svm_clas

November 26, 2025

0.0.1 Importation des bibliothèques

[2]:
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split, RandomizedSearchCV
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, confusion_matrix,␣
↪classification_report, roc_auc_score
import matplotlib.pyplot as plt
import seaborn as sns
import joblib
from scipy.stats import loguniform
```

0.0.2 Chargement des données

[3]:
```
df = pd.read_csv("synthetic_heart_disease_dataset.csv")
```

0.0.3 Exploration des données

[4]:
```
df.shape
```

[4]:
```
(50000, 21)
```

[5]:
```
df.describe()
```

[5]:
```
                Age        Weight        Height           BMI  Hypertension  \
count   50000.00000  50000.000000  50000.000000  50000.000000  50000.000000
mean       54.46406     84.547520    174.460000     28.984284      0.299620
std        14.43809     20.213257     14.420379      6.367494      0.458096
min        30.00000     50.000000    150.000000     18.000000      0.000000
25%        42.00000     67.000000    162.000000     23.500000      0.000000
50%        54.00000     85.000000    174.000000     29.000000      0.000000
75%        67.00000    102.000000    187.000000     34.500000      1.000000
max        79.00000    119.000000    199.000000     40.000000      1.000000

           Diabetes  Hyperlipidemia  Family_History  Previous_Heart_Attack  \
count  50000.000000    50000.000000    50000.000000           50000.000000
```

```
mean      0.199260      0.251660      0.400500               0.099280
std       0.399448      0.433971      0.490005               0.299041
min       0.000000      0.000000      0.000000               0.000000
25%       0.000000      0.000000      0.000000               0.000000
50%       0.000000      0.000000      0.000000               0.000000
75%       0.000000      1.000000      1.000000               0.000000
max       1.000000      1.000000      1.000000               1.000000

        Systolic_BP  Diastolic_BP    Heart_Rate  Blood_Sugar_Fasting  \
count  50000.000000  50000.000000  50000.000000         50000.000000
mean     139.299580     89.528800     84.449560           124.493020
std       23.083544     17.258063     14.491325            31.691507
min      100.000000     60.000000     60.000000            70.000000
25%      119.000000     75.000000     72.000000            97.000000
50%      139.000000     90.000000     85.000000           125.000000
75%      159.000000    104.000000     97.000000           152.000000
max      179.000000    119.000000    109.000000           179.000000

       Cholesterol_Total  Heart_Disease
count       50000.000000   50000.000000
mean          224.556360       0.463460
std            43.157467       0.498668
min           150.000000       0.000000
25%           187.000000       0.000000
50%           225.000000       0.000000
75%           262.000000       1.000000
max           299.000000       1.000000
```

[6]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50000 entries, 0 to 49999
Data columns (total 21 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   Age                  50000 non-null  int64
 1   Gender               50000 non-null  object
 2   Weight               50000 non-null  int64
 3   Height               50000 non-null  int64
 4   BMI                  50000 non-null  float64
 5   Smoking              50000 non-null  object
 6   Alcohol_Intake       29891 non-null  object
 7   Physical_Activity    50000 non-null  object
 8   Diet                 50000 non-null  object
 9   Stress_Level         50000 non-null  object
 10  Hypertension         50000 non-null  int64
 11  Diabetes             50000 non-null  int64
 12  Hyperlipidemia       50000 non-null  int64
```

```
 13  Family_History         50000 non-null  int64
 14  Previous_Heart_Attack  50000 non-null  int64
 15  Systolic_BP            50000 non-null  int64
 16  Diastolic_BP           50000 non-null  int64
 17  Heart_Rate             50000 non-null  int64
 18  Blood_Sugar_Fasting    50000 non-null  int64
 19  Cholesterol_Total      50000 non-null  int64
 20  Heart_Disease          50000 non-null  int64
dtypes: float64(1), int64(14), object(6)
memory usage: 8.0+ MB
```

[7]:
```
df.head()
```

[7]:
```
   Age  Gender  Weight  Height   BMI  Smoking Alcohol_Intake  \
0   48    Male      78     157  26.4    Never            NaN
1   35  Female      73     163  33.0    Never            Low
2   79  Female      88     152  32.3    Never            NaN
3   75    Male     106     171  37.4    Never       Moderate
4   34  Female      65     191  18.5  Current            NaN

  Physical_Activity     Diet Stress_Level  …  Diabetes  Hyperlipidemia  \
0         Sedentary  Healthy       Medium  …         0               1
1            Active  Average         High  …         0               1
2          Moderate  Average       Medium  …         0               0
3          Moderate  Average          Low  …         0               1
4         Sedentary  Healthy          Low  …         1               0

   Family_History  Previous_Heart_Attack  Systolic_BP  Diastolic_BP  \
0               1                      0          104            99
1               1                      0          111            72
2               1                      0          116           102
3               0                      0          171            92
4               0                      0          164            67

   Heart_Rate  Blood_Sugar_Fasting  Cholesterol_Total  Heart_Disease
0          71                  165                200              0
1          60                  145                206              0
2          78                  148                208              0
3         109                  105                290              1
4         108                  116                220              1

[5 rows x 21 columns]
```

[8]:
```
df.columns
```

[8]:
```
Index(['Age', 'Gender', 'Weight', 'Height', 'BMI', 'Smoking', 'Alcohol_Intake',
       'Physical_Activity', 'Diet', 'Stress_Level', 'Hypertension', 'Diabetes',
       'Hyperlipidemia', 'Family_History', 'Previous_Heart_Attack',
```

```
       'Systolic_BP', 'Diastolic_BP', 'Heart_Rate', 'Blood_Sugar_Fasting',
       'Cholesterol_Total', 'Heart_Disease'],
      dtype='object')
```

[9]:
```
df['Heart_Disease'].value_counts()
```

[9]:
```
Heart_Disease
0    26827
1    23173
Name: count, dtype: int64
```

[10]:
```
df.isna().sum()
```

[10]:
```
Age                        0
Gender                     0
Weight                     0
Height                     0
BMI                        0
Smoking                    0
Alcohol_Intake         20109
Physical_Activity          0
Diet                       0
Stress_Level               0
Hypertension               0
Diabetes                   0
Hyperlipidemia             0
Family_History             0
Previous_Heart_Attack      0
Systolic_BP                0
Diastolic_BP               0
Heart_Rate                 0
Blood_Sugar_Fasting        0
Cholesterol_Total          0
Heart_Disease              0
dtype: int64
```

[11]:
```
df[df['Alcohol_Intake'].isna()]
```

[11]:

	Age	Gender	Weight	Height	BMI	Smoking	Alcohol_Intake	\
0	48	Male	78	157	26.4	Never	NaN	
2	79	Female	88	152	32.3	Never	NaN	
4	34	Female	65	191	18.5	Current	NaN	
5	50	Male	116	186	25.3	Current	NaN	
7	51	Male	75	176	18.2	Former	NaN	
…	…	…	…	…	…	…	…	
49985	54	Male	113	190	19.4	Current	NaN	
49986	46	Female	54	167	36.2	Never	NaN	

```
49989   37    Male     117      178  30.4    Never            NaN
49994   62    Male      91      197  36.8    Never            NaN
49995   74    Male     104      155  29.9  Current            NaN

      Physical_Activity       Diet Stress_Level  …  Diabetes  \
0             Sedentary    Healthy       Medium  …         0
2              Moderate    Average       Medium  …         0
4             Sedentary    Healthy          Low  …         1
5             Sedentary    Average       Medium  …         0
7                Active    Average       Medium  …         0
…                     …          …          …  …         …
49985          Moderate    Average          Low  …         1
49986          Moderate    Average       Medium  …         0
49989          Moderate    Healthy          Low  …         0
49994            Active  Unhealthy          Low  …         0
49995            Active    Average       Medium  …         0

       Hyperlipidemia  Family_History  Previous_Heart_Attack  Systolic_BP  \
0                   1               1                      0          104
2                   0               1                      0          116
4                   0               0                      0          164
5                   1               0                      0          171
7                   0               1                      0          117
…                 …             …                    …          …
49985               0               1                      0          113
49986               1               0                      0          108
49989               1               0                      0          138
49994               0               1                      0          117
49995               0               0                      0          127

       Diastolic_BP  Heart_Rate  Blood_Sugar_Fasting  Cholesterol_Total  \
0                99          71                  165                200
2               102          78                  148                208
4                67         108                  116                220
5                91         106                   97                225
7                63          89                  143                154
…               …          …                  …                …
49985            90         108                  122                225
49986           108          70                   83                291
49989            89         102                   99                230
49994            80         106                   97                270
49995            80          83                  174                248

       Heart_Disease
0                  0
2                  0
4                  1
```

```
5                    0
7                    0
...              ...
49985                0
49986                1
49989                0
49994                1
49995                1

[20109 rows x 21 columns]
```

0.0.4 Nettoyage des données

[12]:
```python
df = df.drop(columns="Alcohol_Intake")
```

[13]:
```python
# Vérifier à nouveau le nombre de lignes après suppression
len(df)
```

[13]:
```
50000
```

[14]:
```python
df.isna().sum()
```

[14]:
```
Age                        0
Gender                     0
Weight                     0
Height                     0
BMI                        0
Smoking                    0
Physical_Activity          0
Diet                       0
Stress_Level               0
Hypertension               0
Diabetes                   0
Hyperlipidemia             0
Family_History             0
Previous_Heart_Attack      0
Systolic_BP                0
Diastolic_BP               0
Heart_Rate                 0
Blood_Sugar_Fasting        0
Cholesterol_Total          0
Heart_Disease              0
dtype: int64
```

0.0.5 Encodage des variables catégorielles

[15]:
```python
cate_cols = df.select_dtypes(include=['object']).columns
cate_cols
```

[15]:
```
Index(['Gender', 'Smoking', 'Physical_Activity', 'Diet', 'Stress_Level'],
dtype='object')
```

[16]:
```python
for col in cate_cols:
    print(f"{col} : {df[col].unique()}")
```

```
Gender : ['Male' 'Female']
Smoking : ['Never' 'Current' 'Former']
Physical_Activity : ['Sedentary' 'Active' 'Moderate']
Diet : ['Healthy' 'Average' 'Unhealthy']
Stress_Level : ['Medium' 'High' 'Low']
```

[17]:
```python
for col in cate_cols:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col])
    mapping = dict(zip(le.classes_, range(len(le.classes_))))
    print(f"{col}       : {mapping}")

df.info()
```

```
Gender       : {'Female': 0, 'Male': 1}
Smoking       : {'Current': 0, 'Former': 1, 'Never': 2}
Physical_Activity       : {'Active': 0, 'Moderate': 1, 'Sedentary': 2}
Diet       : {'Average': 0, 'Healthy': 1, 'Unhealthy': 2}
Stress_Level       : {'High': 0, 'Low': 1, 'Medium': 2}
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50000 entries, 0 to 49999
Data columns (total 20 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   Age                    50000 non-null  int64
 1   Gender                 50000 non-null  int64
 2   Weight                 50000 non-null  int64
 3   Height                 50000 non-null  int64
 4   BMI                    50000 non-null  float64
 5   Smoking                50000 non-null  int64
 6   Physical_Activity      50000 non-null  int64
 7   Diet                   50000 non-null  int64
 8   Stress_Level           50000 non-null  int64
 9   Hypertension           50000 non-null  int64
 10  Diabetes               50000 non-null  int64
 11  Hyperlipidemia         50000 non-null  int64
 12  Family_History         50000 non-null  int64
 13  Previous_Heart_Attack  50000 non-null  int64
```

```
 14  Systolic_BP            50000 non-null  int64
 15  Diastolic_BP           50000 non-null  int64
 16  Heart_Rate             50000 non-null  int64
 17  Blood_Sugar_Fasting    50000 non-null  int64
 18  Cholesterol_Total      50000 non-null  int64
 19  Heart_Disease          50000 non-null  int64
dtypes: float64(1), int64(19)
memory usage: 7.6 MB
```

0.0.6 Préparation des données pour l'entraînement

[18]:
```python
X = df.drop(columns='Heart_Disease')
y = df['Heart_Disease']
```

[19]:
```python
y.value_counts()
```

[19]:
```
Heart_Disease
0    26827
1    23173
Name: count, dtype: int64
```

0.0.7 Equilibrage des donnees

[20]:
```python
def equilibrage(X, y):
    import random
    X = X.values
    y = y.values
    idx_ones = [i for i, label in enumerate(y) if label == 1]
    idx_zeros = [i for i, label in enumerate(y) if label == 0]

    if len(idx_ones) > len(idx_zeros):
        majority = idx_ones
        minority = idx_zeros
    else:
        majority = idx_zeros
        minority = idx_ones

    random.shuffle(majority)
    majority = majority[:len(minority)]

    indices = majority + minority
    random.shuffle(indices)

    X_final = [X[i] for i in indices]
    y_final = [y[i] for i in indices]

    X_final = np.array(X_final)
```

```
    y_final = np.array(y_final).reshape(-1, 1)

    return X_final, y_final
```

[21]:
```
X, y = equilibrage(X, y)
```

[22]:
```
(y == 1).sum() == (y == 0).sum()
```

[22]:
```
np.True_
```

[23]:
```
X.shape
```

[23]:
```
(46346, 19)
```

[24]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)
```

0.0.8 Normalisation des données

[25]:
```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

0.0.9 Entraînement du modèle

[26]:
```
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
↪f1_score
import numpy as np

# Résoudre l'avertissement sur la forme de y
y_train_flat = np.ravel(y_train)

# Initialiser et entraîner le modèle de SVM
model = SVC(kernel='linear', random_state=42)
model.fit(X_train, y_train_flat)

# Faire des prédictions sur l'ensemble de test
y_pred = model.predict(X_test)

print("=== RÉSULTATS SVM ===")
print("    Accuracy :", accuracy_score(y_test, y_pred))
print("    Precision :", precision_score(y_test, y_pred))
print("    Recall :", recall_score(y_test, y_pred))
print("    F1-score :", f1_score(y_test, y_pred))

# Afficher les coefficients comme importance des features
if hasattr(model, 'coef_'):
    print("\n=== IMPORTANCE DES FEATURES (Coefficients SVM) ===")
```

```
    # Générer des noms de features si les colonnes originales ne sont pas␣
 ↪disponibles
    try:
        feature_names = X.columns.tolist()
    except:
        feature_names = [f'Feature_{i}' for i in range(X_train.shape[1])]

    importance_df = pd.DataFrame({
        'Feature': feature_names,
        'Coefficient': model.coef_[0],
        'Importance_Abs': abs(model.coef_[0])
    }).sort_values('Importance_Abs', ascending=False)

    print(importance_df.head(10))
else:
    print("\nFeature importances non disponibles pour ce type de kernel SVM")
```

```
=== RÉSULTATS SVM ===
    Accuracy : 0.9264293419633225
    Precision : 0.9230769230769231
    Recall : 0.9313893653516295
    F1-score : 0.9272145144076841

=== IMPORTANCE DES FEATURES (Coefficients SVM) ===
       Feature  Coefficient  Importance_Abs
9    Feature_9     2.374331        2.374331
0    Feature_0     2.282069        2.282069
18  Feature_18     2.261945        2.261945
10  Feature_10     2.089283        2.089283
13  Feature_13     1.578267        1.578267
1    Feature_1    -0.024239        0.024239
15  Feature_15    -0.023390        0.023390
12  Feature_12     0.021601        0.021601
11  Feature_11    -0.018443        0.018443
8    Feature_8    -0.018371        0.018371
```

0.0.10 Sélection des features les plus importantes

[41]:
```
# Sélectionner les 10 features les plus importantes
top_features_idx = importance_df.head(10).index
top_features_names = importance_df.head(10)['Feature'].tolist()

print("=== TOP 10 FEATURES SÉLECTIONNÉES ===")
for i, feature in enumerate(top_features_names, 1):
    importance = importance_df.loc[importance_df['Feature'] == feature,␣
 ↪'Importance_Abs'].values[0]
```

```
    print(f"{i}. {feature} (Importance: {importance:.4f})")
```

```
=== TOP 10 FEATURES SÉLECTIONNÉES ===
1. Feature_9 (Importance: 2.3743)
2. Feature_0 (Importance: 2.2821)
3. Feature_18 (Importance: 2.2619)
4. Feature_10 (Importance: 2.0893)
5. Feature_13 (Importance: 1.5783)
6. Feature_1 (Importance: 0.0242)
7. Feature_15 (Importance: 0.0234)
8. Feature_12 (Importance: 0.0216)
9. Feature_11 (Importance: 0.0184)
10. Feature_8 (Importance: 0.0184)
```

0.0.11 Préparation des données avec seulement les features importantes

[42]:
```
# Créer un mapping des features pour comprendre ce que représentent Feature_0,␣
↪Feature_1, etc.
feature_mapping = {
    'Feature_0': 'Age',
    'Feature_1': 'Gender',
    'Feature_2': 'Weight',
    'Feature_3': 'Height',
    'Feature_4': 'BMI',
    'Feature_5': 'Smoking',
    'Feature_6': 'Physical_Activity',
    'Feature_7': 'Diet',
    'Feature_8': 'Stress_Level',
    'Feature_9': 'Hypertension',
    'Feature_10': 'Diabetes',
    'Feature_11': 'Hyperlipidemia',
    'Feature_12': 'Family_History',
    'Feature_13': 'Previous_Heart_Attack',
    'Feature_14': 'Systolic_BP',
    'Feature_15': 'Diastolic_BP',
    'Feature_16': 'Heart_Rate',
    'Feature_17': 'Blood_Sugar_Fasting',
    'Feature_18': 'Cholesterol_Total'
}

print("\n=== INTERPRÉTATION DES FEATURES IMPORTANTES ===")
for feature in top_features_names:
    real_name = feature_mapping.get(feature, feature)
    importance = importance_df.loc[importance_df['Feature'] == feature,␣
↪'Importance_Abs'].values[0]
    coefficient = importance_df.loc[importance_df['Feature'] == feature,␣
↪'Coefficient'].values[0]
```

```
    print(f"{real_name:.<25} Importance: {importance:.4f} (Coeff: {coefficient:
↪+.4f})")
```

```
=== INTERPRÉTATION DES FEATURES IMPORTANTES ===
Hypertension… Importance: 2.3743 (Coeff: +2.3743)
Age… Importance: 2.2821 (Coeff: +2.2821)
Cholesterol_Total… Importance: 2.2619 (Coeff: +2.2619)
Diabetes… Importance: 2.0893 (Coeff: +2.0893)
Previous_Heart_Attack… Importance: 1.5783 (Coeff: +1.5783)
Gender… Importance: 0.0242 (Coeff: -0.0242)
Diastolic_BP… Importance: 0.0234 (Coeff: -0.0234)
Family_History… Importance: 0.0216 (Coeff: +0.0216)
Hyperlipidemia… Importance: 0.0184 (Coeff: -0.0184)
Stress_Level… Importance: 0.0184 (Coeff: -0.0184)
```

0.0.12 Préparation des données réduites

[43]:
```
# Recharger les données originales pour sélectionner les bonnes colonnes
df_original = pd.read_csv("synthetic_heart_disease_dataset.csv")
df_original = df_original.drop(columns=["Alcohol_Intake"])

# Encoder à nouveau les variables catégorielles
categorical_columns = ['Gender', 'Smoking', 'Physical_Activity', 'Diet',␣
↪'Stress_Level']
for col in categorical_columns:
    le = LabelEncoder()
    df_original[col] = le.fit_transform(df_original[col])

# Sélectionner uniquement les features importantes
important_features_real_names = [feature_mapping[feature] for feature in␣
↪top_features_names]
X_reduced = df_original[important_features_real_names]
y_original = df_original['Heart_Disease']

print(f"Shape avant réduction: {df_original.shape}")
print(f"Shape après réduction: {X_reduced.shape}")
print(f"Features utilisées: {important_features_real_names}")
```

```
Shape avant réduction: (50000, 20)
Shape après réduction: (50000, 10)
Features utilisées: ['Hypertension', 'Age', 'Cholesterol_Total', 'Diabetes',
'Previous_Heart_Attack', 'Gender', 'Diastolic_BP', 'Family_History',
'Hyperlipidemia', 'Stress_Level']
```

0.0.13 Équilibrage des données réduites

[52]:
```
def equilibrage_corrige(X, y):

    # Convertir en arrays numpy
    X_array = X.values if hasattr(X, 'values') else np.array(X)
    y_array = y.values if hasattr(y, 'values') else np.array(y)

    # Aplatir y
    y_array = y_array.ravel()

    # Séparer les classes
    idx_ones = np.where(y_array == 1)[0]
    idx_zeros = np.where(y_array == 0)[0]

    print(f"Distribution originale - Classe 0: {len(idx_zeros)}, Classe 1:␣
 ↪{len(idx_ones)}")

    # Sous-échantillonnage
    if len(idx_ones) > len(idx_zeros):
        majority = idx_ones
        minority = idx_zeros
    else:
        majority = idx_zeros
        minority = idx_ones

    np.random.shuffle(majority)
    majority_sampled = majority[:len(minority)]

    # Combiner et mélanger
    balanced_indices = np.concatenate([majority_sampled, minority])
    np.random.shuffle(balanced_indices)

    X_balanced = X_array[balanced_indices]
    y_balanced = y_array[balanced_indices]

    print(f"Distribution équilibrée - Classe 0: {(y_balanced == 0).sum()},␣
 ↪Classe 1: {(y_balanced == 1).sum()}")

    return X_balanced, y_balanced

# Appliquer l'équilibrage
X_balanced_reduced, y_balanced_reduced = equilibrage_corrige(X_reduced,␣
 ↪y_original)
print(f"\nShape finale des données équilibrées: {X_balanced_reduced.shape}")
```

```
Distribution originale - Classe 0: 26827, Classe 1: 23173
Distribution équilibrée - Classe 0: 23173, Classe 1: 23173
```

```
Shape finale des données équilibrées: (46346, 10)
```

0.0.14 Division et normalisation des données réduites

[45]:
```python
X_train_reduced, X_test_reduced, y_train_reduced, y_test_reduced = train_test_split(
    X_balanced_reduced, y_balanced_reduced, test_size=0.2, random_state=42, stratify=y_balanced_reduced
)

# Normalisation
scaler_reduced = StandardScaler()
X_train_reduced_scaled = scaler_reduced.fit_transform(X_train_reduced)
X_test_reduced_scaled = scaler_reduced.transform(X_test_reduced)

print("=== DONNÉES RÉDUITES PRÊTES ===")
print(f"Train shape: {X_train_reduced_scaled.shape}")
print(f"Test shape: {X_test_reduced_scaled.shape}")
print(f"Réduction du nombre de features: {X.shape[1]} → {X_train_reduced_scaled.shape[1]}")
print(f"Pourcentage de réduction: {(1 - X_train_reduced_scaled.shape[1]/X.shape[1])*100:.1f}%")
```

```
=== DONNÉES RÉDUITES PRÊTES ===
Train shape: (37076, 10)
Test shape: (9270, 10)
Réduction du nombre de features: 19 → 10
Pourcentage de réduction: 47.4%
```

0.0.15 Entraînement du modèle SVM avec features réduites

[46]:
```python
import time

print("=== ENTRAÎNEMENT SVM AVEC FEATURES RÉDUITES ===")

# Mesurer le temps d'entraînement
start_time = time.time()

model_reduced = SVC(kernel='linear', random_state=42)
model_reduced.fit(X_train_reduced_scaled, y_train_reduced)

training_time_reduced = time.time() - start_time

# Prédictions
y_pred_reduced = model_reduced.predict(X_test_reduced_scaled)

print(f"Temps d'entraînement: {training_time_reduced:.4f} secondes")
```

```
print(f"Accuracy : {accuracy_score(y_test_reduced, y_pred_reduced):.6f}")
print(f"Precision : {precision_score(y_test_reduced, y_pred_reduced):.6f}")
print(f"Recall : {recall_score(y_test_reduced, y_pred_reduced):.6f}")
print(f"F1-score : {f1_score(y_test_reduced, y_pred_reduced):.6f}")
```

```
=== ENTRAÎNEMENT SVM AVEC FEATURES RÉDUITES ===
Temps d'entraînement: 36.9736 secondes
Accuracy : 0.926106
Precision : 0.921828
Recall : 0.931176
F1-score : 0.926478
```

0.0.16 Comparaison des performances

[47]:
```
print("=== COMPARAISON DÉTAILLÉE ===")
print("AVEC TOUTES LES FEATURES (19):")
print(f"  Accuracy: {accuracy_score(y_test, y_pred):.6f}")
print(f"  F1-score: {f1_score(y_test, y_pred):.6f}")

print(f"\nAVEC FEATURES SÉLECTIONNÉES ({X_train_reduced_scaled.shape[1]}):")
print(f"  Accuracy: {accuracy_score(y_test_reduced, y_pred_reduced):.6f}")
print(f"  F1-score: {f1_score(y_test_reduced, y_pred_reduced):.6f}")

# Calcul des différences
acc_diff = accuracy_score(y_test_reduced, y_pred_reduced) -␣
 ↪accuracy_score(y_test, y_pred)
f1_diff = f1_score(y_test_reduced, y_pred_reduced) - f1_score(y_test, y_pred)

print(f"\nDIFFÉRENCE (Réduit - Complet):")
print(f"  Accuracy: {acc_diff:+.6f}")
print(f"  F1-score: {f1_diff:+.6f}")

# Évaluation
if acc_diff >= 0 and f1_diff >= 0:
    print(" EXCELLENT! La sélection de features a amélioré les performances!")
elif acc_diff >= -0.01 and f1_diff >= -0.01:
    print(" TRÈS BIEN! Performances maintenues avec moins de features!")
else:
    print("  Légère baisse des performances, mais gain en interprétabilité")
```

```
=== COMPARAISON DÉTAILLÉE ===
AVEC TOUTES LES FEATURES (19):
  Accuracy: 0.926429
  F1-score: 0.927215

AVEC FEATURES SÉLECTIONNÉES (10):
  Accuracy: 0.926106
  F1-score: 0.926478
```

```
DIFFÉRENCE (Réduit - Complet):
  Accuracy: -0.000324
  F1-score: -0.000736
 TRÈS BIEN! Performances maintenues avec moins de features!
```

0.0.17 Sauvegarde des nouveaux modèles

[51]:
```python
# Sauvegarder le modèle avec features réduites
joblib.dump(model_reduced, "svm_clas_reduced_features.pkl")
joblib.dump(scaler_reduced, "scaler_svm_reduced_features.pkl")
joblib.dump(important_features_real_names, "important_features_names.pkl")
```

[51]:
```
['important_features_names.pkl']
```