TD-TP SVM linéaire pour la classification

November 17, 2024

1 Exercice 1 : problème de classification binaire et SVM (à rendre)

On considère ici un problème de classification binaire vers $\mathcal{Y} = \{-1, +1\}$ de données dans un espace de description $\mathcal{X} \subset \mathbb{R}^d$. On note $\{(\mathbf{x}_i, y_i) \in (\mathcal{X}, \mathcal{Y})\}, i \in \{1, \dots, n\}$ l'ensemble d'apprentissage considéré. La fonction de décision du classifieur considéré est donnée par :

$$f_{\mathbf{w},b}(\mathbf{x}) = \text{sign}(\langle \mathbf{w}, \mathbf{x} \rangle + b), \quad \text{avec} \quad h(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle + b.$$

1.1 Marge

On suppose que $\frac{\mathbf{w}}{\|\mathbf{w}\|}$ est un vecteur unitaire orthogonal à la frontière de décision,

1. Faire un dessin
2. Donnerl'expression de la distance $d(\mathbf{x}_i, \mathbf{w}, b)$ d'un point $\mathbf{x}_i$ à la frontière de décision.
3. Montrer que cette distance ne change pas lorsqu'on multiplie la solution $(\mathbf{w}, b)$ par un scalaire, i.e. pour $(\alpha\mathbf{w}, \alpha b)$. Que cela implique-t-il si l'on souhaite éloigner au maximum (au sens géométrique) les points de la frontière de décision ?

1.2 Formulation du SVM

On considère alors le problème d'optimisation sous contraintes suivant :

$$\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2,$$

$$\text{s.t. } y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) \geq 1, \quad \forall i \in \{1, \dots, n\}.$$

1. Pourquoi choisit-on la contrainte ≥ 1 plutôt que ≥ 0 ? Pourquoi 1 ?
2. Poser le Lagrangien à considérer pour optimiser ce problème sous contraintes.
3. Donner la solution analytique de la minimisation de ce Lagrangien selon $\mathbf{w}$ et b.
4. En déduire une nouvelle formulation "duale" de notre problème d'optimisation sous contraintes.
5. Que cette nouvelle formulation permet-elle ?

6. Donner le classifieur obtenue après optimisation de la formulation duale du SVM à marge dure.

7. D'après les conditions de Karush-Kuhn-Tucker (KKT) concernant les propriétés de la solution optimale d'un Lagrangien, on a :

$$a_i(y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) - 1) = 0, \quad \forall i \in \{1, \dots, n\},$$

   Qu'en déduire pour les paramètres a_i obtenus à l'optimum ?

8. Déduire l'estimation de b à partir des conditions KKT ?

2 Exercice 2 : Je code mon propre SVM

2.1 Cas de données linéairement séparables

L'objectif est d'implémenter une solution basique de Machine à Vecteurs de Support (SVM) pour un problème de classification linéairement séparable à l'aide de CVXPy. Il est important de noter que nous ne recherchons pas une implémentation évolutive mais une preuve de concept. Pour un solveur SVM évolutif, il est préférable d'envisager l'utilisation de Scikit Learn pour des ensembles de données de grande taille.

[ ]:
```
import numpy as np
import matplotlib.pyplot as plt
import cvxpy as cp
```

Generation des données d'apprentissage : La fonction ci-dessous génère des données synthétiques pour deux classes distinctes.

[ ]:
```
def generate_synthetic_data(num_points_per_class=8, seed=None):
    """
    Génère des données synthétiques pour deux classes.

    Parameters:
    - num_points_per_class: int, nombre de points par classe.
    - seed: int, valeur pour initialiser le générateur aléatoire (pour␣
 ↪reproductibilité).

    Returns:
    - X: numpy.ndarray, coordonnées des points (2D).
    - y: numpy.ndarray, classes des points (+1 ou -1).
    """
    if seed is not None:
        np.random.seed(seed)

    # Génération des points pour la classe +1
    class1 = np.random.multivariate_normal(mean=[-2, -2], cov=[[1, 0.5], [0.5,␣
 ↪1]], size=num_points_per_class)
```

```
    # Génération des points pour la classe -1
    class2 = np.random.multivariate_normal(mean=[2, 2], cov=[[1, -0.3], [-0.3,␣
↪1]], size=num_points_per_class)

    # Combinaison des deux classes
    X = np.vstack((class1, class2))
    y = np.hstack((np.ones(num_points_per_class), -np.
↪ones(num_points_per_class)))

    return X, y
```

Question : Tester cette fonction. Décriver les variables X et y ?

[ ]:

La fonction suivante affiche le nuage de points correspondant à chacune des classes générées précédemment.

[ ]:
```
def plot_points(X, y):
    plt.scatter(X[y == 1][:, 0], X[y == 1][:, 1], c='r', label='Classe +1')
    plt.scatter(X[y == -1][:, 0], X[y == -1][:, 1], c='b', label='Classe -1')
    plt.legend()
    plt.title("Données Synthétiques")
    #plt.show()
```

Dans ce qui suit, on cherche à résoudre le problème du SVM formulé dans l'exercice 1

Question: completer la fonction suivante pour qu'elle renvoie la solution du problème primal **w** et b

[ ]:

[ ]:
```
def solve_svm(X, y):
    """
    Résout un problème SVM linéaire en utilisant CVXPY.

    Parameters:
    - X: numpy.ndarray, matrice des caractéristiques (shape: n_samples x␣
↪n_features).
    - y: numpy.ndarray, vecteur des étiquettes (+1 ou -1) (shape: n_samples).

    Returns:
    - w: numpy.ndarray, vecteur des poids appris (shape: n_features).
    - b: float, biais appris.
    - status: str, statut de la solution (e.g., "optimal").
    """
    n, f = X.shape  # Dimensions des données (n_samples, n_features)
```

```
    # Variables d'optimisation
    w = cp.Variable(f)
    b = cp.Variable()

    # Fonction objectif :
    objective = cp.Minimize(# a completer)

    # Contraintes
    constraints = []
    for i in range(n):
        constraints.append(# a completer)

    # Définir et résoudre le problème d'optimisation
    problem = cp.Problem(# a completer)
    problem.solve()


    return # a completer # Extraire les valeurs optimales de w et b
```

Question Tester cette fonction sur les données générées. Afficher le vecteurs des parametres optimale $\mathbf{w}$ et le biais b. Afficher les vecteurs de supports.

[ ]:

Fonction de visualisation de la frontière de décision et de la marge Completer la fonction suivante pour afficher la frontière de décision ainsi que les marge

[ ]:
```
def plot_decision_boundary_with_margins(w, b, X_range=(-5, 5), Y_range=(-5, 5),␣
↪title="Frontière de décision et marges SVM"):
    """
    Affiche la frontière de décision et les marges d'un SVM linéaire.

    Parameters:
    - w: numpy.ndarray, vecteur des poids appris.
    - b: float, biais appris.
    - X_range: tuple, plage des valeurs pour l'axe x (min, max).
    - Y_range: tuple, plage des valeurs pour l'axe y (min, max).
    - title: str, titre du graphique.
    """
    plt.figure(figsize=(8, 6))
    plot_points(X, y)

    # Plage de valeurs x pour afficher la frontière et les marges
    x1 = np.linspace(X_range[0], X_range[1], 100)
```

```
    # Calcul de x2 pour les trois lignes : frontière, marge positive et marge␣
↪négative
    decision_boundary = # a complter # Frontière de décision
    margin_positive = # a completer  # Marge positive
    margin_negative = # a completer # Marge négative

    # Tracer les trois lignes
    plt.plot(x1, decision_boundary, color='green', label='Frontière de␣
↪décision')
    plt.plot(x1, margin_positive, '--', color='m', label='Marge positive')
    plt.plot(x1, margin_negative, '--', color='m', label='Marge négative')

    # Définir les limites du graphique
    plt.xlim(X_range)
    plt.ylim(Y_range)

    # Ajouter des labels et une légende
    plt.title(title)
    plt.xlabel('x1')
    plt.ylabel('x2')
    plt.legend()
    plt.grid(True)
    plt.show()
```

Afficher le resultat

[ ]:

3 Et pour finir, je m'approprie scikit-learn

[ ]:
```
"""
a installer !
!pip3 -q install sklearn
!pip3 -q install matplotlib
!pip3 -q install seaborn
"""
```

Cette partie est librement inspirée du travail de Jake VenderPlas, auteur du livre Python Data Science Handbook. Son GitHub (en anglais) regorge de fichiers utiles.

Dans un premier temps, on va générer des données jouets, linéairement séparables :

[ ]:
```
%matplotlib inline
import matplotlib.pyplot as plt

#Un petit environment qui donne de meilleurs graphes
```

```
import seaborn as sns; sns.set()

# fonction sklearn pour générer des données simples
from sklearn.datasets import make_blobs
X, y = make_blobs(n_samples=50, centers=2,
                  random_state=0, cluster_std=0.60)

# Affichage des données
plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='prism');
```

On va commencer par apprendre un SVM linéaire (sans noyau) à l'aide de scikit-learn :

[ ]:
```
#import de la classe - qui s'appelle SVC et pas SVM...
from sklearn.svm import SVC
#Définition du modèle
model = SVC(kernel='linear', C=1E10)
#Apprentissage sur les donnée
model.fit(X, y)
```

On va utiliser une fonction d'affichage qui va bien, où tout ce qui est nécessaire est affiché.

[ ]:
```
import numpy as np

def affiche_fonction_de_decision(model, ax=None, plot_support=True):
    """Affiche le séparateur, les marges, et les vecteurs de support d'un SVM␣
 ↪en 2D"""
    if ax is None:
        ax = plt.gca()
    xlim = ax.get_xlim()
    ylim = ax.get_ylim()

    # création de la grille pour l'évaluation
    x = np.linspace(xlim[0], xlim[1], 30)
    y = np.linspace(ylim[0], ylim[1], 30)
    Y, X = np.meshgrid(y, x)
    xy = np.vstack([X.ravel(), Y.ravel()]).T
    P = model.decision_function(xy).reshape(X.shape)

    # affichage de l'hyperplan et des marges
    ax.contour(X, Y, P, colors='k',
               levels=[-1, 0, 1], alpha=0.5,
               linestyles=['--', '-', '--'])

    # Affichage des vecteurs de support
    if plot_support:
        ax.scatter(model.support_vectors_[:, 0],
                   model.support_vectors_[:, 1],
```

```
                    s=300, linewidth=1, facecolors='none', edgecolor='black');
    ax.set_xlim(xlim)
    ax.set_ylim(ylim)
```

Voyons ce que cela donne sur notre séparateur linéaire à vaste marge :

[ ]:
```
plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='prism')
affiche_fonction_de_decision(model);
```

Sur ce graphe, on voit le séparateur (ligne pleine), les vecteurs de support (points entourés) et la marge (matérialisée par des lignes discontinues). On a ici le séparateur qui maximise la marge. Scikit-learn nous permet, après apprentissage, de récupérer les vecteurs de supports:

[ ]:
```
model.support_vectors_
```

Seules trois données sont utiles pour classer de nouvelles données. On peut s'en assurer en rajoutant des données sans changer le modèle :

[ ]:
```
X2, y2 = make_blobs(n_samples=200, centers=2,
                    random_state=0, cluster_std=0.60)

model2 = SVC(kernel='linear', C=1E10)
model2.fit(X2, y2)

plt.scatter(X2[:, 0], X2[:, 1], c=y2, s=50, cmap='prism')
affiche_fonction_de_decision(model2);
```

[ ]:
```

```