Домашнее задание №7: «Ядра SVM»

Дедлайн 1 (20 баллов): 27 апреля, 23:59
Дедлайн 2 (10 баллов): 4 мая, 23:59

Домашнее задание нужно написать на Python и сдать в виде одного файла. Правило именования файла: `name_surname_7.[py | ipnb]`. Например, если вас зовут Иван Петров, то имя файла должно быть: `ivan_petrov_7.py` или `ivan_petrov_7.ipnb`.

---

1 Реализуйте Линейный SVM через решение прямой задачи QP для SVM. Рекомендуется воспользоваться пакетом cvxopt [1]. Также обратите внимание на разреженное представление матрицы `cvxopt.spmatrix`, qp-солвер работает быстрее с разреженными матрицами. Функция `solvers.qp()` решает задачу следующего вида:

$$\begin{cases} \frac{1}{2}x^T P x + q^T x \to \min\limits_x \\ Gx \leq h \\ Ax = b \end{cases}$$

Необходимо решить следующую задачу оптимизации:

$$\begin{cases} \frac{1}{2}w^T w + C\sum\limits_i \xi_i \to \min\limits_{w,\xi} \\ \xi_i \geq 0 \\ y_i(w^T x_i + w_0) \geq 1 - \xi_i \quad \forall i = 1 \dots l \end{cases}$$

Сформулируем ее в виде задачи для QP-солвера:

$$x = (w, w_0, \xi)$$

$$P = \left[\begin{array}{c|c} I & 0 \\ \hline 0 & 0 \end{array}\right] \qquad q = \left[\begin{array}{c} 0 \\ \hline C \cdot 1 \end{array}\right]$$

$$G = \left[\begin{array}{c|c|c} 0 & 0 & -I \\ \hline -y \odot X & -y & -I \end{array}\right] \qquad h = \left[\begin{array}{c} 0 \\ \hline -1 \end{array}\right]$$

Объект x_i является опорным, если в оптимальной точке для задачи линейного SVM неравенство отступов переходит в равенство:

$$y_i(w^T x_i + w_0) = 1 - \xi_i$$

[1] `http://cvxopt.org/userguide/index.html`

Структура класса приведена ниже:

```python
class LinearSVM:
    def __init__(self, C):
        self.C = C
        ...

    def fit(self, X, y):
        ...

    def decision_function(self, X):
        ...

    def predict(self, X):
        sign(self.decision_function(X))
```

2 Реализуйте Ядровой SVM через решение двойственной задачи QP для SVM.

Необходимо решить следующую задачу оптимизации:

$$\begin{cases} \sum_i \alpha_i - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j K(x_i, x_j) \to \max\limits_{\alpha} \\ 0 \leq \alpha_i \leq C, \quad \forall i = 1, \dots, l \\ \sum_i \alpha_i y_i = 0 \end{cases}$$

Сформулируем ее в виде задачи для QP-солвера:

$$P = \left[y_i y_j K(x_i, x_j)\right] \qquad q = [-1]$$

$$G = \left[\begin{array}{c} I \\ \hline -I \end{array}\right] \qquad h = \left[\begin{array}{c} C \cdot 1 \\ \hline 0 \end{array}\right]$$

$$A = y^T \qquad b = 0$$

Объект x_i является опорным, если $\alpha_i > 0$.

Решение принимается по следующему правилу:

$$a(x) = \operatorname{sign}\left(\sum_i \alpha_i y_i K(x, x_i) + w_0\right).$$

Для предсказания необходимо оценить значение w_0. Известно, что для любого опорного объекта, который классифицируется безошибочно, верно следующее:

$$y_i = \sum_j \alpha_j y_j K(x_i, x_j) + w_0,$$

значит для любого такого объекта:

$$w_0 = y_i - \sum_j \alpha_j y_j K(x_i, x_j)$$

В случае наличия ошибок классификации обучающей выборки предлагается усреднять значение w_0 по всем опорным векторам:

$$w_0 = \frac{1}{N_{\text{SV}}} \sum_{n \in \text{SV}} \left(y_i - \sum_j \alpha_j y_j K(x_i, x_j) \right).$$

Интуиция здесь такова, что суммарные ошибки в положительную сторону примерно равны суммарным ошибкам в отрицательную сторону.

Структура класса аналогична:

```
class KernelSVM:
    def __init__(self, C, kernel=None, sigma=1.0, degree=2):
        self.C = C
        ...

    def fit(self, X, y):
        ...

    def decision_function(self, X):
        ...

    def predict(self, X):
        ...
```

Параметр конструктора `degree` понадобится при использовании полиномиального ядра, игнорируется другими ядрами. Параметр `sigma` используется для гауссовского ядра. Параметры конструктора соответствуют параметрам стандартного SVM в Scikit-learn [2].

3 Реализуйте полиномиальное и гауссовское ядра. Функция должна принимать на вход две матрицы объектов X и возвращать матрицу K.

Например, линейное ядро выглядит следующим образом:

```
import numpy as np

def linear_kernel(X1, X2):
    return np.dot(X1, X2)
```

4 Воспользуйтесь визуализатором разделяющей поверхности, приведенным по ссылке[3] или напишите свой с соответствующей сигнатурой.

```
def visualize(clf, X, y):
  ...
```

5 Протестируйте следующие алгоритмы на случайных выборках с признаками длины 2 и визуализируйте на плоскости получающиеся решающие правила.

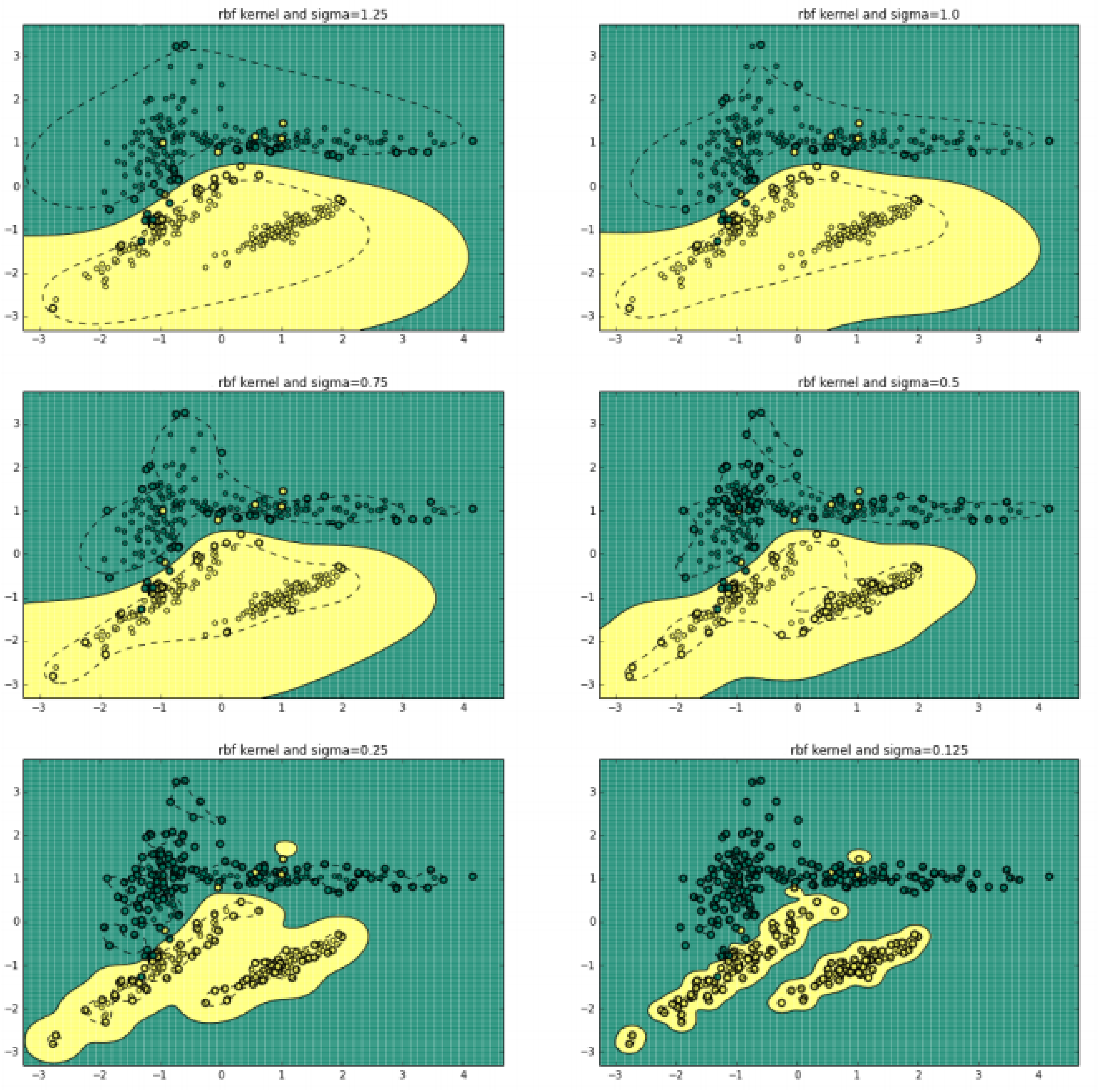
1. Линейная разделяющая гиперплоскость обученная линейным SVM и обученная ядровым SVM с линейным ядром
2. Квадратичная разделяющая гиперплоскость (ядровой SVM с полиномиальным ядром степени 2)
3. Radial basis function SVM (ядровой SVM с гауссовским ядром)

[2] http://scikit-learn.org/stable/modules/svm.html
[3] https://gist.github.com/ktisha/ae995b7b553db26316f9

Для генерации случайных наборов данных можно воспользоваться SciKit-Learn [4]

6 Поварьируйте параметры классификатора. Что можно сказать про влияние параметров на классификацию?

Пример полученной визуализации для RBF ядра с разными значениями



[4]`http://scikit-learn.org/stable/datasets/#sample-generators`