TP: Support Vector Machine (SVM)

Zakaria ELGUAZZAR

March 29, 2025

1 Introduction

Le but de ce TP est d'implémenter un SVM non linéaire pour la classification de données, d'optimiser ses hyperparamètres et d'analyser les vecteurs supports ainsi que les frontières de décision.

2 Chargement des Bibliothèques

Nous commençons par importer les bibliothèques nécessaires :

```
import kagglehub
import pandas as pd
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
import matplotlib.pyplot as plt
import seaborn as sns
from mlxtend.plotting import plot_decision_regions
from sklearn.metrics import accuracy_score, precision_score,
    recall_score, f1_score, classification_report
```

3 Prétraitement des Données

Nous commençons par charger le jeu de données et explorer ses caractéristiques.

3.1 Chargement des données

```
path = kagglehub.dataset_download("hosammhmdali/diabetes-dataset")
data = pd.read_csv(path+"/diabetes.csv")
```

3.2 Analyse exploratoire

Nous visualisons les corrélations entre les variables pour sélectionner les plus discriminantes.

```
plt.figure(figsize=(10, 6))
sns.heatmap(data.corr(), annot=True, cmap="coolwarm")
plt.show()
```

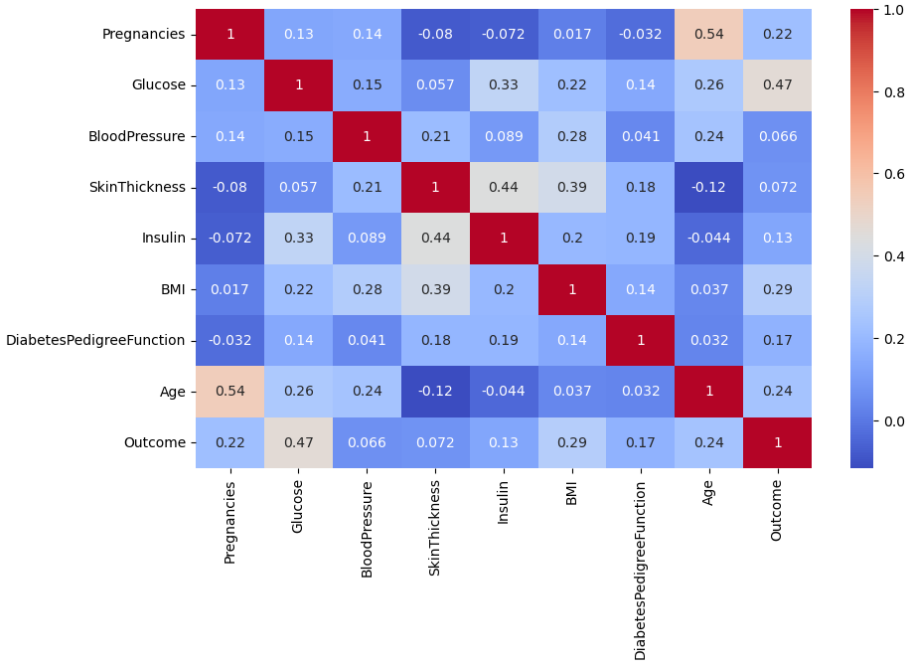

Figure 1: Légende de l'image

3.3 Réduction de Dimension avec PCA

Nous appliquons une standardisation avant de projeter les données en deux dimensions avec PCA.

```
X = data.drop("Outcome", axis=1).values
y = data["Outcome"].values

# Standardisation des donnees
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Application du PCA
pca = PCA(n_components=2)
```

```
X_pca = pca.fit_transform(X_scaled)

# Affichage des donnees projetees
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=y, cmap="coolwarm",
    edgecolors='k')
plt.xlabel("PC1")
plt.ylabel("PC2")
plt.title("Projection␣PCA␣pour␣la␣visualisation")
plt.show()
```

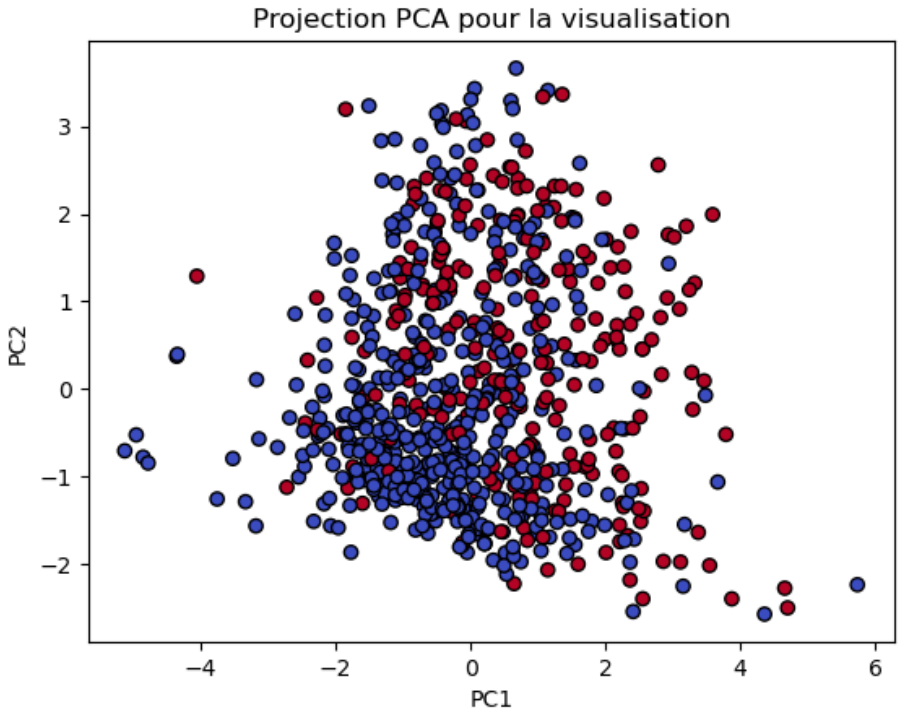

Figure 2: Projectionn PCA

4 Séparation du Jeu de Données

Nous divisons les données en ensemble d'entraînement (70%) et ensemble de test (30%).

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
    =0.3, random_state=42)
```

5 Modélisation avec SVM

Nous entraînons un SVM non linéaire (noyau RBF) pour la classification.

```
# Creer et entrainer le modele SVM avec un noyau RBF
svm_rbf = SVC(kernel='rbf')
svm_rbf.fit(X_train, y_train)

# Predictions sur l'ensemble de test
y_pred_rbf = svm_rbf.predict(X_test)

# Evaluer la performance du modele
accuracy = accuracy_score(y_test, y_pred_rbf)
print(f"Accuracy: {accuracy:.4f}")

# Afficher un rapport detaille
print(classification_report(y_test, y_pred_rbf))
```

Nous entraînons un SVM linéaire pour la classification.

```
# Creer et entrainer le modele SVM avec un noyau lineaire
svm_linear = SVC(kernel='linear')
svm_linear.fit(X_train, y_train)

# Predictions sur l'ensemble de test
y_pred_linear = svm_linear.predict(X_test)

# Evaluer la performance du modele
accuracy = accuracy_score(y_test, y_pred_linear)
print(f"Accuracy: {accuracy:.4f}")

# Afficher un rapport detaille
print(classification_report(y_test, y_pred_linear))
```

6 Optimisation des Hyperparamètres

Nous utilisons `GridSearchCV` pour trouver les meilleurs paramètres C et gamma.

```
# Define the hyperparameter grid
param_grid = {
    'C': [0.1, 1, 10, 100],  # Regularization parameter
    'kernel': ['linear','rbf'],  # Kernel types
    'gamma': ['scale', 'auto', 0.01, 0.1, 1]  # Gamma values (for '
        rbf' kernel)
}

# Create the SVC model
svc = SVC()

# Create the GridSearchCV instance
grid_search = GridSearchCV(svc, param_grid, cv=5, scoring='accuracy
    ', n_jobs=4, verbose=1)

# Fit GridSearchCV on training data
grid_search.fit(X_train, y_train)
```

```
# Display the best parameters and best accuracy
print("Best Parameters:", grid_search.best_params_)
print("Best Accuracy:", grid_search.best_score_)

# Convert results to DataFrame for easier analysis
results = pd.DataFrame(grid_search.cv_results_)

# Sort results by best accuracy
sorted_results = results[['param_C', 'param_kernel', 'param_gamma',
     'mean_test_score']].sort_values(by='mean_test_score',
    ascending=False)
# Print sorted results
print(sorted_results)
```

```
Fitting 5 folds for each of 40 candidates, totalling 200 fits
Best Parameters: {'C': 1, 'gamma': 0.01, 'kernel': 'rbf'}
Best Accuracy: 0.7713049498096228
    param_C param_kernel param_gamma  mean_test_score
15      1.0          rbf        0.01         0.771305
25     10.0          rbf        0.01         0.762080
0       0.1       linear       scale         0.756490
2       0.1       linear        auto         0.756490
8       0.1       linear           1         0.756490
20     10.0       linear       scale         0.756490
6       0.1       linear         0.1         0.756490
4       0.1       linear        0.01         0.756490
32    100.0       linear        auto         0.756490
34    100.0       linear        0.01         0.756490
38    100.0       linear           1         0.756490
30    100.0       linear       scale         0.756490
28     10.0       linear           1         0.756490
26     10.0       linear         0.1         0.756490
24     10.0       linear        0.01         0.756490
22     10.0       linear        auto         0.756490
36    100.0       linear         0.1         0.756490
18      1.0       linear           1         0.754638
12      1.0       linear        auto         0.754638
16      1.0       linear         0.1         0.754638
10      1.0       linear       scale         0.754638
14      1.0       linear        0.01         0.754638
35    100.0          rbf        0.01         0.749031
27     10.0          rbf         0.1         0.741623
17      1.0          rbf         0.1         0.739754
11      1.0          rbf       scale         0.736068
13      1.0          rbf        auto         0.732347
7       0.1          rbf         0.1         0.728539
21     10.0          rbf       scale         0.721201
23     10.0          rbf        auto         0.717497
1       0.1          rbf       scale         0.713690
3       0.1          rbf        auto         0.711821
37    100.0          rbf         0.1         0.704534
31    100.0          rbf       scale         0.685929
33    100.0          rbf        auto         0.684060
19      1.0          rbf           1         0.672828
39    100.0          rbf           1         0.661734
29     10.0          rbf           1         0.661734
```

7 Analyse des Vecteurs Supports

Nous affichons le nombre et la répartition des vecteurs supports.

```
# Affichage du nombre total de vecteurs supports
total_support_vectors = svm_rbf.support_vectors_.shape[0]
print(f"Nombre total de vecteurs supports : {total_support_vectors}
    ")

# Affichage de la repartition des vecteurs supports par classe
support_vectors_per_class = svm_rbf.n_support_
for i, count in enumerate(support_vectors_per_class):
    print(f"Classe {i}: {count} vecteurs supports")
```

8 Visualisation des Frontieres de Decision

Nous utilisons une projection PCA pour visualiser la frontière de décision.

```
svm_rbf.fit(X_pca, y)
plt.figure(figsize=(8, 6))
plot_decision_regions(X_pca, y, clf=svm_rbf, legend=2)
plt.xlabel("Composante principale 1")
plt.ylabel("Composante principale 2")
plt.title("Frontiere de decision SVM sur PCA")
plt.show()
```

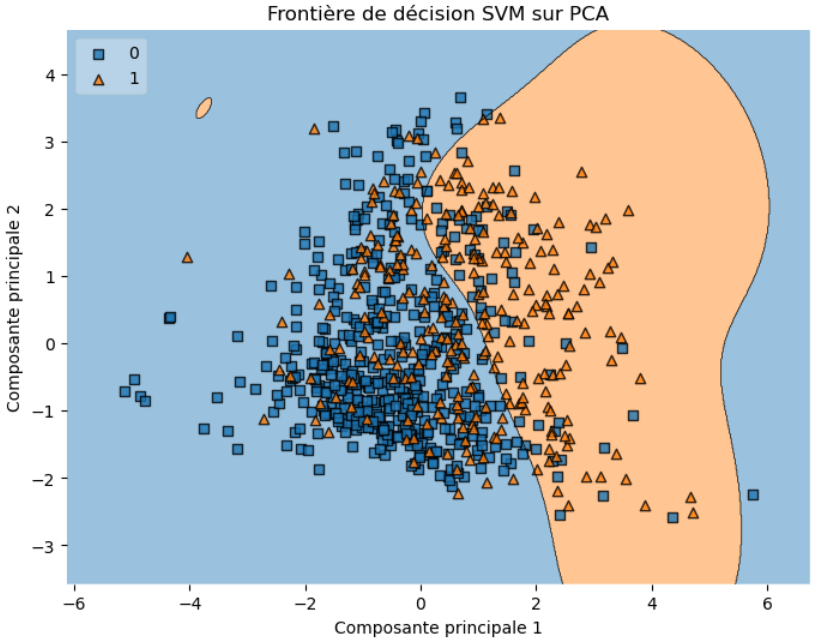

Figure 3: Projection PCA

9 Évaluation du Modèle

Nous évaluons la performance du modèle en utilisant les métriques standard.

```
y_pred = svm_model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

print(f"Accuracy: {accuracy:.2f}")
print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
print(f"F1-score: {f1:.2f}")
```

```
Evaluation du modele SVM Lineaire:
Accuracy  = 0.7879
Pr cision = 0.7333
Rappel    = 0.5714
F1-score  = 0.6423

Evaluation du modele SVM RBF:
Accuracy  = 0.7835
Pr cision = 0.7455
Rappel    = 0.5325
F1-score  = 0.6212

  valuation   du mod le SVM_optimal:
Accuracy  = 0.7965
Pr cision = 0.7586
Rappel    = 0.5714
F1-score  = 0.6519
```

Questions a repondre

Pourquoi certains points deviennent des vecteurs supports ?

Un point est considéré comme un vecteur support lorsqu'il est situé à proximité de la séparation entre deux classes. Ces points influencent directement la position de la frontière de décision : toute modification de leur position peut entraîner un ajustement de cette frontière.

Impact sur la frontière de décision

L'ajout ou la suppression d'un vecteur support peut modifier la position de la frontière de séparation.

- **Moins il y a de vecteurs supports**, plus la frontière est influencée par un petit nombre de points, ce qui peut entraîner un **surapprentissage**.

- Un **SVM bien régularise** cherche à utiliser un nombre optimal de vecteurs supports pour garantir une bonne capacité de **généralisation** sur de nouvelles données.

Comparaison

Dans notre cas, le **SVM avec noyau RBF** est le plus adapté. Bien que notre problème soit une classification binaire, les frontières entre les classes ne sont pas parfaitement linéaires. L'utilisation d'un noyau RBF avec $\mathbf{C = 1}$ et **gamma = 0.01** permet d'obtenir une meilleure séparation des classes et d'améliorer les performances du modèle.

Différences entre SVM linéaire et SVM avec noyau RBF

- **SVM linéaire** : Convient lorsque les données sont parfaitement séparables par une frontière linéaire. Cependant, dans notre cas, il ne capture pas suffisamment bien la structure des données.
- **SVM avec noyau RBF** : Plus adapté lorsque les frontières de décision sont non linéaires. En utilisant les paramètres $\mathbf{C = 1}$ et **gamma = 0.01**, nous obtenons une meilleure généralisation et une classification plus précise.

Ainsi, pour notre problème de classification binaire, le SVM avec **noyau RBF** s'avère être le choix optimal, offrant un bon compromis entre flexibilité et performance.

Ainsi, le SVM linéaire est un choix optimal pour notre problème, offrant une bonne précision tout en étant plus simple et plus efficace en termes de calcul.

10 Conclusion

Les vecteurs supports sont les points situés au plus proche de la frontière de décision et jouent un rôle essentiel dans sa détermination.