103.jpg"

# Chargement de l'image en niveaux de gris
img = io.imread(img_path, as_gray=True)

# Redimensionnement à 64x64 avec anti-aliasing
img_resized = transform.resize(img, (64, 64), anti_aliasing=True)

# Normalisation
img_normalized = img_resized / 255.0

# Extraction des features HOG et visualisation
fd, hog_image = feature.hog(img_normalized, orientations=8,␣
↪pixels_per_cell=(16, 16),
                            cells_per_block=(1, 1), visualize=True,␣
↪channel_axis=None)
```

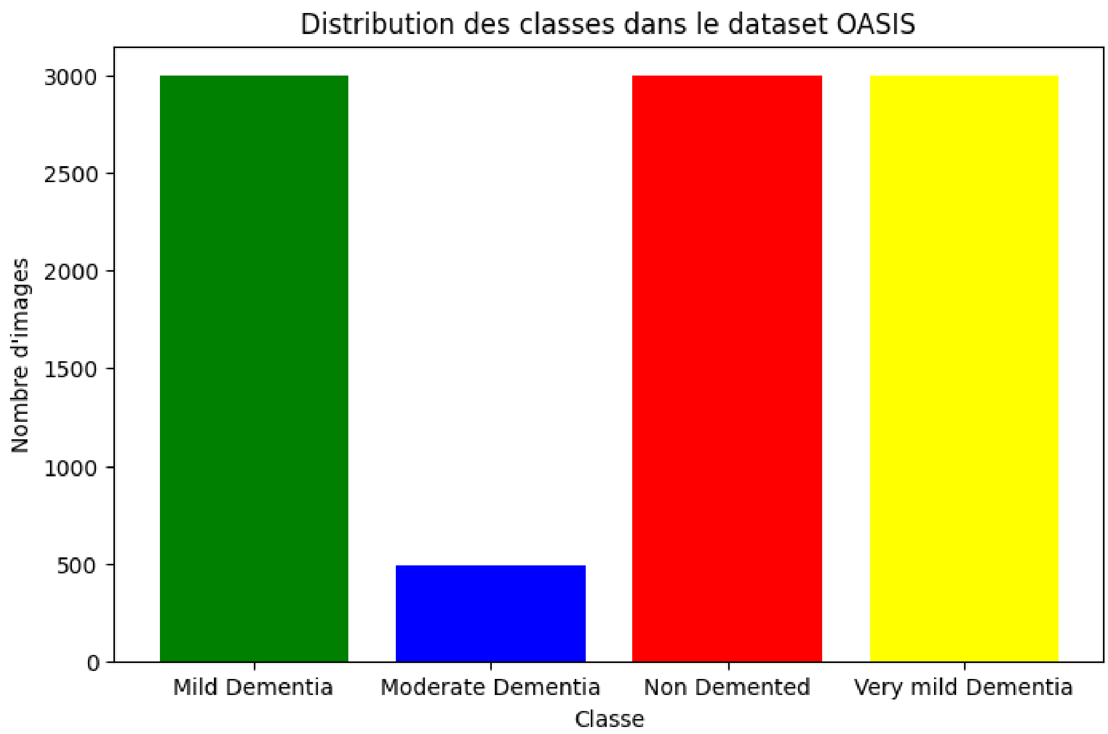
[6]:

```python
# Affichage des résultats
fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2, figsize=(12, 8))

# Image originale
ax1.imshow(img, cmap='gray')
ax1.set_title("Image originale", fontsize=14)
ax1.axis('off')

# Image redimensionnée
ax2.imshow(img_resized, cmap='gray')
ax2.set_title("Image redimensionnée (64x64)", fontsize=14)
ax2.axis('off')

# Image normalisée
ax3.imshow(img_normalized, cmap='gray')
ax3.set_title("Image normalisée", fontsize=14)
ax3.axis('off')

# Visualisation HOG
ax4.imshow(hog_image, cmap='gray')
ax4.set_title("Features HOG", fontsize=14)
ax4.axis('off')

plt.tight_layout()
plt.show()
```



[7]:
```
# Analyse de la distribution des classes
unique_classes, class_counts = np.unique(y, return_counts=True)
print("\n Distribution des classes ")
for cls, count in zip(unique_classes, class_counts):
    print(f"Classe: {cls} | Nombre d'exemples: {count}")

# Visualisation
plt.figure(figsize=(8,5))
plt.bar(unique_classes, class_counts, color=['green','blue','red','yellow'])
plt.xlabel('Classe')
plt.ylabel('Nombre d\'images')
plt.title('Distribution des classes dans le dataset OASIS')
plt.show()
```

```
 Distribution des classes
Classe: Mild Dementia | Nombre d'exemples: 3000
Classe: Moderate Dementia | Nombre d'exemples: 488
Classe: Non Demented | Nombre d'exemples: 3000
Classe: Very mild Dementia | Nombre d'exemples: 3000
```



[8]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score
```

[9]:
```
# Prétraitement pour le modèle
X = np.array(X)
y = np.array(y)
label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)

# Train-Test split (sans SMOTE)
X_train, X_test, y_train, y_test = train_test_split(X, y_encoded, test_size=0.
↪2, stratify=y_encoded)
```

[10]:
```
# Comparaison des modèles (sans SMOTE)
print("\n Comparaison des modèles")
models = [
    ('LR', LogisticRegression(max_iter=2000)),
    ('KNN', KNeighborsClassifier()),
    ('DT', DecisionTreeClassifier()),
    ('RF', RandomForestClassifier(n_estimators=100)),
```

```
    ('SVM', SVC(kernel='linear', C=1.0, probability=True))
]

results = []
names = []
kfold = StratifiedKFold(n_splits=10, shuffle=True)

for name, model in models:
    cv_results = cross_val_score(model, X_train, y_train, cv=kfold,␣
 ↪scoring='f1_macro')
    results.append(cv_results)
    names.append(name)
    print(f"{name}: {cv_results.mean():.3f} ({cv_results.std():.3f})")
```

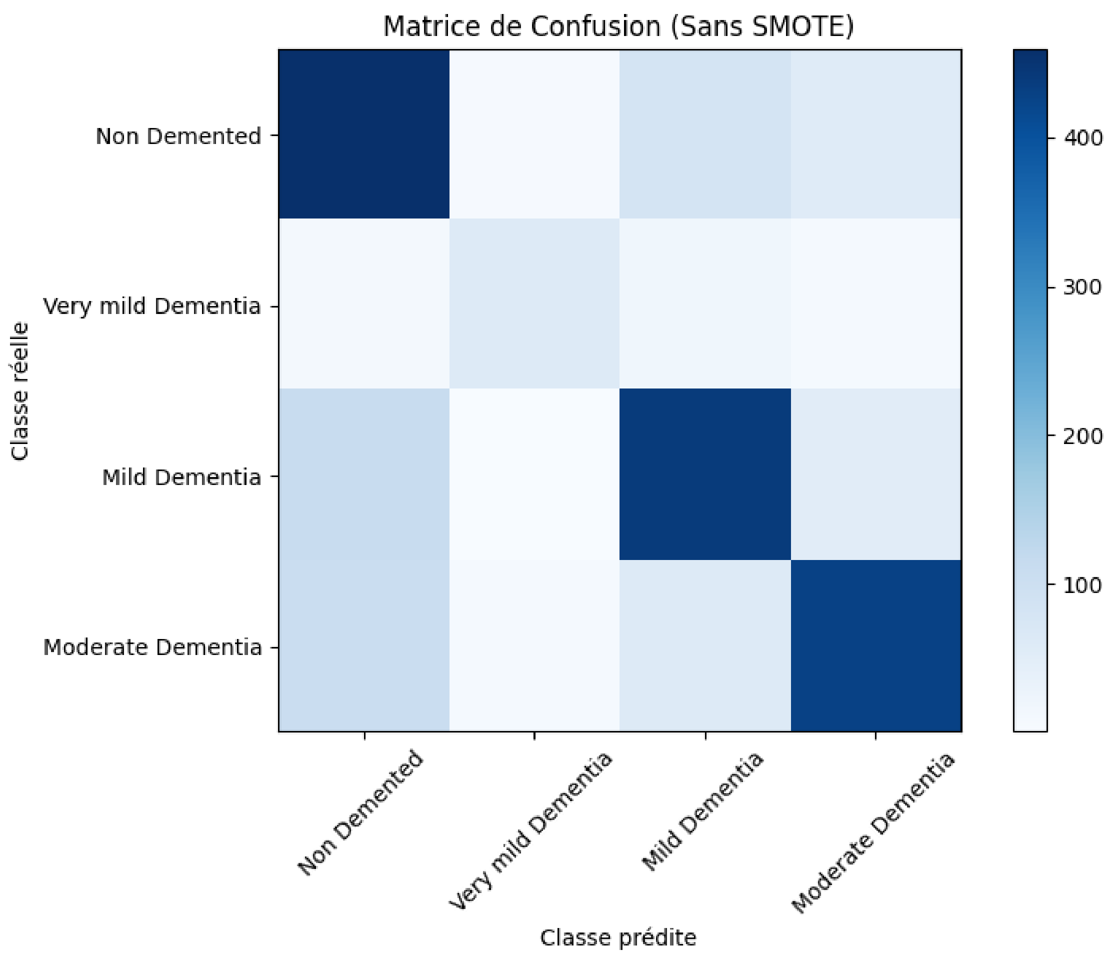
```
 Comparaison des modèles
LR: 0.660 (0.016)
KNN: 0.970 (0.010)
DT: 0.729 (0.022)
RF: 0.964 (0.010)
SVM: 0.702 (0.024)
```

[11]:
```
# Baseline SVM
svm_base = SVC(kernel='linear', C=1.0, probability=True)
svm_base.fit(X_train, y_train)
y_pred_base = svm_base.predict(X_test)
print("\n Rapport de Classification (Sans Rééquilibrage)")
print(classification_report(y_test, y_pred_base, target_names=label_encoder.
 ↪classes_))
```

```
 Rapport de Classification (Sans Rééquilibrage)
                    precision    recall  f1-score   support

     Mild Dementia       0.67      0.77      0.71       600
 Moderate Dementia       0.81      0.62      0.71        98
      Non Demented       0.73      0.73      0.73       600
Very mild Dementia       0.79      0.71      0.75       600

          accuracy                           0.73      1898
         macro avg       0.75      0.71      0.72      1898
      weighted avg       0.74      0.73      0.73      1898
```

[12]:
```
# Matrice de confusion
cm = confusion_matrix(y_test, y_pred_base)
plt.figure(figsize=(8,6))
```

```
plt.imshow(cm, interpolation='nearest', cmap='Blues')
plt.title("Matrice de Confusion (Sans SMOTE)")
plt.colorbar()
tick_marks = np.arange(len(classes))
plt.xticks(tick_marks, classes, rotation=45)
plt.yticks(tick_marks, classes)
plt.tight_layout()
plt.ylabel('Classe réelle')
plt.xlabel('Classe prédite')
plt.show()
```



[13]:
```
# Rééquilibrage avec SMOTE
smote = SMOTE()
X_train_res, y_train_res = smote.fit_resample(X_train, y_train)

# Rééquilibrage
_, counts = np.unique(y_train_res, return_counts=True)
```

```python
print("\n Classes après SMOTE ")
for cls, cnt in zip(unique_classes, counts):
    print(f"{cls}: {cnt}")
```

```
 Classes après SMOTE
Mild Dementia: 2400
Moderate Dementia: 2400
Non Demented: 2400
Very mild Dementia: 2400
```

[14]:
```python
# GridSearchCV
param_grid = {
    'C': [0.1, 1, 10],
    'kernel': ['linear', 'rbf'],
    'gamma': ['scale', 0.01, 0.1]}
```

[15]:
```python
# Cross-validation
cv = StratifiedKFold(n_splits=3, shuffle=True)
grid_search = GridSearchCV(SVC(), param_grid, cv=cv, n_jobs=-1,␣
 ↪scoring='f1_macro')
```

[16]:
```python
# Entraînement avec GridSearch sur données SMOTED
grid_search.fit(X_train_res, y_train_res)

print(f"Meilleurs paramètres: {grid_search.best_params_}")
```

```
Meilleurs paramètres: {'C': 10, 'gamma': 'scale', 'kernel': 'rbf'}
```

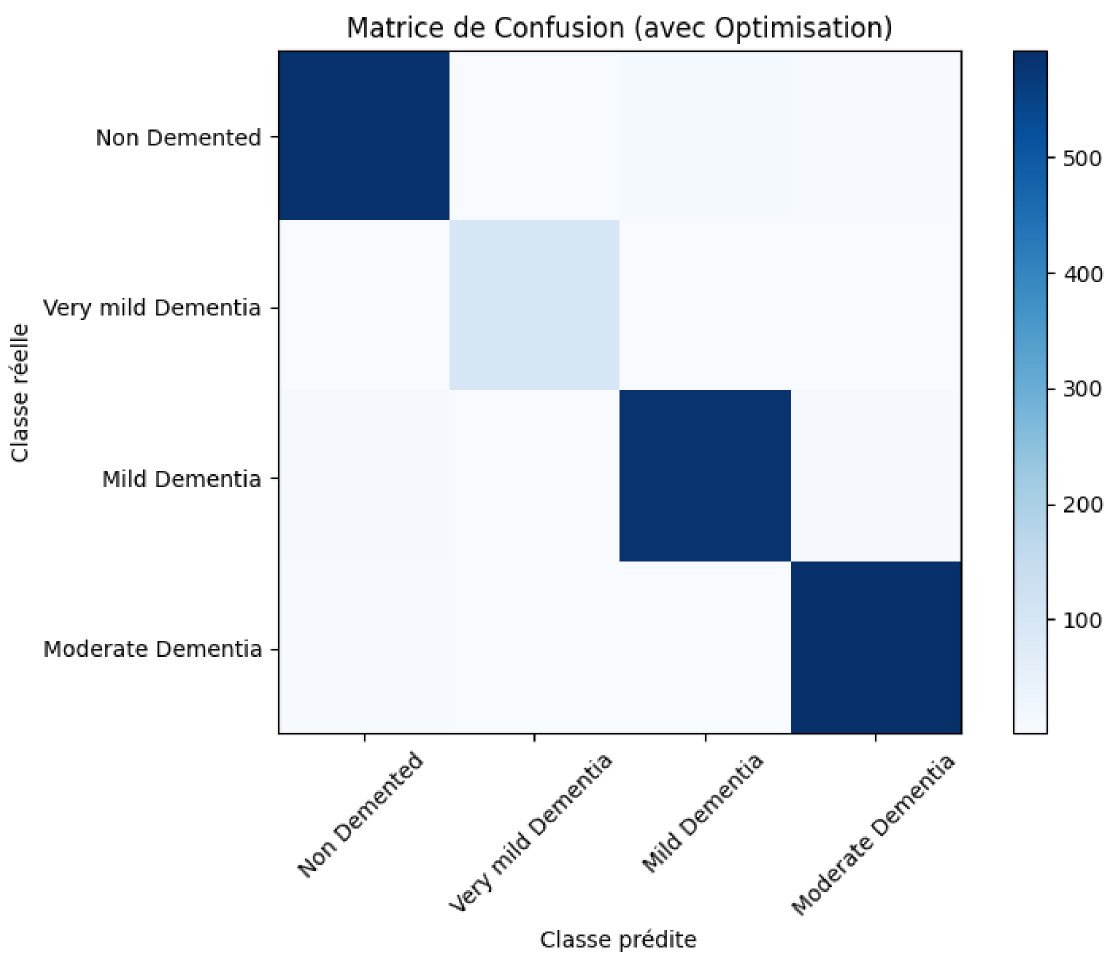
[17]:
```python
# Evaluation du modèle optimisé
best_svm = grid_search.best_estimator_
y_pred = best_svm.predict(X_test)
```

[18]:
```python
# Rapport de classification final
print("\n Rapport de Classification (avec SMOTE + Optimisation) ")
print(classification_report(y_test, y_pred, target_names=label_encoder.
 ↪classes_))
```

```
 Rapport de Classification (avec SMOTE + Optimisation)
                    precision    recall  f1-score   support

     Mild Dementia       0.98      0.98      0.98       600
 Moderate Dementia       0.97      0.96      0.96        98
      Non Demented       0.98      0.97      0.98       600
Very mild Dementia       0.98      0.99      0.98       600

          accuracy                           0.98      1898
```

```
   macro avg       0.98      0.97      0.98      1898
weighted avg       0.98      0.98      0.98      1898
```

[19]:
```
# Matrice de confusion finale
cm_optimized = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8,6))
plt.imshow(cm_optimized, interpolation='nearest', cmap='Blues')
plt.title("Matrice de Confusion (avec Optimisation)")
plt.colorbar()
plt.xticks(tick_marks, classes, rotation=45)
plt.yticks(tick_marks, classes)
plt.tight_layout()
plt.ylabel('Classe réelle')
plt.xlabel('Classe prédite')
plt.show()
```



[20]:
```
# Analyse de l'impact de SMOTE
print("\n Comparaison des performances ")
print("Précision avant SMOTE:", svm_base.score(X_test, y_test))
print("Précision après SMOTE:", best_svm.score(X_test, y_test))
```

```
 Comparaison des performances
Précision avant SMOTE: 0.7297154899894626
Précision après SMOTE: 0.9783983140147524
```

[21]:
```
# Validation croisée sur le modèle optimisé
cv_scores = cross_val_score(best_svm, X, y_encoded, cv=cv)
print(f"\nValidation croisée (Moyenne): {cv_scores.mean():.3f} ± {cv_scores.
 ↪std():.3f}")
```

```
Validation croisée (Moyenne): 0.976 ± 0.003
```