Метод опорних векторів

Означення

Метод опорних векторів (Support Vector Machine, SVM) — це алгоритм машинного навчання для класифікації та регресії, який працює шляхом знаходження гіперплощини, що найкраще розділяє два або більше класи даних.

§ 1. Суть методу

У SVM дані представляються у вигляді векторів у n-вимірному просторі (де n — кількість ознак у даних), і для розділення класів шукається гіперплощина (тобто $(n-1)$-вимірна площина у n-вимірному просторі), що максимально відділяє класи один від одного.

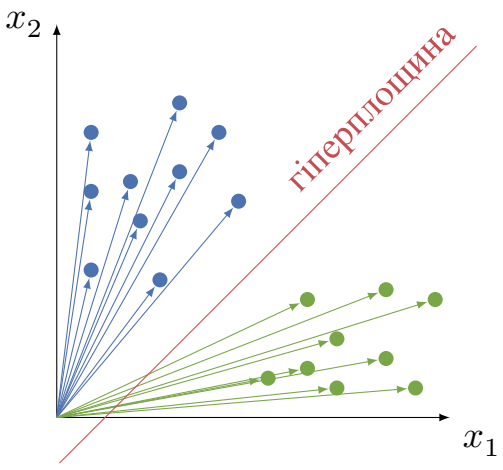


Рис. 1. Ілюстрація суті методу опорних векторів

Оскільки може бути декілька таких гіперплощин, то SVM знаходить ту, яка максимізує мінімальну відстань між гіперплощиною та найближчими до неї точками обох класів (це відстань називається «мінімальною шириною розділяючої смуги» або «margin»).

SVM може використовуватися для класифікації даних, які мають два або більше класів, а також для регресії, тобто для передбачення числового значення залежної змінної на основі інших ознак даних. SVM є потужним алгоритмом машинного навчання, що зазвичай дає дуже хороші результати для класифікації даних у випадку, коли класи добре розділені гіперплощиною.

§ 2. Математична модель

2.1. Жорстке розділення

Означення

Жорстке розділення (або hard margin) — це концепція, яка описує ситуацію, коли алгоритм SVM прагне знайти таку гіперплощину (межу), яка повністю розділяє два класи даних без помилок.

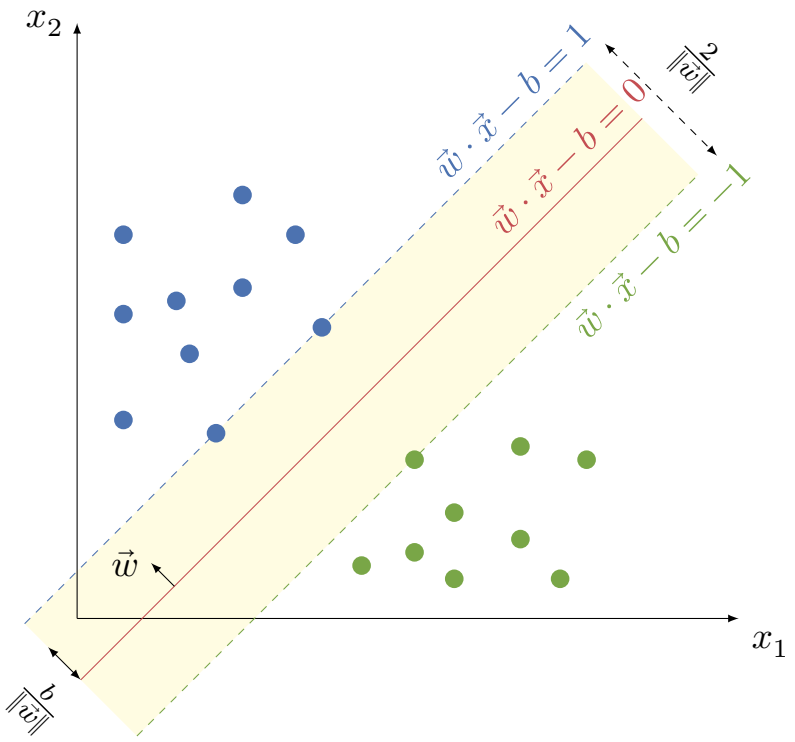


Рис. 2. Ілюстрація методу опорних векторів

Сформулюємо попереднє визначення більш строго та математично. В своїй основі ми маємо знову лінійну модель, і позначення ми використовуємо як і в попередніх модулях.

В нас є тренувальний набір даних з n точок вигляду $(\vec{x}_1, y_1), \ldots, (\vec{x}_n, y_n)$, де y_i є або 1, або -1, і кожен з них вказує клас, до якого належить точка $\vec{x}_i$. Кожен $\vec{x}_i$ є n-вимірним вектором.

Нам треба знайти «максимально розділову гіперплощину», яка відділяє групу точок $\vec{x}_i$, для яких $y_i = 1$, від групи точок, для яких $y_i = -1$, і визначається таким чином, що відстань між цією гіперплощиною та найближчою точкою $\vec{x}_i$ з кожної з груп є максимальною.

Якщо тренувальні дані лінійно роздільні, то ми можемо обрати дві паралельні гіперплощини, які розділяють два класи даних так, що відстань між ними якомога більша. Область, обмежена цими двома гіперплощинами, називається «розділенням» (margin), а максимально розділова гіперплощина це гіперплощина, яка лежить посередині між цими двома. Ці гіперплощини може бути описано рівняннями:

$\vec{w} \cdot \vec{x} - b = 1,$ (будь-що на або вище цієї межі належить до класу з міткою 1)

$\vec{w} \cdot \vec{x} - b = -1,$ (будь-що на або вище цієї межі належить до класу з міткою -1)

Оскільки ми також маємо завадити потраплянню точок даних до розділення, ми додаємо таке обмеження:

$$\begin{cases} \vec{w} \cdot \vec{x}^{(i)} - b \geqslant 1, & \text{якщо } y^{(i)} = 1, \\ \vec{w} \cdot \vec{x}^{(i)} - b \leqslant -1, & \text{якщо } y^{(i)} = -1 \end{cases} \tag{1}$$

Ці обмеження стверджують, що кожна точка даних мусить лежати з правильного боку розділення.

Таке розділення класів гіперплощиною називається жорстким, оскільки така модель не дозволяє робити помилок, і різні класи чітко лежать по різні сторони гіперплощини, що розділяє класи. Геометрично ширина смуги, що розділяє класи (margin) дорівнює $\frac{2}{\|\vec{w}\|}$, де $\|\vec{w}\|$ норма вектору вагових коефіцієнтів, що рівна кореню зі скалярного добутку вектора вагових коефіцієнтів самого на себе, тобто:

$$\|\vec{w}\| = \sqrt{\vec{w} \cdot \vec{w}} = \sqrt{\sum_{i=1}^{n} w_i^2}.$$

І оскільки ми хочемо максимізумати ширину розділяючої смуги:

$$\frac{2}{\|\vec{w}\|} \to \max,$$

то такий тип класифікаторів називають maximum margin.

В свою чергу, проблема знаходження максимуму еквівалентна до проблеми знаходження мінімуму:

$$\frac{1}{2}\|\vec{w}\|^2 \to \min,$$

тобто величина норми вектора вагових коефіцієнтів повинна бути мінімальною. А тому, ми можемо привести цю задачу до оптимізаційної і використати алгоритм градієнтного спуску та знайти оптимальні значення коефіцієнтів w_i[1].

2.2. М'яке розділення

Жорстке розділення — це ідеальна ситуація, в реальних даних жорстке розділення може бути неможливим через наявність шуму або перекриття класів.

Означення

Для задач, у яких дані не можна жорстко розділити, використовують гнучкіший метод, який називають м'яким розділенням (soft margin), що дає змогу деяким об'єктам перебувати на «неправильній» стороні межі з певною помилкою, але водночас намагається мінімізувати цю помилку й забезпечити найкраще розділення класів.

Дозволимо нашому класифікатору робити помилки, проте нам потрібно оцінити наскільки сильно помиляється класифікатор і додати ці оцінки у функцію втрат.

Спочатку вияснимо, як порахувати такі оцінки.

Нерівності (1) можна переписати в одну:

$$y_i \left[\vec{w} \cdot \vec{x}^{(i)} - b\right] \geqslant 1 \tag{2}$$

Нам необхідно придумати таку функцію, яка дорівнює 0, якщо обмеження в (2) задовольняється, іншими словами, якщо x_i лежить із правильного боку розділення. Для даних із неправильного боку розділення значення цієї функції є пропорційним до відстані M_i від розділення. Ця відстань дорівнює $M_i = y_i \left[\vec{w} \cdot \vec{x}^{(i)} - b\right]$. В результаті ми отримаємо таку функцію втрат:

$$\max\left(0, 1 - y_i \left[\vec{w} \cdot \vec{x}^{(i)} - b\right]\right) \tag{3}$$

Ця функція називається завісною функцією втрат (англ. hinge loss function).

[1]Зауважимо, що останнє визначення довжини вектору може нам дещо нагадати — а саме доданок L_2-регуляризації.

Отже, повна функція втрат :

$$\text{Loss} = C\sum_{i=1}^{m}\max\left(0, 1 - y_i\left[\vec{w}\cdot\vec{x}^{(i)} - b\right]\right) + \frac{1}{2}\|\vec{w}\|^2 \tag{4}$$

де гіперпараметр C визначає компроміс між збільшенням розміру розділення та забезпеченням того, що x_i лежить із правильного боку розділення. Для достатньо малих значень C SVM із м'яким розділенням поводитиметься однаково з SVM із жорстким розділенням, якщо вхідні дані є лінійно класифіковними.

Поглянемо на графіки й порівняємо функцію втрат для логістичної регресії та методу опорних векторів (рис. 3).

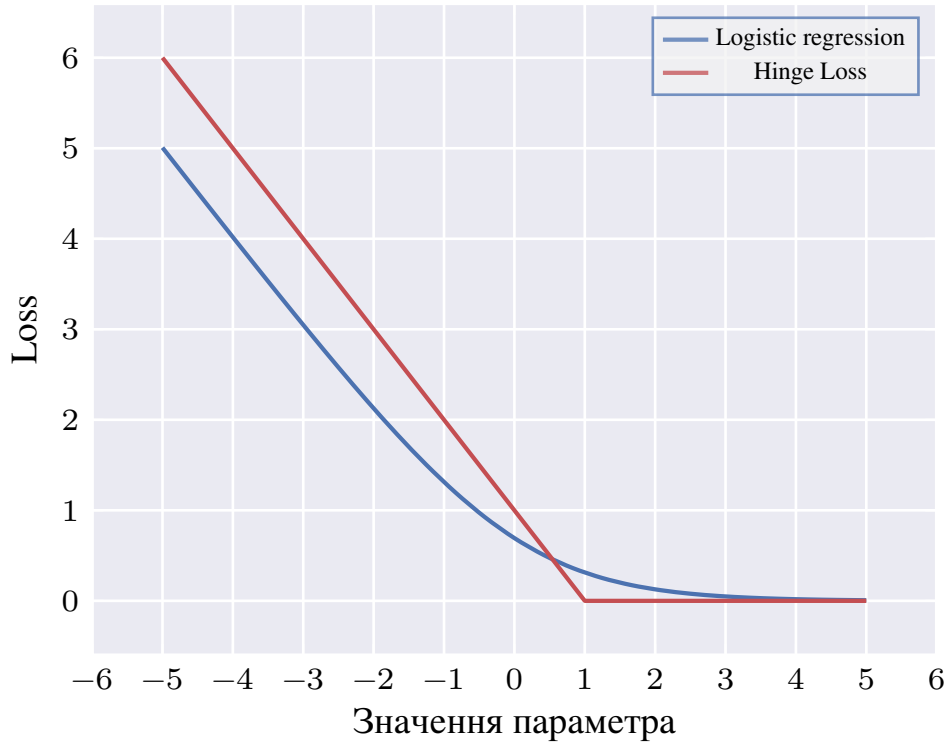


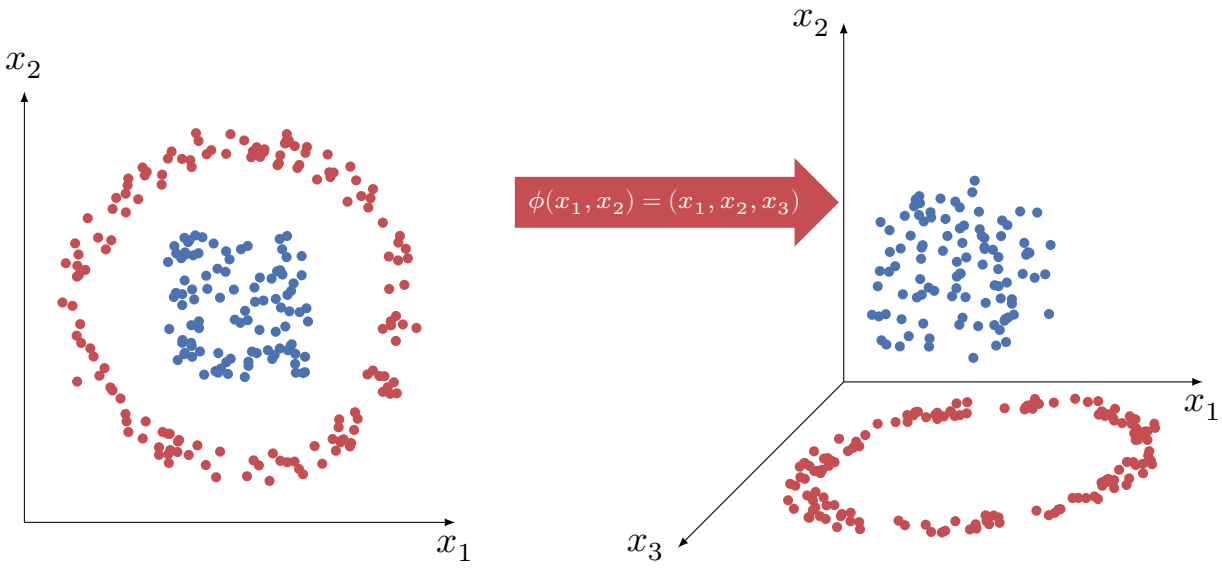
Рис. 3. Порівняння функції втрат для логістичної регресії та методу опорних векторів

§ 3. Ядровий трюк

Ось ми розібрались, як побудувати лінійний класифікатор за допомогою методу опорних векторів. Проте, як ми знаємо, часто ми не можемо лінійно розділити наші дані, як наприклад на рис. 4a.

Для цього ми повинні перейти до нового простору ознак, раніше ми займались створенням нових фіч, таких як x_j^2 чи x_j^3, у такому випадку ми повинні

перевести всі наші дані у новий векторний простір доповнений новими штучно створеними ознаками. Та це може бути досить дорого з точки зору обчислень, а ядровий трюк полягає у тому, що ми не робимо трансформацію даних, щоб відобразити їх у новому векторному просторі, а лише розраховуємо відстані між векторами так, наче ми вже перебуваємо у цьому просторі.



(а) Не роздільні ознаки в 2D просторі

(б) Додавання ознаки x_3

Рис. 4. Ілюстрація ядрового трюку

Це досягається за допомогою ядерної функції, яка обчислює скалярний добуток між векторами у вищорозмірному просторі, не потребуючи безпосереднього перетворення даних у цей простір. Це дозволяє SVM працювати зі складними нелінійно-роздільними даними, зберігаючи при цьому швидкість роботи алгоритму.

Математика ядрового трюку

Функція втрат (або функція втрат Хінджа) для лінійного SVM у бінарній класифікації може бути подана таким чином:

$$\max\left(0, 1 - y_i f(\vec{x}^{(i)})\right),$$

де в лінійному випадку $f(\vec{x}^{(i)}) = \vec{w} \cdot \vec{x}^{(i)} - b$.

Ядровий трюк полягає в заміні $\vec{x}^{(i)} \mapsto \phi(\vec{x}^{(i)})$ на нову функцію, яка переводить дані з вихідного простору в інший, можливо, більш високорозмірний простір, тоді:

$$f(\vec{x}^{(i)}) = \vec{w} \cdot \phi(\vec{x}^{(i)}) - b.$$

Вектор $\vec{w}$ можна записати як лінійну комбінацію опорних векторів (у просторі

ознак):

$$\vec{w} = \sum_{j=1}^{n} \alpha_j y_j \phi(\vec{x}^{(j)}).$$

де n — кількість об'єктів у навчальному наборі, а α_j — нові вагові коефіцієнти, які отримуються в результаті навчання SVM.

Тоді

$$f(\vec{x}) = \sum_{j=1}^{n} \alpha_j y_j \phi(\vec{x}^{(j)}) \phi(\vec{x}^{(i)}) - b,$$

Введемо ядро як:

$$K(\vec{x}^{(j)}, \vec{x}^{(i)}) = \phi(\vec{x}^{(j)}) \phi(\vec{x}),$$

тоді

$$f(\vec{x}^{(i)}) = \sum_{j=1}^{n} \alpha_j y_j K(\vec{x}^{(j)}, \vec{x}^{(i)}) - b,$$

Застосування ядерного трюку в SVM дозволяє підвищити точність класифікації та зменшити ризик перенавчання моделі. Проблеми, які можуть бути вирішені за допомогою SVM з ядерним трюком, включають класифікацію текстових даних, розпізнавання образів та прогнозування фінансових показників.

Існує кілька типів ядерних функцій, які можна використовувати з методом опорних векторів (SVM), залежно від характеристик даних та завдання класифікації. Ось деякі з найпоширеніших ядерних функцій:

- Лінійна функція ядра: $K(x, y) = xy$;
- Поліноміальна функція ядра: $K(x, y) = (xy + r)^d$, де r — константа зміщення. d — степінь полінома;
- Радіальна базисна функція (RBF) ядра: $K(x, y) = e^{-\gamma \cdot \|x-y\|^2}$, де γ — параметр ширини гаусівської функції;
- Сигмоїдальна функція ядра: $K(x, y) = \tanh(\alpha \cdot xy + r)$, де α — параметр швидкості зростання, r — константа зміщення.

Кожен тип ядерної функції має свої унікальні властивості та може бути корисним для різних завдань класифікації. Наприклад, лінійна функція ядра добре підходить для лінійно роздільних даних, тоді як RBF функція може допомогти з розділенням складних нелінійних даних.

3.1. Гаусівське ядро

Розглянемо більш детально радіально базисну функцію, котра також називається ще й гаусівським ядром. Напишемо формулу густини ймовірності гаусівського розподілу:

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}.$$

Де символи μ та σ позначають математичне очікування та стандартне відхилення відповідно.

Тоді як ядрова функція радіального базису має такий вигляд:

$$K(x, y) = e^{-\gamma \cdot \|x-y\|^2}.$$

Ну і якщо ми зробимо такі заміну $\gamma = \frac{1}{2\sigma^2}$, то побачимо що дані функції відрізняються лише множенням на скаляр.

Проте як саме працює ядровий трюк? Як ми бачимо з визначенням ядрових функції в розрахунках ми використовуємо два вектори $\vec{x}$ та $\vec{y}$ і ми розраховуємо відстань між ними у новому лінійному просторі. А далі, у якості нових ознак для нашої моделі, описаної вище, ми будемо брати нові ознаки — відстані між нашими приклади розраховані за допомогою нашої ядрової функції. Формалізуємо вище сказане твердження:

$$f_j^{(i)} = K(\vec{x}^{(i)}, \vec{x}^{(j)}) = e^{-\gamma \cdot \|\vec{x}^{(i)} - \vec{x}^{(j)}\|^2}.$$

Отже тепер ми маємо замість вектора $\vec{x}^{(i)} \in \mathbb{R}^n$ новий $f^{(i)} \in \mathbb{R}^m$. І тепер будемо використовувати нові ознаки і в новому просторі шукати розподільчу смугу і максимізувати її ширину.

§ 4. Реалізація методу SVM у бібліотеці `scikit-learn`

Вся вищенавадена математична модель вже є реалізована у знайомій нам бібліотеці, і ви можете ознайомитись більше з різними версіями даного методу за посиланням.

Як і інші класифікатори, SVC приймає на вхід два масиви: масив X форми (n_samples, n_fetures), що містить навчальні вибірки, та масив y міток класів (рядки або цілі числа) форми (n_samples):

In [1]:
```
from sklearn import svm
X = [[0, 0], [1, 1]]
y = [0, 1]
clf = svm.SVC()
clf.fit(X, y)
```

Out[1]:
```
SVC()
```

Після навчання, модель можна використовувати для прогнозування нових значень:

In [2]:
```
clf.predict([[2., 2.]])
```

Out[2]:
```
array([1])
```

Властивості опорних векторів можна знайти в атрибутах support_vectors_, support_ та n_support_:

In [3]:
```
# get support vectors
clf.support_vectors_
```

Out[3]:
```
array([[0., 0.],
       [1., 1.]])
```

In [4]:
```
# get indices of support vectors
clf.support_
```

Out[4]:
```
array([0, 1]...)
```

In [5]:
```
# get number of support vectors for each class
clf.n_support_
```

Out[5]:
```
array([1, 1]...)
```

Розглянемо більш складний приклад, який включає в себе генерацію випадкових даних, поділ даних на тренувальні та тестові, та, власне, саме навчання, і побудуємо візуалізацію даних та граничних площин:

```
# Import the necessary libraries
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC

# Generate synthetic data (two point classes)
X, y = datasets.make_classification(
```

```
    n_samples=100,
    n_features=2,
    n_informative=2,
    n_seaborn_redundant=0,
    n_clusters_per_class=1,
    random_state=42)

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42)

# Create and train the SVM model
svm_classifier = SVC(kernel='linear', C=1)
svm_classifier.fit(X_train, y_train)

# Output the classification accuracy on the test set
accuracy = svm_classifier.score(X_test, y_test)
print(f'Classification accuracy: {accuracy:.2f}')

# Create a grid to visualize the solution boundary
xx, yy = np.meshgrid(np.linspace(X[:, 0].min() - 1,
                                 X[:, 0].max() + 1, 100),
                     np.linspace(X[:, 1].min() - 1,
                                 X[:, 1].max() + 1, 100))
Z = svm_classifier.decision_function(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

# Data visualization and decision boundaries
plt.figure(figsize=(8, 6))
plt.contourf(xx, yy, Z,
             levels=[-1, 0, 1],
             colors=['r', 'g', 'b'], alpha=0.2)
plt.scatter(X[:, 0], X[:, 1], c=y,
            cmap=plt.cm.coolwarm,
            edgecolors='k')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.title('Reference vector method (SVM)')
plt.show()
```

Висновок

SVM (англ. Support Vector Machine) — це метод машинного навчання, який знайшов широке застосування в задачах класифікації та регресії. Основною ідеєю методу є побудова гіперплощини, яка розділяє два класи даних з максимальною можливою шириною проміжку між ними.

При роботі з SVM ми спочатку перетворюємо наші дані до більш високої розмірності за допомогою ядерної функції, що дозволяє лінійно нероздільні дані

перетворювати в лінійно роздільні. Потім ми побудовуємо гіперплощину, максимізуючи відстань між гіперплощиною і найближчими до неї точками (векторами опору). Ці точки називаються опорними векторами і вони є ключовими елементами моделі SVM.

Серед переваг методу SVM можна виділити високу точність класифікації в задачах з великою кількістю ознак та можливість використання різних ядерних функцій для перетворення даних, які підходять для різних типів даних та задач.

Однак, метод SVM може бути чутливий до викидів, тобто до даних, які значно відрізняються від інших даних у вибірці. Крім того, вибір оптимальних параметрів для SVM може бути часо- та ресурсомістким завданням.

У загальному, SVM є потужним та ефективним методом машинного навчання, який може бути застосований у багатьох різних задачах. Використання SVM може залежати від характеру даних та конкретних умов задачі, тому варто розглянути його як один з варіантів при вирішенні конкретної задачі.

Посилання

1. *Bhattacharyya S.* Support Vector Machine: Kernel Trick; Mercer's Theorem // Towards Data Science. — 2018.
2. *Ponomarenko S.* Notebook: Support Vector Machines (SVM). — URL: https://github.com/sergiokapone/DataScience/blob/main/Hw5/TeX/SVM.ipynb.
3. Support Vector Machines (SVM). — URL: https://scikit-learn.org/stable/modules/svm.html.