Постановка задачи

Реализовать алгоритм SVM для решения задачи бинарной классификации.

Теоретическая справка

Вначале рассмотрим некоторую **линейно-разделимую** выборку из двух классов. Выборка называется линейно разделимой, если в пространстве признаков существует такая гиперплоскость, что объекты разных классов будут находиться по разные стороны от этой плоскости.

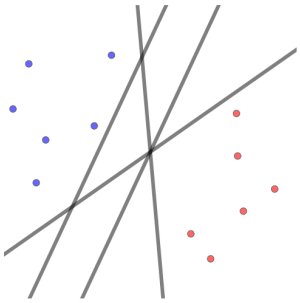

Рис. 1: Разделяющая гиперплоскость не единственна для линейно-разделимой выборки

Возникает задача отыскания **оптимальной разделяющей гиперплоскости**. Пусть разделяющая гиперплоскость существует и задается уравнением $\langle w, x\rangle - w_0 = 0$. Можно выбрать две параллельные ей и расположенные по разные стороны от нее гиперплоскости так, чтобы между ними не было объектов выборки, а расстояние между ними было максимальным. В таком случае каждая из двух получившихся граничных плоскостей будет «приставлена» к соотвествующему классу.

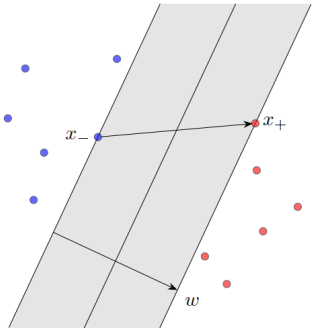

Рис. 2: Разделяющая полоса

Поскольку уравнение плоскости можно умножать на ненулевое число без изменения соответствующей плоскости, всегда можно выбрать (отнормировать) w и w_0 таким образом, чтобы уравнения граничных плоскостей имели вид:

$$\langle w, x\rangle - w_0 = \pm 1.$$

Это условие нормировки можно также сформулировать следующим образом:

$$\min_{i=1,\dots,l} y_i(\langle w, x\rangle - w_0) = 1.$$

На каждой из двух граничных плоскостей будет лежать как минимиум один объект из соответствующего ей класса (иначе расстояние между плоскостями можно увеличить). Пусть x_+ и x_- – два таких лежащие

на построенных плоскостях и принадлежащие соответствующим классам. Тогда для ширины разделяющей полосы будет справедливо выражение (как это следует из аналитической геометрии):

$$\left\langle (x_+ - x_-), \frac{w}{||w||} \right\rangle = \frac{2}{||w||}$$

Теперь можно поставить задачу построения такой разделяющей гиперплоскости, что расстояние между соответствующими ей граничными плоскостями будет максимальным:

$$\begin{cases} \frac{1}{2}\langle w, w\rangle \to \min\limits_{w, w_0}, \\ y_i(\langle w, x\rangle - w_0) \geq 1, \quad i = 1, \ldots, l. \end{cases}$$

В случае **линейно-неразделимой выборки** по определению любой линейный классификатор будет ошибаться, т.е. условие $y_i(\langle w, x\rangle - w_0) \geq 1$ не может быть выполнено для $\forall i$. Естественным обобщением задачи построения оптимальной гиперплоскости на случай линейно неразделимой выборки является введение ошибок (**slack variables**) $\xi_i \geq 0$ алгоритма и штрафов (C) за эти ошибки в минимизируемую функцию следующим образом:

$$\begin{cases} \frac{1}{2}\langle w, w\rangle + C\sum\limits_{i=1}^{l} \xi_i \to \min\limits_{w, w_0, \xi}, \\ y_i(\langle w, x\rangle - w_0) \geq 1 - \xi_i, \quad i = 1, \ldots, l, \\ \xi_i \geq 0, \quad i = 1, \ldots, l. \end{cases}$$

Получившаяся оптимизационная задача является оптимизационной задачей в **методе опорных векторов (SVM)** и непосредственно связана с задачей линейной классификации. Поскольку $M_i = y_i(\langle w, x\rangle - w_0)$ – отступ на i-ом объекте выборки:

$$y_i(\langle w, x\rangle - w_0) \geq 1 - \xi_i \Rightarrow \xi_i \geq 1 - M_i.$$

Учитывая также условие $\xi_i \geq 0$, можно получить:

$$\begin{cases} \frac{1}{2}\langle w, w\rangle + C\sum\limits_{i=1}^{l} \xi_i \to \min\limits_{w, w_0, \xi}, \\ \xi_i \geq \max\{0, 1 - M_i\}, \quad i = 1, \ldots, l. \end{cases}$$

При фиксированных w и w_0 задача оптимизации по ξ имеет следующий вид:

$$\sum_{i=1}^{l} \xi_i \to \min_{\xi}, \text{ при условии } \xi_i \geq \max\{0, 1 - M_i\}, \quad i = 1, \ldots, l.$$

Ее решением будет $\xi_i = \max\{0, 1 - M_i\} = (1 - M_i)_+$.

Теперь можно вернуться к общей задаче минимизации и переписать ее как задачу безусловной оптимизации:

$$Q(w, w_0) = \sum_{i=1}^{l} (1 - M_i(w, w_0))_+ + \frac{1}{2C}||w||^2 \to \min_{w, w_0}.$$

В такой формулировке отчетливо видно и кусочно-линейную функцию потерь (**hinge loss**), и L_2-регуляризатор. Это задача квадратичного программирования, которая решается с использованием условия Каруша-Куна-Таккера с поиском седловой точки функции Лагранжа. В итоге, $w = \sum\limits_{i=1}^{l} \lambda_i y_i x_i$. И если для какого-либо i-го наблюдения $\lambda_i = 0$, значит, он не используется для вычисления вектора w. Нулевых значений λ получается достаточно много и из всей обучающей выборки остается несколько, которые и играю роль при расчете коэффициентов. Такие наблюдения (векторы) получили название **опорных**. Отсюда и пошло название **метод опорных векторов**.

О датасете

Выбранный датасет представляет собой реальные данные о классификации зерен трех разных сортов пшеницы на основе их геометрических свойств [1]. У каждого случайно выбранного для эксперимента представителя выборки выделены следующие 7 признаков:

- **Площадь** A
- **Периметр** P
- $C = \frac{4\pi A}{P^2}$
- **Длина зерна**
- **Ширина зерна**
- **Коэффициэнт ассиметрии**
- **Длина желобка**

Целевым признаком являются **сорта** пшеницы : “Kama”, “Rosa”, “Canadian”. Поскольку мы рассматриваем задачу бинарной классификации, то рассмотрим только сорта “Kama” (метка 1) и “Canadian” (метка -1).

В обучающей выборке данные о 100 зернах (по 50 на каждый сорт). В тестовом датасете данные о 40 объектах (в каждом классе по 20).

Использование алгоритма

Для решения вышеописанной задачи квадратичного программирования и реализации SVM используется библиотека “CVXPY” [2].

В самом начале ищем оптимальный параметр регуляризации $\lambda = \frac{1}{2C}$:

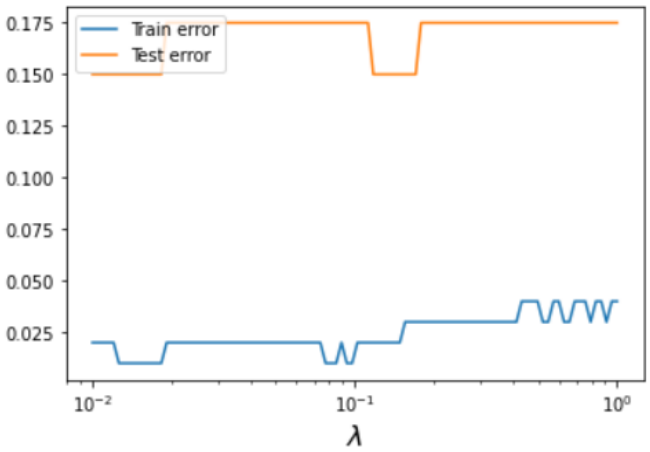

Рис. 3: Ошибки на обучающей и тестовой выборке при выборе различных параметров λ

При $\lambda = 10^{-2}$ достигаются наименьшие потери на тестовой выборке. При этом точность на тесте $acc = 0.85$.

Проверим устойчивость алгоритма: удалим по пять записей каждого класса из обучающего множества и проверим работу (качество классификации) на той же тестовой выборке:

	Train size	Test size	Accuracy	Right	Wrong
1	90	40	0.85	34	6
2	80		0.85	34	6
3	70		0.85	34	6

Таблица 1: Проверка устойчивости решения при $\lambda = 10^{-2}$

Видим, что SVM достаточно устойчив.

Заключение

В работе рассмотрен и реализован алгоритм SVM. Проведен анализ датасета, содержащего информацию о геометрических характеристиках зерен пшеницы двух сортов, на нем применен реализованный алгоритм. SVM показал высокую точность на тесте. Также SVM оказался устойчивым при удалении части данных из обучающей выборки.

Список литературы

[1] Charytanowicz,Magorzata, Niewczas,Jerzy, Kulczycki,Piotr, Kowalski,Piotr, and Lukasik,Szymon. (2012). seeds. UCI Machine Learning Repository. DOI

[2] CVXPY

Листинг программы

```
import numpy as np

# Загрузим наш датасет
import pandas as pd
target_index = {'Kama': 1, 'Canadian': -1}
df = pd.read_csv('train-1.csv')
df['Target'] = df['Target'].apply(lambda x: target_index[x])
train = df.to_numpy()
X, y = train[:, :-1], train[:, -1].reshape(-1, 1)
test_df = pd.read_csv('test.csv')
test_df['Target'] = test_df['Target'].apply(lambda x: target_index[x])
test = test_df.to_numpy()
X_test, y_test = test[:, :-1], test[:, -1].reshape(-1, 1)

l = len(X)
n = X.shape[1]
l_test = len(X_test)

# Сформулируем задачу оптимизации с помощью CVXPY
import cvxpy as cp
w = cp.Variable((n,1))
w0 = cp.Variable()
loss = cp.sum(cp.pos(1 - cp.multiply(y, X*w - w0)))
reg = cp.norm(w, 2)
lambd = cp.Parameter(nonneg=True)
prob = cp.Problem(cp.Minimize(loss/l + lambd*reg))

# Решаем оптимизационную задачу для различных  lambda из диапазона.
# Наиболее правильным будет выбор  lambda   при котором достигается наименьшие потери на тестовой выборке.
y = y.reshape(1, -1)
y_test = y_test.reshape(1, -1)
TRIALS = 100
train_error = np.zeros(TRIALS)
test_error = np.zeros(TRIALS)
lambda_vals = np.logspace(-2, 0, TRIALS)
for i in range(TRIALS):
    lambd.value = lambda_vals[i]
    prob.solve()
    train_error[i] = (y != np.sign(X.dot(w.value) - w0.value).reshape(1, -1)).sum()/l
    test_error[i] = (y_test != np.sign(X_test.dot(w.value) - w0.value).reshape(1, -1)).sum()/l_test


import matplotlib.pyplot as plt
plt.plot(lambda_vals, train_error, label="Train error")
plt.plot(lambda_vals, test_error, label="Test error")
plt.xscale('log')
plt.legend(loc='upper left')
plt.xlabel(r"$\lambda$", fontsize=16)
plt.show()


opt_lambd = lambda_vals[np.argmin(test_error)]
print(f'Optimal lambda is {opt_lambd}')

lambd.value = opt_lambd
#lambd.value = 0.01
prob.solve()


def predict(X, w, w0):
    return np.sign(np.dot(X, w) - w0).reshape(1, -1)

def accuracy(y_true, y_pred):
    return np.mean(y_pred == y_true)


y_pred = predict(X_test, w.value, w0.value)
print(f"Accuracy : {accuracy(y_test, y_pred)}")
```