UE C0 - Apprentissage automatique

TP 3 - Support Vector Machine

Thomas Boulanger Jules Farnault

26 novembre 2024

Table des matières

Introduction

Ce TP a pour objectif de nous initier à la création de SVM (Support Vector Machine - Séparateur à Vaste Marge). Pour cela, nous utiliserons la bibliothèque scikit-learn pour mettre en place :

1. un SVM dans le cas d'un problème linéairement séparable
2. un SVM dans le cas d'un problème non linéairement séparable
3. un SVM dans le cas du jeu de données MNIST

Pour cela, nous devrons mettre en place le jeu de données (sauf dans le cas des chiffres manuscrit qui utilise un jeu de données publiques), chercher les meilleurs paramètres du SVM, visualiser les résultats.

L'ensemble des codes réalisés sont disponible sur Github

1 Problème linéairement séparable

Dans cette partie, nous allons mettre en place un SVM dans le cadre d'un problème linéairement séparable.

1.1 - Création des données

On commence par mettre en place un jeu de données correspondant à notre problème. On utilise deux nuages de points gaussiens que l'on sépare fortement

```
def genere_ex_1(n1=100, n2=50, mu1=[0, 3], mu2=[3, 0], sd1=0.15,
    sd2=0.2):
    X = np.concatenate(
        (
            np.random.multivariate_normal(mu1, np.diagflat(sd1 *
                np.ones(2)), n1),
            (np.random.multivariate_normal(mu2, np.diagflat(sd2 *
                np.ones(2)), n2)),
        )
    )
    Y = np.concatenate((np.ones((n1, 1)), -1 * np.ones((n2, 1))))[:, 0]
    return X, Y
```

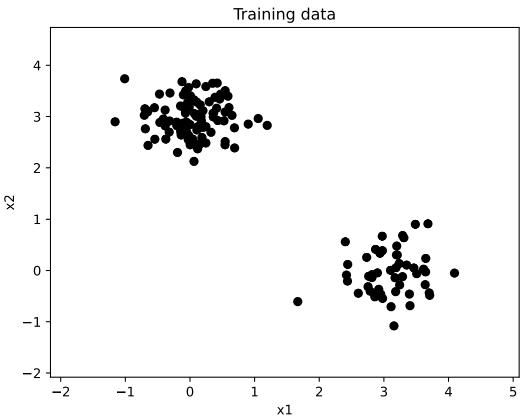


FIGURE 1.1 – Jeu de données

On remarque que notre jeu de données donne bien deux ensembles de points linéairement séparables.

1.2 - Création du modèle SVM

On met maintenant en place le SVM et on l'entraine sur le jeu de données

```
X, Y = genere_ex_1()
classifier = SVC(kernel="linear", probability=True)
classifier = classifier.fit(X, Y)

w = classifier.coef_[0]
b = classifier.intercept_[0]
```

1.3 - Résultat et visualisation

Nous mettons en place une fonction de visualisation qui permet d'afficher les marges et les probabilités d'appartenance.

```
def plot_data_hyperplan(X, Y, classifier, title, show_probability=False,save=False):
    # Create a mesh to plot in
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.02),
                         np.arange(y_min, y_max, 0.02))

```

```python
    plt.figure(figsize=(10, 8))

    if show_probability:
        # Plot the probability gradient
        Z = classifier.predict_proba(np.c_[xx.ravel(), yy.ravel()])[:, 1]
        Z = Z.reshape(xx.shape)
        cs = plt.contourf(xx, yy, Z, levels=np.linspace(0, 1, 11), cmap=plt.cm.RdYlBu, alpha=0.8)
        plt.colorbar(cs, label='Probability')

    Z = classifier.decision_function(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)

    # Plot the hyperplane

    plt.contour(xx, yy, Z, colors=['red', 'black', 'blue'], levels=[-1, 0, 1], alpha=1, linestyles=['-', '-', '-'])

    # Plot the training points
    scatter = plt.scatter(X[:, 0], X[:, 1], c=Y, cmap=plt.cm.RdYlBu, edgecolor='black')

    # Add legend
    legend1 = plt.legend(*scatter.legend_elements(),
                         loc="upper right", title="Classes")


    plt.xlabel('x1')
    plt.ylabel('x2')
    plt.title(title)
    plt.xlim(x_min, x_max)
    plt.ylim(y_min, y_max)
    if save:
        plt.savefig(f'plots/{title}.pdf')
    plt.show()
    plt.close()
```

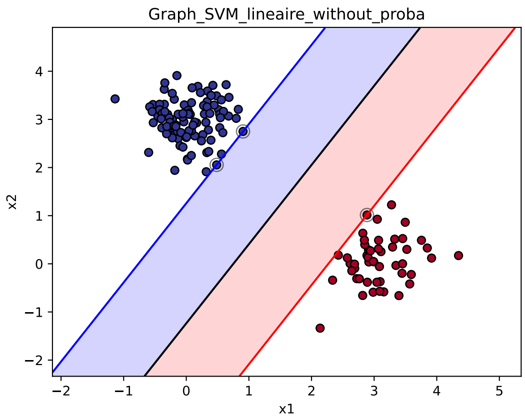


FIGURE 1.2 – Résultat du SVM avec la marge

Le résultat est positif, le séparateur passe bien au milieu et maximise la marge entre les points bleus et les points rouges. On va maintenant chercher à savoir comment évolue la probabilité d'appartenance dans le plan.

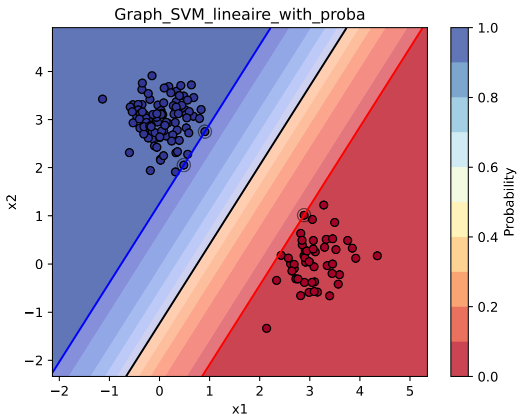


FIGURE 1.3 – Affichage du SVM avec les probabilités d'appartenance

On remarque que plus, on est proche du séparateur, moins on est sûr de l'appartenance et plus on s'en éloigne, plus on est sûr de l'appartenance au groupe rouge ou bleu.

2 | Problème non linéairement séparable

On va maintenant passer à un problème ou les données ne sont pas linéairement séparables.

2.1 - Création des données

Pour les données, nous générons un nuage gaussien centré en 0, traversé par une fonction polynomiale de degré 3 :

$$x(x-1)(x+1)$$

On ajoute delta à l'ordonnée des points au-dessus de cette frontière et on retranche delta à l'ordonnée des points en dessous de cette frontière. Cela aura pour effet de créer une frontière de taille 2 fois delta le long de la fonction.

On utilise dans la suite $\delta = 0.2$.

```
def genere_ex_2(n=300, mu = [0,0], std=0.25, delta = 0.2):
    X = np.random.multivariate_normal(mu, np.diagflat(std*np.ones(2)),n)
    Y = np.zeros((X.shape[0]))
    for i in range(X.shape[0]):
        x = X[i,0]
        y = X[i,1]
        if y < x*(x-1)*(x+1):
            Y[i] = -1
            X[i,1] = X[i,1] - delta
        else:
            Y[i] = 1
            X[i,1] = X[i,1] + delta
    return X,Y
```

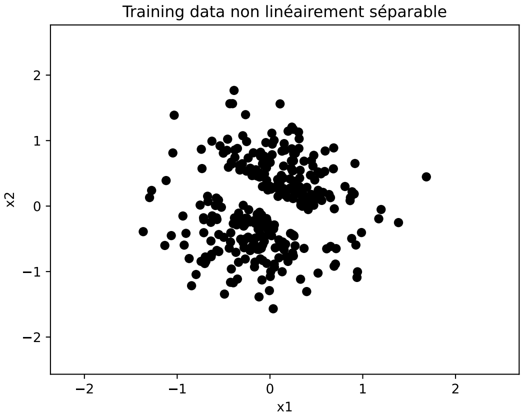


FIGURE 2.1 – Jeu de données

On remarque que l'on a un jeu de données avec une séparation qui ne peut pas être linéaire si on ne veut pas d'erreur. On est donc bien dans un problème non linéairement séparable.

2.2 - Création du modèle SVM

Dans la création du SVM, nous allons utiliser un noyau de type rbf qui est recommandé dans le cadre de problème non linéairement séparable et le noyau polynomial, car notre frontière est polynomiale par construction.

Pour trouver les meilleurs paramètres, nous allons utiliser la fonction GridSearchCV. Cette fonction demande des listes de paramètres et cherchent ceux qui répondent le mieux au problème. Nous allons chercher les paramètres C (le paramètre de régularisation), dans le cas polynomial, on ajoutera le degré du polynôme.

2.2.1 - Kernel rbf

```
param_grid = {
        'C': LC,         # Paramètre de régularisation
        'kernel': ['rbf'],               # Type de noyau
        'probability' : [True],
    }

# Configuration de GridSearchCV
grid_search = GridSearchCV(estimator=model, param_grid=param_grid, cv=5,
    scoring='precision', n_jobs=-1, verbose=2)
grid_search.fit(X, Y)

```

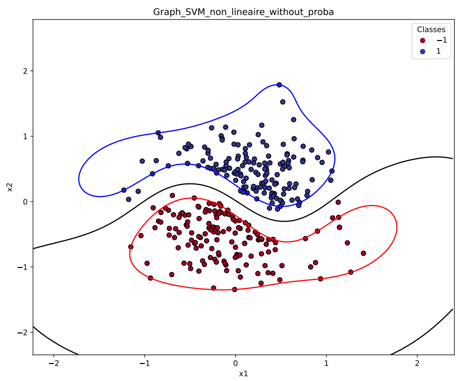
```
# Afficher les résultats
print("Meilleurs paramètres :", grid_search.best_params_)
print("Meilleure précision :", grid_search.best_score_)

classifier = SVC(**grid_search.best_params_).fit(X,Y)
```

2.2.2 - Kernel polynomiale

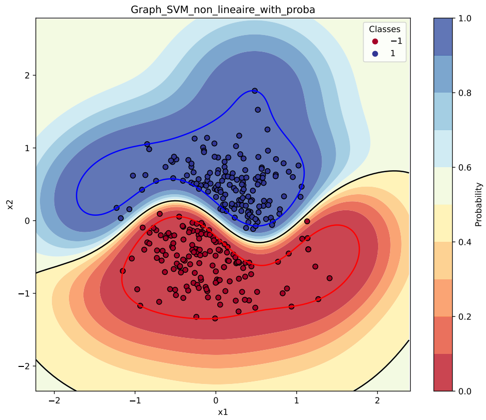
```
param_grid = {
        'C': LC,         # Paramètre de régularisation
        'degree': Ldeg,
        'kernel': ['poly'],               # Type de noyau
        'probability' : [True],
    }

# Configuration de GridSearchCV
grid_search = GridSearchCV(estimator=model, param_grid=param_grid, cv=5,
    scoring='precision', n_jobs=-1, verbose=2)
grid_search.fit(X, Y)

# Afficher les résultats
print("Meilleurs paramètres :", grid_search.best_params_)
print("Meilleure précision :", grid_search.best_score_)

classifier = SVC(**grid_search.best_params_).fit(X,Y)
```

2.3 - Résultat et visualisation

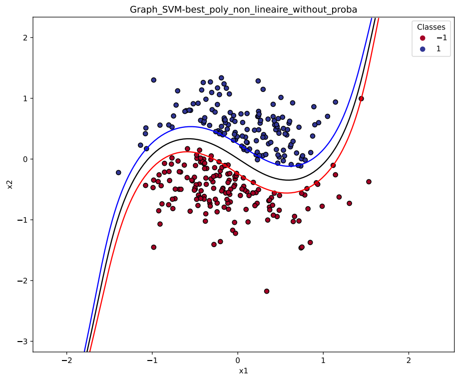
On utilise la même fonction que précédemment afin de visualiser en 2D les résultats avec et sans la probabilité.



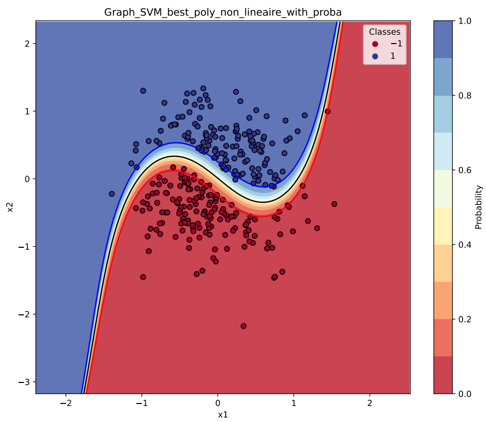
(a) Visualisation de la séparatrice et des bornes, kernel = RBF



(b) Visualisation avec la probabilité



(c) Visualisation de la séparatrice et des bornes, kernel = poly



(d) Visualisation avec la probabilité

FIGURE 2.2 – Résultats sur le problème non linéaire

On remarque que les deux kernels permettent de bien identifier les classes, mais leur approche sont très différentes :

- RBF encercle les deux classes avec une forme fermée. Un point hors de ces zones aura une probabilité proche de 50%, c'est-à-dire que la donnée est difficile à classifier.
- Poly sépare en deux les groupes avec un polynôme d'ordre 3. Cette séparation est plus généraliste et identifiera avec une forte probabilité un point même s'il est très éloigné de l'ensemble d'apprentissage.

3 Reconnaissance de chiffres manuscrits

Le jeu de données MNIST est un ensemble d'images de chiffres manuscrits employés pour l'entraînement et l'évaluation de modèles en vision par ordinateur et en apprentissage automatique. Le jeu de données contient 60 000 images pour l'entraînement et 10 000 images de tests. Chaque image est de taille 28 pixels par 28 pixels et représente un chiffre écrit à la main en niveau de gris. En plus des images, il y a les résultats qui permettent de faire de l'apprentissage supervisé.

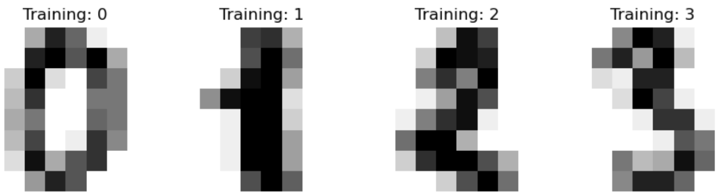


FIGURE 3.1 – Exemple d'images du jeu de données MNIST

On utilise la fonction GridSearchCV avec le choix du noyau parmi poly, rbf et sigmoid.

```python
param_grid = {
    'kernel': ['poly', 'rbf', 'sigmoid'],
    'C': np.logspace(-2,2,20),  # Regularization parameter
    'gamma': ['scale', 'auto'],  # Kernel coefficient for 'rbf', 'poly' and 'sigmoid'
    'degree': list(range(1,8))
}

# Create a Support Vector Classifier (SVC) model
model = SVC()

grid_search = GridSearchCV(estimator=model, param_grid=param_grid,scoring='accuracy', cv=5, n_jobs=-1,verbose=2)
    param_grid = {
    'kernel': ['poly', 'rbf', 'sigmoid'],
```

```python
    'C': np.logspace(-2,2,20),  # Regularization parameter
    'gamma': ['scale', 'auto'],  # Kernel coefficient for 'rbf', 'poly' and 'sigmoid'
    'degree': list(range(1,8))
}

# Fit the grid search to the training data
grid_search.fit(X_train, y_train)

# Print the best parameters and best score found by GridSearchCV
print(f"Best parameters found: {grid_search.best_params_}")
print(f"Best cross-validation accuracy: {grid_search.best_score_:.2f}")

# Evaluate the best model on the test set
best_model = grid_search.best_estimator_
test_accuracy = best_model.score(X_test, y_test)
print(f"Test set accuracy: {test_accuracy:.2f}")
```

La fonction nous retourne que les meilleurs paramètres sont :

— kernel : rbf

— C : 3.35

— γ : scale

En utilisant ses paramètres, on a 97% de bon résultat. Parmi les mauvais résultats, on peut trouver ceux ci-dessous :

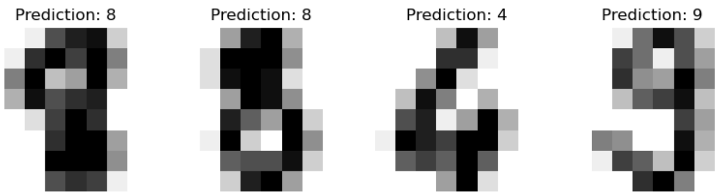


FIGURE 3.2 – Exemple d'erreur de prédiction du jeu de données MNIST

Les chiffres portent à confusion, même pour les humains, ce qui peut expliquer les erreurs.

On peut tracer la matrice de confusion :

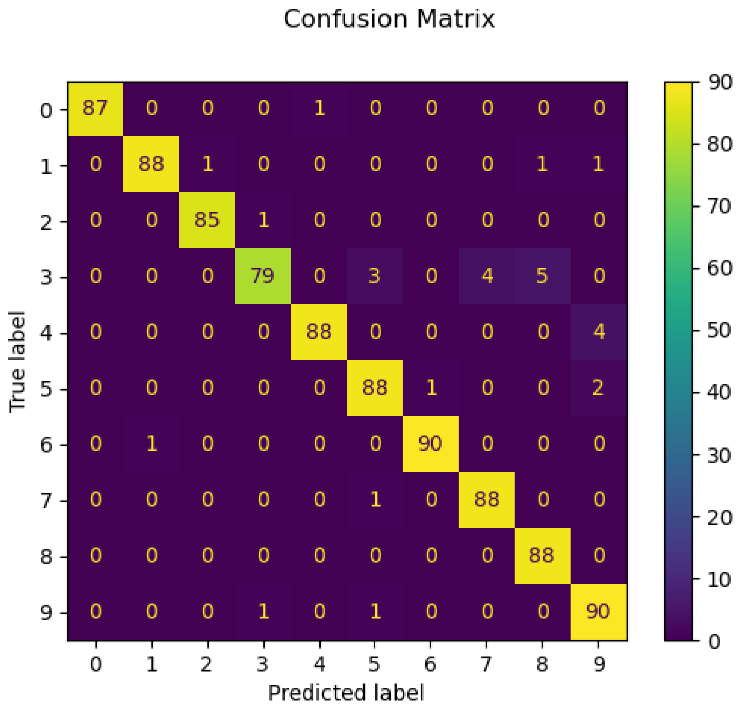


FIGURE 3.3 – Matrice de confusion résultant

Les erreurs sont sur des chiffres qui se ressemblent calligraphiquement (exemple 3 et 8 ou 4 et 9). La pire prédiction possible est pour le chiffre 3 qui ressemble à pas mal d'autres chiffres (5,7,8).

Conclusion

Lors de ce TP, nous avons implémenté divers modèles de SVM afin de résoudre des problèmes linéairement séparables, non linéairement séparable et enfin un problème de classification de chiffres manuscrit.