Математический анализ данных и машинное обучение

Лекция 3

Саркисян Вероника

План на сегодня

9:30 - 10:45	SVM, многоклассовая классификация.
11:00 - 12:30	Семинар
12:30 - 13:30	Обед
13:30 - 14:30	Регуляризация, LASSO и Ридж-регрессия.
14:45 - 17:30	Снижение размерности признакового пространства: метод главных компонент.

Метод опорных векторов: разделимый случай

Будем рассматривать классификаторы вида:

$$a(x) = \text{sign}(\langle w, x \rangle + b), \qquad w \in \mathbb{R}^d, b \in \mathbb{R}.$$

Расстояние от объекта до разделяющей гиперплоскости:

$$\rho(x_0, a) = \frac{|\langle w, x \rangle + b|}{\|w\|}.$$

Расстояние от гиперплоскости до ближайшего объекта выборки:

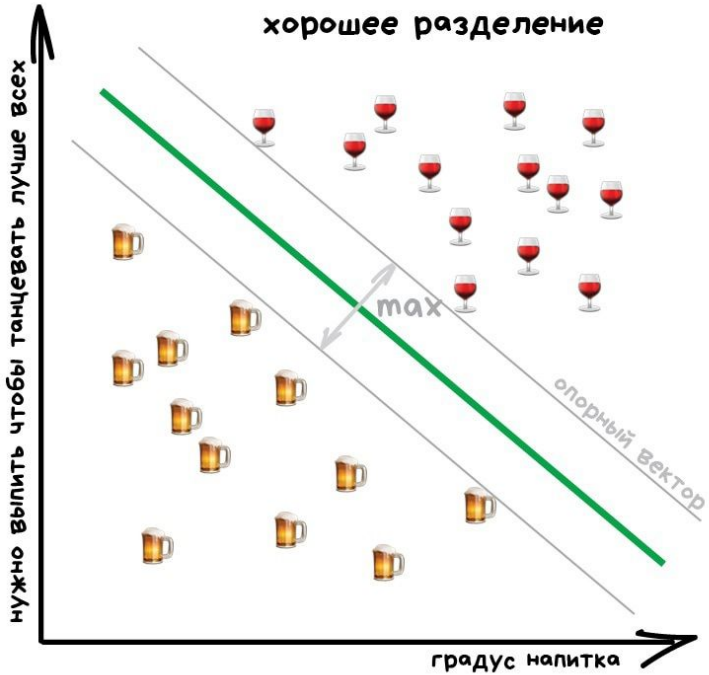
$$\min_{x \in X^\ell} \frac{|\langle w, x \rangle + b|}{\|w\|} = \frac{1}{\|w\|} \min_{x \in X} |\langle w, x \rangle + b| = \frac{1}{\|w\|}.$$

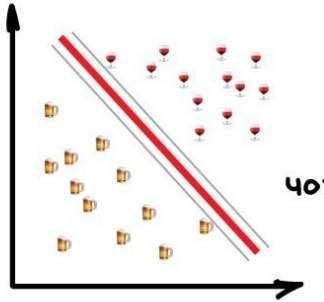
(здесь воспользовались тем, что можно одновременно умножать w и b на положительную константу)

Оптимизационная задача:

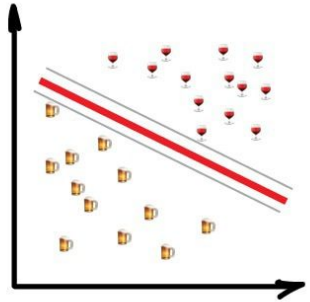
$$\begin{cases} \frac{1}{2}\|w\|^2 \to \min\limits_{w,b} \\ y_i\left(\langle w, x_i \rangle + b\right) \geqslant 1, \quad i = 1, \ldots, \ell. \end{cases}$$

Разделяем виды алкоголя





чот не очень =/



ты пьян,
иди домой

Метод Опорных Векторов

Метод опорных векторов: неразделимый случай

Введем штраф за попадание объектов внутрь разделяющей полосы:

$$y_i\left(\langle w, x_i\rangle + b\right) \geqslant 1 - \xi_i, \quad i = 1, \ldots, \ell.$$

Новая оптимизационная задача: параметр C отвечает за то, как сильно мы штрафуем за попадание внутрь полосы.

$$\begin{cases} \dfrac{1}{2}\|w\|^2 + C\sum\limits_{i=1}^{\ell}\xi_i \to \min\limits_{w,b,\xi} \\ y_i\left(\langle w, x_i\rangle + b\right) \geqslant 1 - \xi_i, \quad i = 1, \ldots, \ell, \\ \xi_i \geqslant 0, \quad i = 1, \ldots, \ell. \end{cases}$$

sklearn.svm.SVC

class `sklearn.svm.` **SVC** (*C=1.0*, *kernel='rbf'*, *degree=3*, *gamma='auto_deprecated'*, *coef0=0.0*, *shrinking=True*, *probability=False*, *tol=0.001*, *cache_size=200*, *class_weight=None*, *verbose=False*, *max_iter=-1*, *decision_function_shape='ovr'*, *random_state=None*) [source]

Parameters:

C : *float, optional (default=1.0)*

Penalty parameter C of the error term.

kernel : *string, optional (default='rbf')*

Specifies the kernel type to be used in the algorithm. It must be one of 'linear', 'poly', 'rbf', 'sigmoid', 'precomputed' or a callable. If none is given, 'rbf' will be used. If a callable is given it is used to pre-compute the kernel matrix from data matrices; that matrix should be an array of shape `(n_samples, n_samples)`.

degree : *int, optional (default=3)*

Degree of the polynomial kernel function ('poly'). Ignored by all other kernels.

tol : ***float, optional (default=1e-3)***

Tolerance for stopping criterion.

cache_size : ***float, optional***

Specify the size of the kernel cache (in MB).

class_weight : ***{dict, 'balanced'}, optional***

Set the parameter C of class i to class_weight[i]*C for SVC. If not given, all classes are supposed to have weight one. The "balanced" mode uses the values of y to automatically adjust weights inversely proportional to class frequencies in the input data as

```
n_samples / (n_classes * np.bincount(y))
```

verbose : ***bool, default: False***

Enable verbose output. Note that this setting takes advantage of a per-process runtime setting in libsvm that, if enabled, may not work properly in a multithreaded context.

max_iter : ***int, optional (default=-1)***

Hard limit on iterations within solver, or -1 for no limit.

decision_function_shape : ***'ovo', 'ovr', default='ovr'***

Whether to return a one-vs-rest ('ovr') decision function of shape (n_samples, n_classes) as all other classifiers, or the original one-vs-one ('ovo') decision function of libsvm which has shape (n_samples, n_classes * (n_classes - 1) / 2). However, one-vs-one ('ovo') is always used as multi-class strategy.

Многоклассовая классификация: One-VS-All

Обучим K (K = число классов) линейных классификаторов: $b_1(x), \ldots, b_K(x)$

Каждый классификатор (бинарный!) будет отличать один класс от всех остальных. $b_k(x) = \langle w_k, x \rangle + w_{0k}.$

Итоговый класс будем вычислять как наиболее вероятный, исходя из прогнозов всех алгоритмов:

$$a(x) = \underset{k \in \{1,\ldots,K\}}{\arg\max}\, b_k(x).$$

Многоклассовая классификация: All-VS-All

Обучим C_K^2 классификаторов (для всех возможных пар классов):

$$b_k(x) = \text{sign}(\langle w_k, x \rangle + w_{0k}).$$

Каждый классификатор (бинарный!) обучаем на подвыборке, содержащей только 2 класса.

Для классификации нового объекта подадим его на вход всем построенным классификаторам; в качестве ответа выберем наиболее “частый” среди ответов класс.

$$a(x) = \arg\max_{k \in \{1,\dots,K\}} \sum_{i=1}^{K} \sum_{j \neq i} [a_{ij}(x) = k]$$

Метрики качества

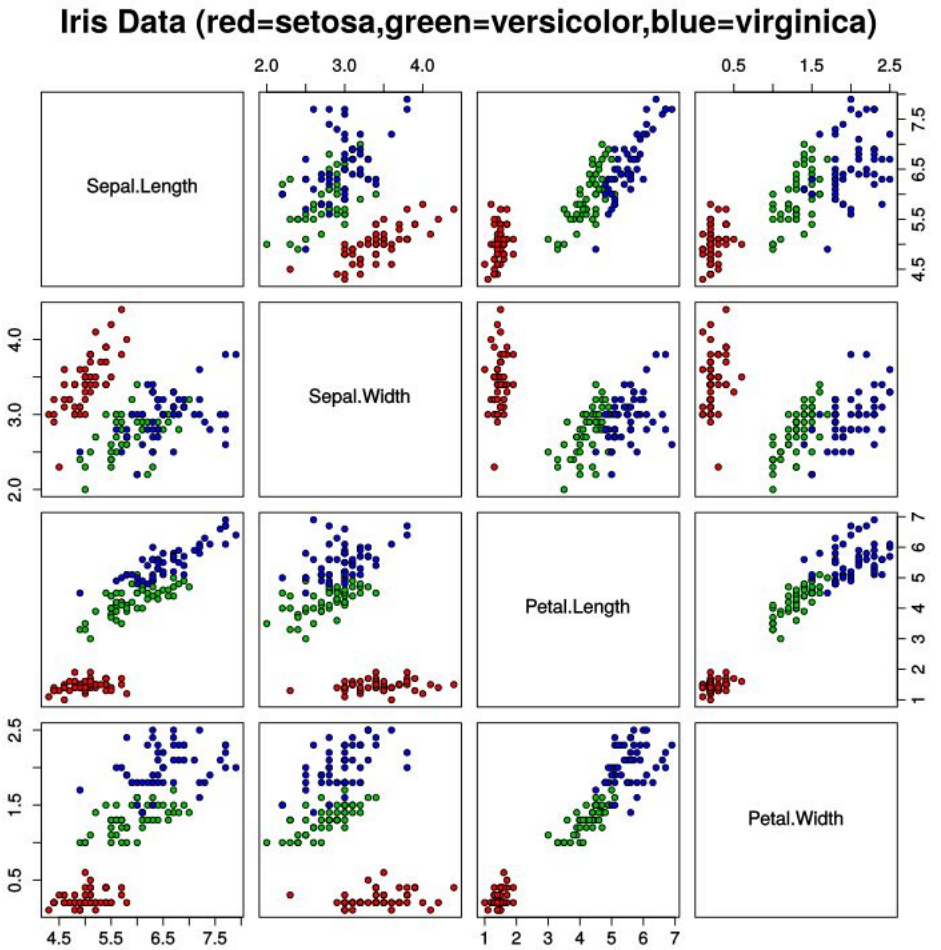
Рассмотрим K двухклассовых задач (one-VS-all),
для каждой вычислим матрицу ошибок: $\mathrm{TP}_k, \mathrm{FP}_k, \mathrm{FN}_k, \mathrm{TN}_k$

Микро-усреднение: $\mathrm{precision}(a, X) = \dfrac{\overline{\mathrm{TP}}}{\overline{\mathrm{TP}} + \overline{\mathrm{FP}}}, \qquad \overline{\mathrm{TP}} = \dfrac{1}{K}\sum_{k=1}^{K} \mathrm{TP}_k$

Макро-усреднение:

$$\mathrm{precision}(a, X) = \frac{1}{K}\sum_{k=1}^{K} \mathrm{precision}_k(a, X); \qquad \mathrm{precision}_k(a, X) = \frac{\mathrm{TP}_k}{\mathrm{TP}_k + \mathrm{FP}_k}.$$

Датасет Iris



Регуляризация

Вернемся к задаче линейной регрессии:

$$a(x) = w_0 + \langle w, x \rangle$$

Будем штрафовать за “сложность” модели:

$$Q_\alpha(w) = Q(w) + \alpha R(w)$$

L2 - регуляризатор:

$$R(w) = \|w\|_2 = \sum_{i=1}^{d} w_i^2,$$

L1 - регуляризатор:

$$R(w) = \|w\|_1 = \sum_{i=1}^{d} |w_i|.$$

Ридж-регрессия (L2-регуляризация)

sklearn.linear_model.Ridge

class `sklearn.linear_model.` **`Ridge`** (*alpha=1.0*, *fit_intercept=True*, *normalize=False*, *copy_X=True*, *max_iter=None*, *tol=0.001*, *solver='auto'*, *random_state=None*) [source]

LASSO (L1-регуляризация)

sklearn.linear_model.Lasso

class sklearn.linear_model. **Lasso** (*alpha=1.0, fit_intercept=True, normalize=False, precompute=False, copy_X=True, max_iter=1000, tol=0.0001, warm_start=False, positive=False, random_state=None, selection='cyclic'*)

[source]

Метод главных компонент (PCA)

Пусть $X \in \mathbb{R}^{\ell \times D}$ — матрица «объекты-признаки», где ℓ — число объектов, а D — число признаков. Поставим задачу уменьшить размерность пространства до d. Будем считать, что данные являются центрированными — то есть среднее в каждом столбце матрицы X равно нулю.

Будем искать главные компоненты $u_1, \ldots, u_D \in \mathbb{R}^D$, которые удовлетворяют следующим требованиям:

1. Они ортогональны: $\langle u_i, u_j \rangle = 0$, $i \neq j$;
2. Они нормированы: $\|u_i\|^2 = 1$;
3. При проецировании выборки на компоненты $u_1, \ldots, u_d$ получается максимальная дисперсия среди всех возможных способов выбрать d компонент.

Чтобы понизить размерность выборки до d, мы будем проецировать её на первые d компонент — из последнего свойства следует, что это оптимальный способ снижения размерности.

Дисперсия проецированной выборки показывает, как много информации нам удалось сохранить после понижения размерности — и поэтому мы требуем максимальной дисперсии от проекций.

sklearn.decomposition.PCA

class sklearn.decomposition. **PCA** (*n_components=None*, *copy=True*, *whiten=False*, *svd_solver='auto'*, *tol=0.0*, *iterated_power='auto'*, *random_state=None*) [source]