SVM, ядерные методы и RVM

Сергей Николенко

СПбГУ — Санкт-Петербург

29 февраля 2024 г.

Random facts:

- 29 февраля в Шотландии с 1288 г. был объявлен днём, когда женщина может предложить брак мужчине; нужно было, чтобы «всякая дама, которая идёт свататься, надевала нижнюю сорочку из багряной фланели и чтобы кромка её была хорошо видна, иначе мужчине придётся заплатить за неё штраф» в 1 фунт
- 29 февраля 1504 г. Христофор Колумб сумел обмануть аборигенов Ямайки, отказывавшихся поставлять европейцам еду; Колумб знал, что в этот день будет лунное затмение, и объявил, что за такое поведение испанский бог лишит индейцев Луны; в итоге индейцы возобновили поставки, а Колумб вернул им Луну
- за 29 февраля 1712 г. в Швеции последовало 30 февраля; чтобы вернуться к юлианскому календарю после пропуска високосных дней (собирались постепенно перейти на григорианский, начали в 1700, но потом началась Северная война и стало не до того), Швеция в 1712 г. добавила к февралю два лишних дня вместо одного
- 29 февраля 1880 г. было закончено строительство тоннеля Готард через Альпы
- 29 февраля 1888 г. Герман Холлерит для ускорения обработки данных переписи населения изобрёл первую электрическую счётную машину — табулятор, работавший на перфокартах; с 1924 г. фирма Холлерита стала называться IBM

SVM и задача линейной классификации

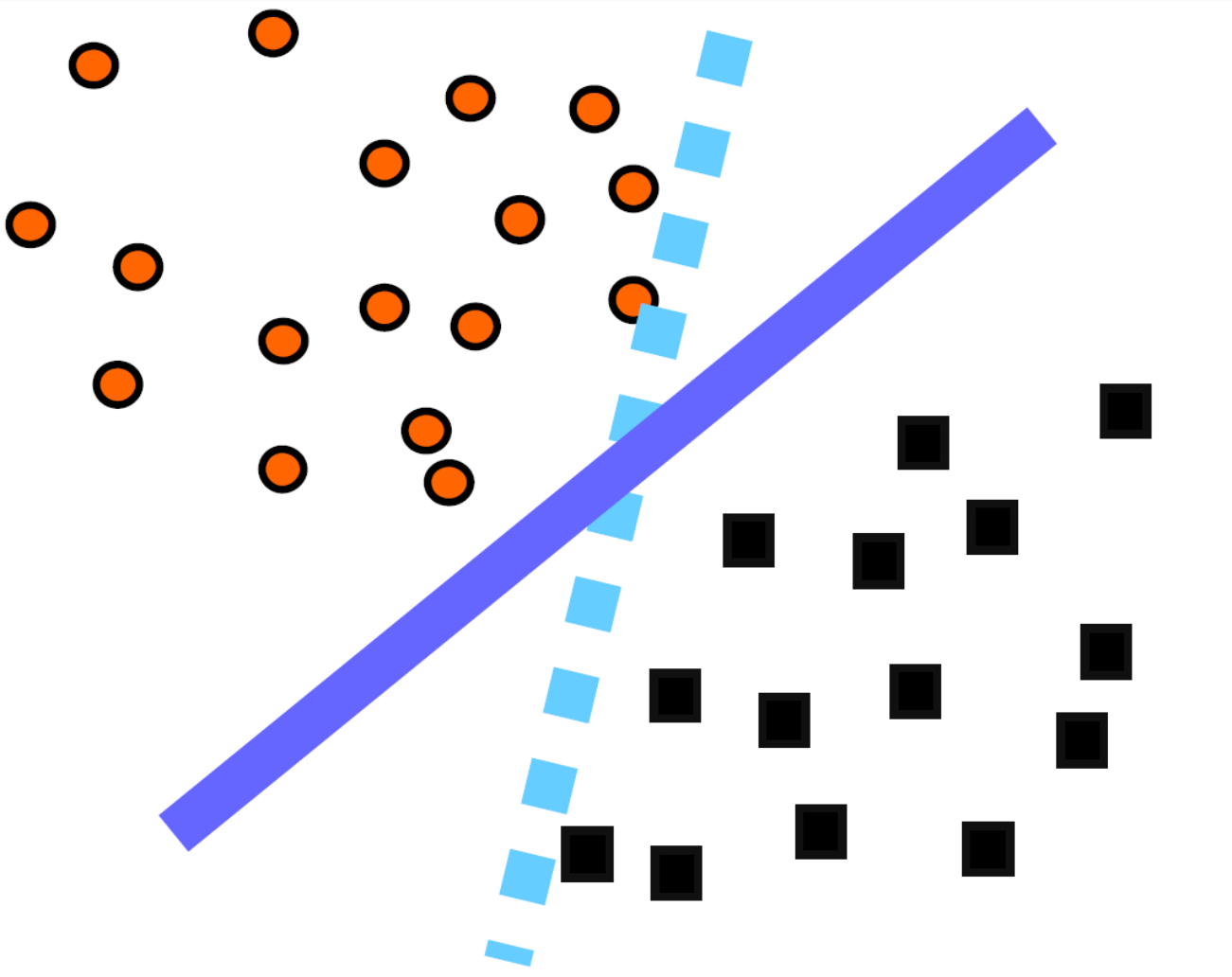
Постановка задачи

- Метод опорных векторов решает задачу классификации.
- Каждый элемент данных — точка в n–мерном пространстве $\mathbb{R}^n$.
- Формально: есть точки x_i, $i = 1..m$, у точек есть метки $y_i = \pm 1$.
- Мы интересуемся: можно ли разделить данные $(n-1)$–мерной гиперплоскостью, а также хотим найти эту гиперплоскость.
- Это всё?

Постановка задачи

- Нет, ещё хочется научиться разделять этой гиперплоскостью *как можно лучше*.
- То есть желательно, чтобы два разделённых класса лежали как можно дальше от гиперплоскости.
- Практическое соображение: тогда от небольших возмущений в гиперплоскости ничего не испортится.

Пример



Выпуклые оболочки

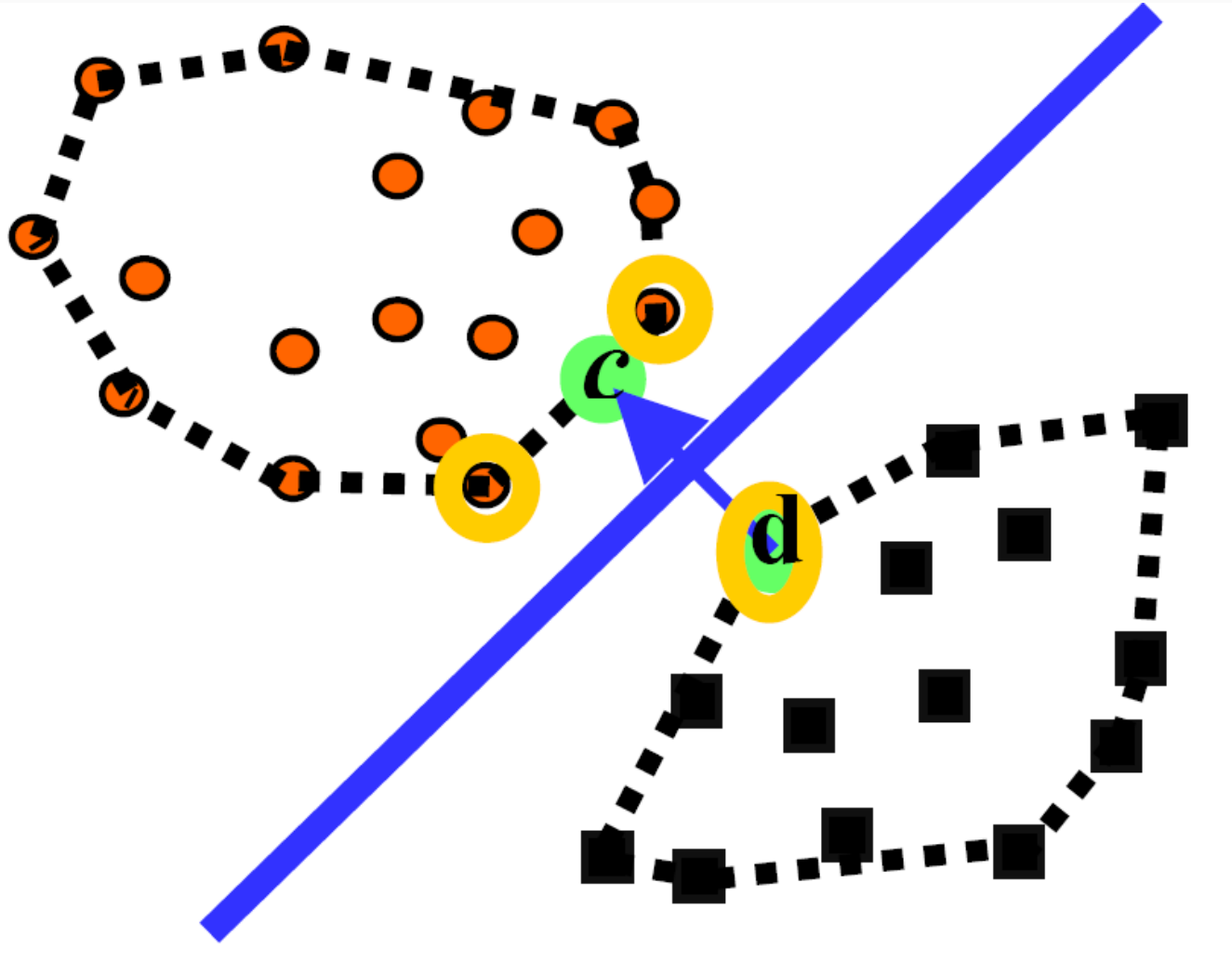
- Один подход: найти две ближайшие точки в выпуклых оболочках данных, а затем провести разделяющую гиперплоскость через середину отрезка.
- Формально это превращается в задачу квадратичной оптимизации:

$$\min_{\alpha} \left\{ ||c-d||^2, \text{ где } c = \sum_{y_i=1} \alpha_i x_i, d = \sum_{y_i=-1} \alpha_i x_i \right\}$$

$$\text{при условии } \sum_{y_i=1} \alpha_i = \sum_{y_i=-1} \alpha_i = 1, \alpha_i \geq 0.$$

- Эту задачу можно решать общими оптимизационными алгоритмами.

Пример



Максимизация зазора

- Другой подход: максимизировать *зазор* (margin) между двумя параллельными опорными плоскостями, затем провести им параллельную на равных расстояниях от них.
- Гиперплоскость называется *опорной* для множества точек X, если все точки из X лежат под одну сторону от этой гиперплоскости.
- Формально: расстояние от точки до гиперплоскости $y(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + w_0 = 0$ равно $\frac{|y(\mathbf{x})|}{\|\mathbf{w}\|}$.

Максимизация зазора

- Расстояние от точки до гиперплоскости $y(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + w_0 = 0$ равно $\frac{|y(\mathbf{x})|}{\|\mathbf{w}\|}$.
- Все точки классифицированы правильно: $t_n y(\mathbf{x}_n) > 0$ ($t_n \in \{-1, 1\}$).
- И мы хотим найти

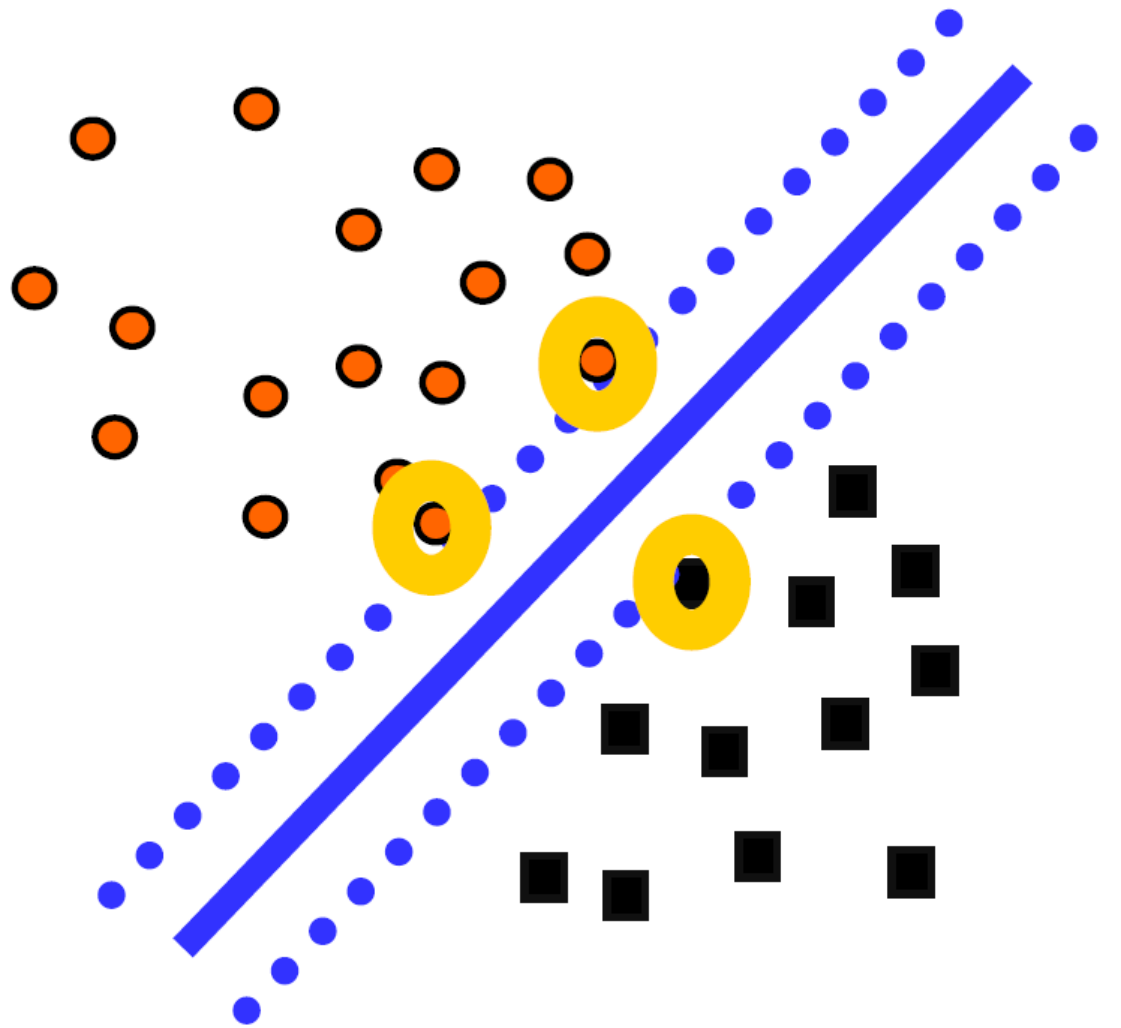
$$\arg\max_{\mathbf{w}, w_0} \min_n \frac{t_n y(\mathbf{x}_n)}{\|\mathbf{w}\|} =$$

$$= \arg\max_{\mathbf{w}, w_0} \left\{ \frac{1}{\|\mathbf{w}\|} \min_n \left[t_n (\mathbf{w}^\top \mathbf{x}_n + w_0) \right] \right\}.$$

Максимизация зазора

- $\arg\max_{\mathbf{w},w_0} \left\{ \frac{1}{\|\mathbf{w}\|} \min_n \left[t_n(\mathbf{w}^\top \mathbf{x}_n + w_0) \right] \right\}$. Сложно.
- Но если перенормировать $\mathbf{w}$, гиперплоскость не изменится.
- Давайте перенормируем так, чтобы $\min_n \left[t_n(\mathbf{w}^\top \mathbf{x}_n + w_0) \right] = 1$.

Пример



Максимизация зазора

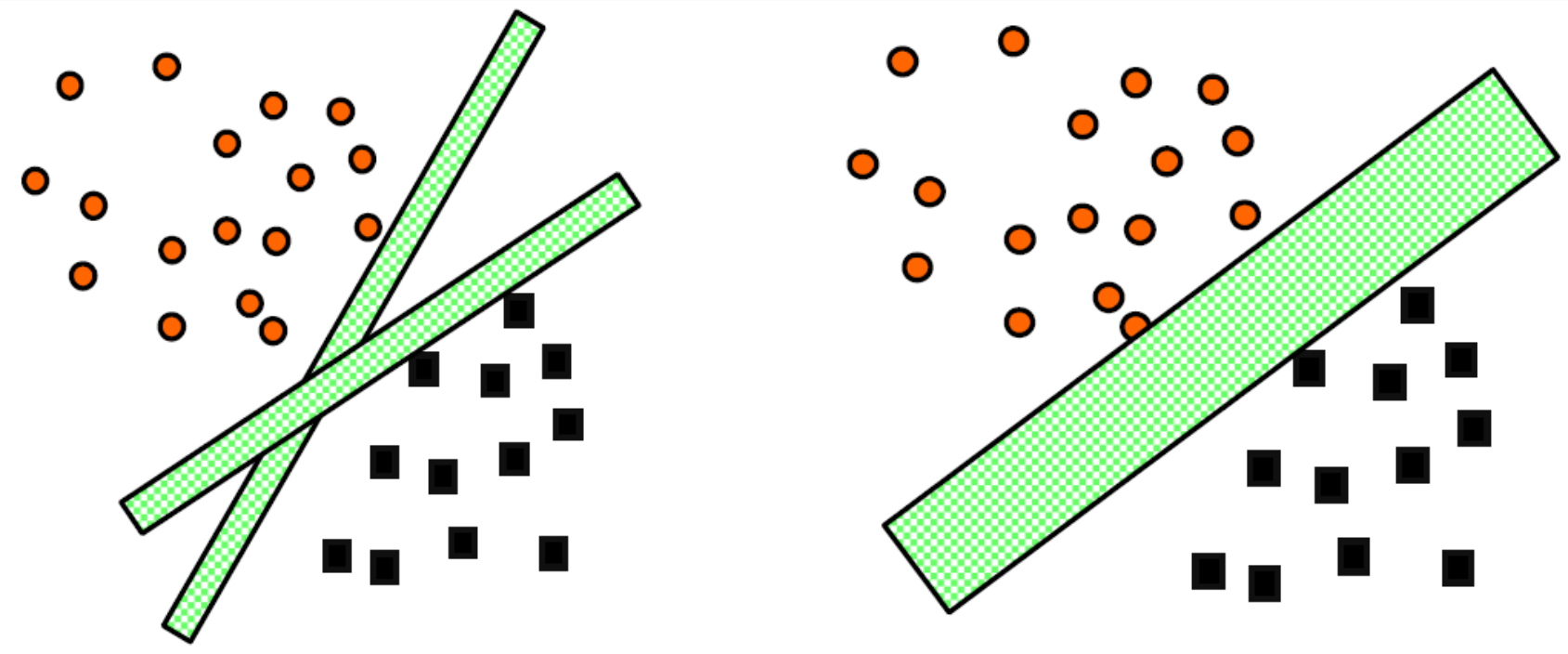
- Получается тоже задача квадратичного программирования:

$$\min_{\vec{w},b}\left\{\frac{1}{2}||\mathbf{w}||^2\right\} \text{ при условии } t_n(\mathbf{w}^\top \mathbf{x}_n + w_0) \geq 1.$$

Результаты

- Результаты получаются хорошие. Такой подход позволяет находить *устойчивые* решения, что во многом решает проблемы с оверфиттингом и позволяет лучше предсказывать дальнейшую классификацию.
- В каком-то смысле в решениях с «толстыми» гиперплоскостями между данными содержится больше информации, чем в «тонких», потому что «толстых» меньше.
- Это всё можно сформулировать и доказать (позже).

Пример



Дуальные задачи

- Напомним, что такое дуальные задачи.
- Прямая задача оптимизации:

$$\min\{f(x)\} \text{ при условии } h(x) = 0,\ g(x) \leq 0,\ x \in X.$$

- Для дуальной задачи вводим параметры λ, соответствующие равенствам, и μ, соответствующие неравенствам.

Дуальные задачи

- Прямая задача оптимизации:

$$\min\{f(x)\} \text{ при условии } h(x) = 0,\ g(x) \leq 0,\ x \in X.$$

- Дуальная задача оптимизации:

$$\min\{\phi(\lambda, \mu)\} \text{ при условии } \mu \geq 0,$$
$$\text{где } \phi(\lambda, \mu) = \inf_{x \in X} \left\{ f(x) + \lambda^\top h(x) + \mu^\top g(x) \right\}.$$

Дуальные задачи

- Тогда, если $(\bar{\lambda}, \bar{\mu})$ – допустимое решение дуальной задачи, а $\bar{x}$ – допустимое решение прямой, то

$$\phi(\bar{\lambda}, \bar{\mu}) = \inf_{x \in X} \left\{ f(x) + \bar{\lambda}^\top h(x) + \bar{\mu}^\top g(x) \right\} \leq$$
$$\leq f(\bar{x}) + \bar{\lambda}^\top h(\bar{x}) + \bar{\mu}^\top g(\bar{x}) \leq f(\bar{x}).$$

- Это называется *слабой дуальностью* (только $\leq$), но во многих случаях достигается и равенство.

Дуальные задачи

- Для линейного программирования прямая задача:

$$\min c^\top x \text{ при условии } Ax = b,\ x \in X = \{x \leq 0\}.$$

- Тогда дуальная задача получается так:

$$\begin{aligned}\phi(\lambda) = \inf_{x \geq 0} \left\{c^\top x + \lambda^\top (b - Ax)\right\} = \\ = \lambda^\top b + \inf_{x \geq 0} \left\{(c^\top - \lambda^\top A)x\right\} = \\ = \begin{cases} \lambda^\top b, & \text{если } c^\top - \lambda^\top A \geq 0, \\ -\infty & \text{в противном случае.} \end{cases}\end{aligned}$$

Дуальные задачи

- Для линейного программирования прямая задача:

$$\min\left\{c^\top x\right\} \text{ при условии } Ax = b,\ x \in X = \{x \leq 0\}.$$

- Дуальная задача:

$$\max\left\{b^\top \lambda\right\} \text{ при условии } A^\top \lambda \leq c,\ \lambda \text{ не ограничены.}$$

Дуальные задачи

- Для квадратичного программирования прямая задача:

$$\min\left\{\frac{1}{2}x^\top Qx + c^\top x\right\} \text{ при условии } Ax \leq b,$$

где Q – положительно полуопределённая матрица (т.е. $x^\top Qx \geq 0$ всегда).

- Дуальная задача (проверьте):

$$\max\left\{\frac{1}{2}\mu^\top D\mu + \mu^\top d - \frac{1}{2}c^\top Q^{-1}c\right\} \text{ при условии } c \geq 0,$$

где $D = -AQ^{-1}A^\top$ (отрицательно определённая матрица), $d = -b - AQ^{-1}c$.

Дуальная задача к SVM

- В случае SVM надо ввести множители Лагранжа:

$$L(\mathbf{w}, w_0, \alpha) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_n \alpha_n \left[t_n(\mathbf{w}^\top \mathbf{x}_n + w_0) - 1\right], \quad \alpha_n \geq 0.$$

- Берём производные по $\mathbf{w}$ и w_0, приравниваем нулю, получаем

$$\mathbf{w} = \sum_n \alpha_n t_n \mathbf{x}_n,$$
$$0 = \sum_n \alpha_n t_n.$$

Дуальная задача к SVM

- Подставляя в $L(\mathbf{w}, w_0, \alpha)$, получим

$$L(\alpha) = \sum_n \alpha_n - \frac{1}{2} \sum_n \sum_m \alpha_n \alpha_m t_n t_m \left(\mathbf{x}_n^\top \mathbf{x}_m\right)$$

при условии $\alpha_n \geq 0, \sum_n \alpha_n t_n = 0.$

- Это дуальная задача, которая обычно в SVM и используется.

Предсказание и ККТ

- А для предсказания потом надо посмотреть на знак $y(\mathbf{x})$:

$$y(\mathbf{x}) = \sum_{n=1}^{N} \alpha_n t_n \mathbf{x}^\top \mathbf{x}_n + w_0.$$

- Получилось, что предсказания зависят от всех точек $\mathbf{x}_n$...

Предсказание и ККТ

- ...но нет. :) Условия ККТ (Karush–Kuhn–Tucker):

$$\begin{aligned} \alpha_n &\geq 0, \\ t_n y(\mathbf{x}_n) - 1 &\geq 0, \\ \alpha_n \left(t_n y(\mathbf{x}_n) - 1\right) &= 0. \end{aligned}$$

- Т.е. реально предсказание зависит от небольшого числа *опорных* векторов, для которых $t_n y(\mathbf{x}_n) = 1$ (они находятся собственно на границе разделяющей поверхности).

Постановка задачи

- Все эти методы работают, когда данные действительно линейно разделимы.
- А что делать, когда их всё-таки немножко не получается разделить?
- Первый вопрос: что делать для первого метода, метода выпуклых оболочек?

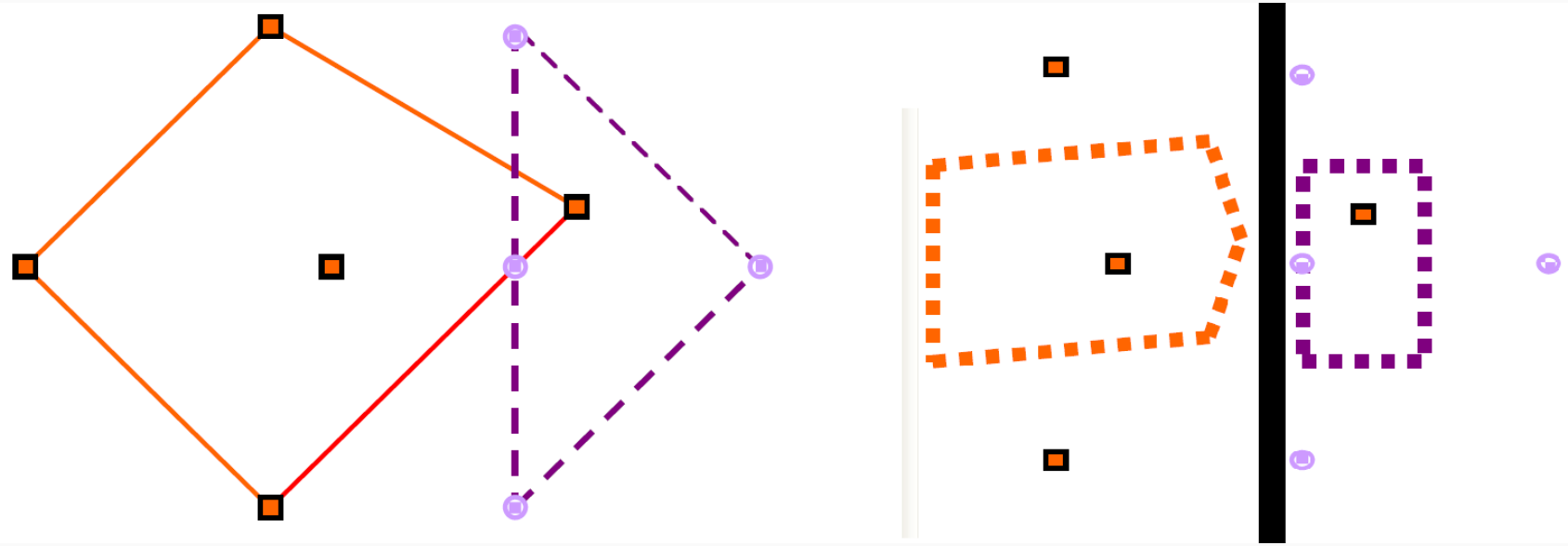
Редуцированные выпуклые оболочки

- Вместо обычных выпуклых оболочек можно рассматривать *редуцированные* (reduced), у которых коэффициенты ограничены не 1, а сильнее:

$$c = \sum_{y_i=1} \alpha_i x_i, \quad 0 \leq \alpha_i \leq D.$$

- Тогда для достаточно малых D редуцированные выпуклые оболочки не будут пересекаться.
- И мы будем искать оптимальную гиперплоскость между редуцированными выпуклыми оболочками.

Пример



Для метода опорных векторов

- Естественно, для метода опорных векторов тоже надо что-то изменить. Что?

Для метода опорных векторов

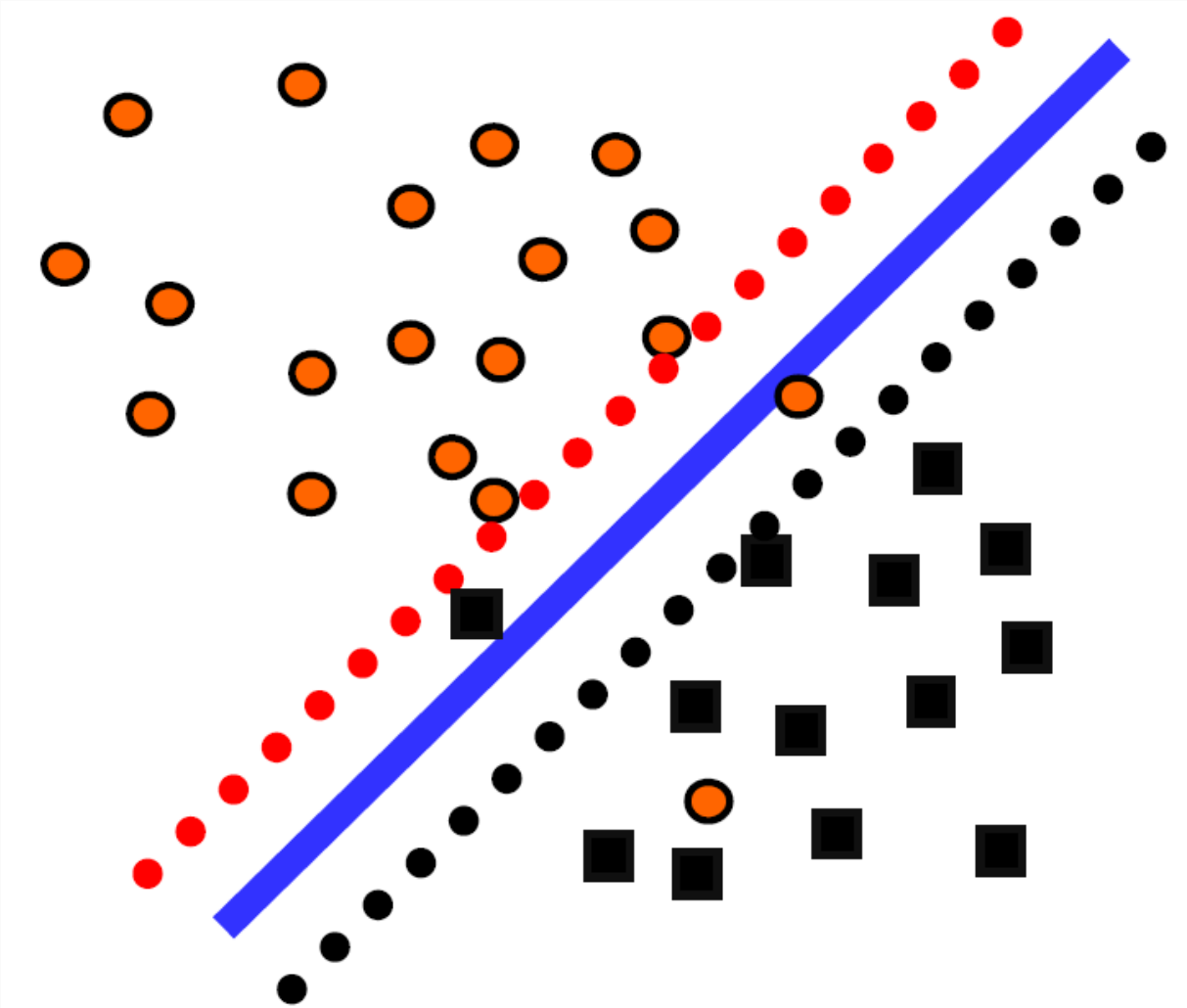
- Естественно, для метода опорных векторов тоже надо что-то изменить. Что?
- Мы просто добавим в оптимизирующуюся функцию неотрицательную ошибку (slack):

$$\min_{\vec{w},w_0}\left\{||\vec{w}||^2 + C\sum_{i=1}^{m} z_i\right\}$$

при условии $t_i(\vec{w}\cdot\vec{x}_i - w_0) + z_i \geq 1.$

- Это прямая задача...

Пример



Дуальная переформулировка

- ...а вот дуальная:

$$\min_{\alpha} \left\{ \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{m} t_i t_j \alpha_i \alpha_j \left(\vec{x}_i \cdot \vec{x}_j \right) - \sum_{i=1}^{m} \alpha_i, \right.$$
$$\left. \text{где } \sum_{i=1}^{m} t_i \alpha_i = 0, \quad 0 \leq \alpha_i \leq C. \right\}$$

- Эта формулировка чаще всего используется в теории SVM.
- Единственное отличие от линейно разделимого случая – верхняя граница C на α_j, т.е. на влияние каждой точки.

Итого

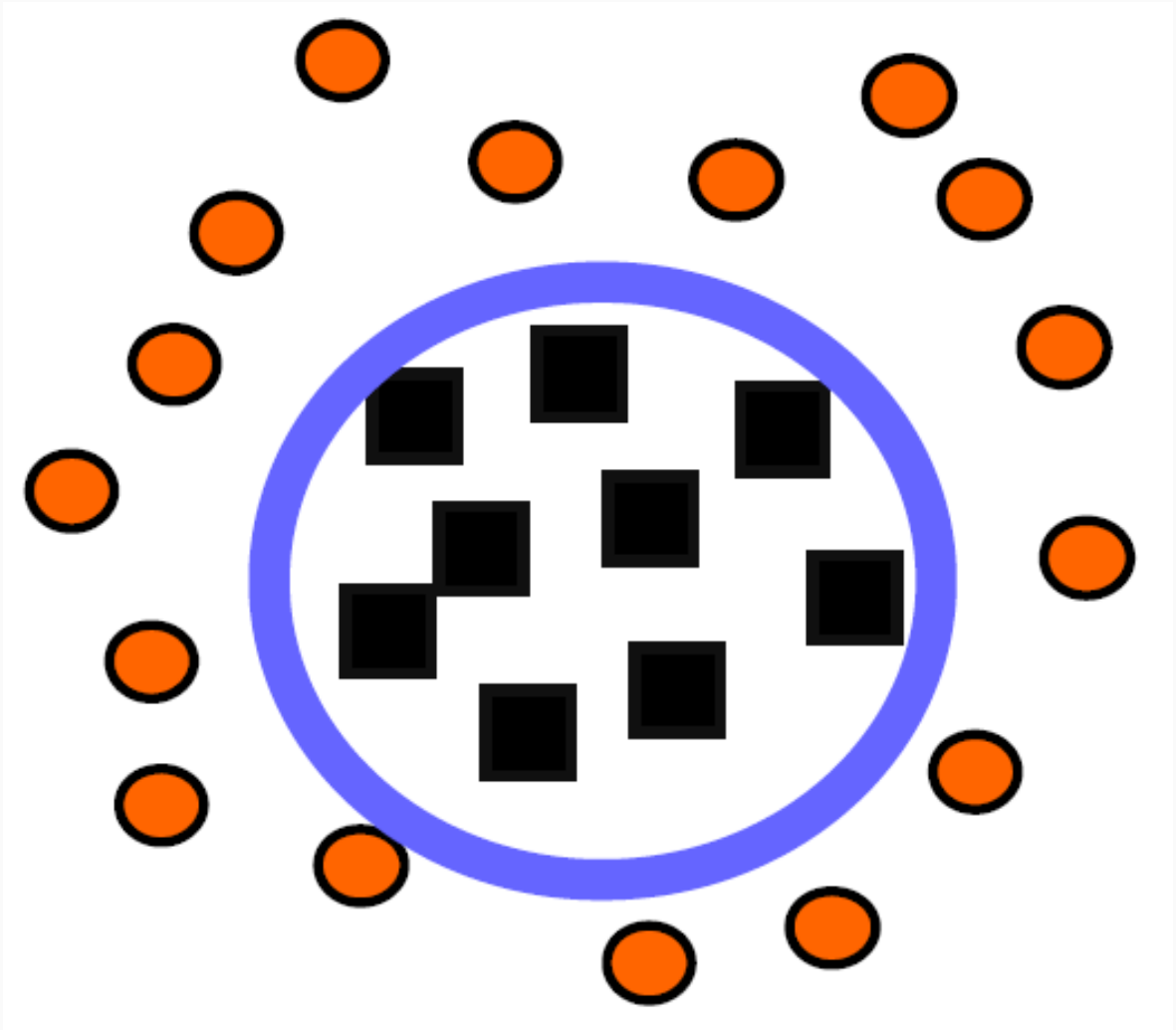
- Метод опорных векторов отлично подходит для линейной классификации.
- Решая задачу квадратичного программирования, мы получаем параметры оптимальной гиперплоскости.
- Точно так же, как и в дуальном случае, если бы мы просто искали середину между выпуклыми оболочками.

SVM и разделение нелинейными функциями

Введение

- Часто бывает нужно разделять данные не только линейными функциями.
- Что делать в таком случае?

Пример



Введение

- Часто бывает нужно разделять данные не только линейными функциями.
- Классический метод: развернуть нелинейную классификацию в пространство большей размерности (feature space), а там запустить линейный классификатор.
- Для этого просто нужно для каждого монома нужной степени ввести новую переменную.

Пример

- Чтобы в двумерном пространстве $[r, s]$ решить задачу классификации квадратичной функцией, надо перейти в пятимерное пространство:

$$[r, s] \longrightarrow [r, s, rs, r^2, s^2].$$

- Или формальнее; определим $\theta : \mathbb{R}^2 \to \mathbb{R}^5$: $\theta(r, s) = (r, s, rs, r^2, s^2)$. Вектор в $\mathbb{R}^5$ теперь соответствует квадратичной кривой общего положения в $\mathbb{R}^2$, а функция классификации выглядит как

$$f(\vec{x}) = \operatorname{sign}(\theta(\vec{w}) \cdot \theta(\vec{x}) - b).$$

- Если решить задачу линейного разделения в этом новом пространстве, тем самым решится задача квадратичного разделения в исходном.

Проблемы с классическим подходом

- Во-первых, количество переменных растёт экспоненциально.
- Во-вторых, по большому счёту теряются преимущества того, что гиперплоскость именно оптимальная; например, оверфиттинг опять становится проблемой.
- Важное замечание: *концептуально* мы задачу уже решили. Остались *технические* сложности: как обращаться с гигантской размерностью. Но в них-то всё и дело.

Основная идея и схема работы SVM

- Тривиальная схема алгоритма классификации такова:
  - входной вектор $\vec{x}$ трасформируется во входной вектор в feature space (большой размерности);
  - в этом большом пространстве мы вычисляем опорные векторы, решаем задачу разделения;
  - потом по этой задаче классифицируем входной вектор.
- Это нереально, потому что пространство слишком большой размерности.

Основная идея и схема работы SVM

- Оказывается, кое-какие шаги здесь можно переставить. Вот так:
  - опорные векторы вычисляются в исходном пространстве малой размерности;
  - там же они перемножаются (сейчас увидим, что это значит);
  - и только потом мы делаем нелинейную трансформацию того, что получится;
  - потом по этой задаче классифицируем входной вектор.
- Осталось теперь объяснить, что всё это значит. :)

Постановка задачи

- Напомним, что наша задача поставлена следующим образом:

$$\min_{\alpha} \left\{ \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{m} y_i y_j \alpha_i \alpha_j \left(\vec{x}_i \cdot \vec{x}_j \right) - \sum_{i=1}^{m} \alpha_i, \right.$$
$$\left. \text{где } \sum_{i=1}^{m} y_i \alpha_i = 0, \quad 0 \leq \alpha_i \leq C. \right\}$$

Постановка задачи

- Мы теперь хотим ввести некое отображение $\theta : \mathbb{R}^n \to \mathbb{R}^N$, $N > n$. Получится:

$$\min_{\alpha} \left\{ \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{m} y_i y_j \alpha_i \alpha_j \left(\theta(\vec{x}_i) \cdot \theta(\vec{x}_j) \right) - \sum_{i=1}^{m} \alpha_i, \right.$$
$$\left. \text{где } \sum_{i=1}^{m} y_i \alpha_i = 0, \quad 0 \leq \alpha_i \leq C. \right\}$$

Теория Гильберта--Шмидта--Мерсера

- Придётся немножко вспомнить (или изучить) функциональный анализ.
- Мы хотим обобщить понятие *скалярного произведения*; давайте введём новую функцию, которая (минуя трансформацию) будет сразу вычислять скалярное произведение векторов в feature space:

$$k(\vec{u}, \vec{v}) := \theta(\vec{u}) \cdot \theta(\vec{v}).$$

Теория Гильберта--Шмидта--Мерсера

- Первый результат: любая симметрическая функция $k(\vec{u}, \vec{v}) \in L_2$ представляется в виде

$$k(\vec{u}, \vec{v}) = \sum_{i=1}^{\infty} \lambda_i \theta_i(\vec{u}) \cdot \theta_i(\vec{v}),$$

где $\lambda_i \in \mathbb{R}$ — собственные числа, а θ_i — собственные векторы интегрального оператора с ядром k, т.е.

$$\int k(\vec{u}, \vec{v}) \theta_i(\vec{u}) \mathrm{d}\vec{u} = \lambda_i \theta_i(\vec{v}).$$

Теория Гильберта--Шмидта--Мерсера

- Чтобы k задавало скалярное произведение, достаточно, чтобы все собственные числа были положительными. А собственные числа положительны тогда и только тогда, когда (*теорема Мерсера*)

$$\int\int k(\vec{u},\vec{v})g(\vec{u})g(\vec{v})\mathrm{d}\vec{u}\mathrm{d}\vec{v} > 0$$

  для всех g таких, что $\int g^2(\vec{u})\mathrm{d}\vec{u} < \infty$.
- Вот, собственно и всё. Теперь мы можем вместо подсчёта $\theta(\vec{u}) \cdot \theta(\vec{v})$ в задаче квадратичного программирования просто использовать подходящее *ядро* $k(\vec{u},\vec{v})$.

Теория Гильберта--Шмидта--Мерсера

- Итого задача наша выглядит так:

$$\min_{\alpha} \left\{ \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{m} y_i y_j \alpha_i \alpha_j k(\vec{x}_i, \vec{x}_j) - \sum_{i=1}^{m} \alpha_i, \right.$$
$$\left. \text{где } \sum_{i=1}^{m} y_i \alpha_i = 0, \quad 0 \leq \alpha_i \leq C. \right\}$$

- Просто меняя ядро k, мы можем вычислять самые разнообразные разделяющие поверхности.
- Условия на то, чтобы k была подходящим ядром, задаются теоремой Мерсера.

Примеры ядер

- Рассмотрим ядро
$$k(\vec{u}, \vec{v}) = (\vec{u} \cdot \vec{v})^2.$$
- Какое пространство ему соответствует?

Примеры ядер

- После выкладок получается:

$$k(\vec{u}, \vec{v}) = (\vec{u} \cdot \vec{v})^2 = \\ = \left(u_1^2, u_2^2, \sqrt{2}u_1 u_2\right) \cdot \left(v_1^2, v_2^2, \sqrt{2}v_1 v_2\right).$$

- Иначе говоря, линейная поверхность в новом пространстве соответствует квадратичной поверхности в исходном (эллипс, например).

Примеры ядер

- Естественное обобщение: ядро $k(\vec{u}, \vec{v}) = (\vec{u} \cdot \vec{v})^d$ задаёт пространство, оси которого соответствуют всем *однородным* мономам степени d.
- А как сделать пространство, соответствующее произвольной полиномиальной поверхности, не обязательно однородной?

Примеры ядер

- Поверхность, описывающаяся полиномом степени d:

$$k(\vec{u}, \vec{v}) = (\vec{u} \cdot \vec{v} + 1)^d.$$

- Тогда линейная разделимость в feature space в точности соответствует полиномиальной разделимости в базовом пространстве.

Примеры ядер

- Нормальное распределение (radial basis function):

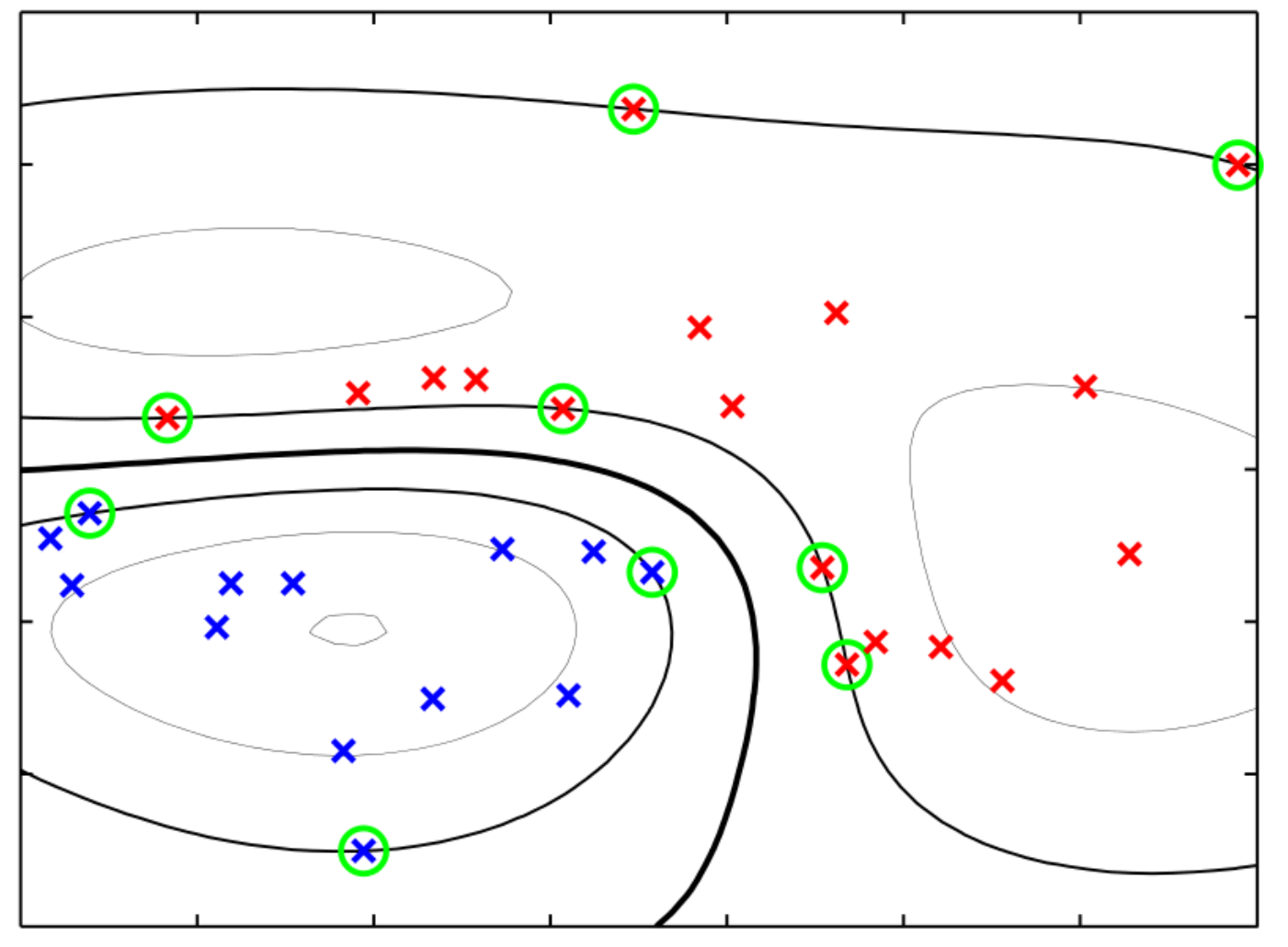
$$k(\vec{u}, \vec{v}) = e^{-\frac{||\vec{u}-\vec{v}||^2}{2\sigma}}.$$

- Двухуровневая нейронная сеть:

$$k(\vec{u}, \vec{v}) = o(\eta\vec{u} \cdot \vec{v} + c),$$

где o — сигмоид.

Пример



SVM и эмпирический риск

- Ещё один взгляд на SVM — какая вообще задача у любой классификации?
- Мы хотим минимизировать эмпирический риск, то есть число неправильных ответов:

$$\sum_n [y_i \neq t_i] \to \min_{\mathbf{w}}.$$

- И если функция линейная с параметрами $\mathbf{w}$, w_0, то это эквивалентно

$$\sum_n \left[t_i \left(\mathbf{x}_n^\top \mathbf{w} - w_0\right) < 0\right] \to \min_{\mathbf{w}}.$$

- Величину $M_i = \mathbf{x}_n^\top \mathbf{w} - w_0$ назовём *отступом* (margin).
- Оптимизировать напрямую сложно...

SVM и эмпирический риск

- ...поэтому заменим на оценку сверху:

$$\sum_n [M_i < 0] \leq \sum_n (1 - M_i) \to \min_{\mathbf{w}}.$$

- А потом ещё добавим регуляризатор для стабильности:

$$\sum_n [M_i < 0] \leq \sum_n (1 - M_i) + \frac{1}{2C}\|\mathbf{w}\|^2 \to \min_{\mathbf{w}}.$$

- И это снова получилась задача SVM!

Резюме

- Вот какой получается в итоге алгоритм.
  1. Выбрать параметр C, от которого зависит акцент на минимизации ошибки или на максимизации зазора.
  2. Выбрать ядро и параметры ядра, которые у него, возможно, есть.
  3. Решить задачу квадратичного программирования.
  4. По полученным значениям опорных векторов определить w_0 (как именно?).
  5. Новые точки классифицировать как

$$f(\vec{x}) = \text{sign}(\sum_i y_i \alpha_i k(\vec{x}, \vec{x}_i) - w_0).$$

Эквивалентное ядро

Эквивалентное ядро

- Вспомним наши байесовские предсказания:

$$p(t \mid \mathbf{t}, \alpha, \beta) = N(t \mid \mu_N^\top \phi(\mathbf{x}), \sigma_N^2),$$
$$\text{где } \sigma_N^2 = \frac{1}{\beta} + \phi(\mathbf{x})^\top \Sigma_N \phi(\mathbf{x}).$$

- Давайте перепишем среднее апостериорного распределения в другой форме (вспомним, что $\mu_N = \beta \Sigma_N \Phi^\top \mathbf{t}$):

$$y(\mathbf{x}, \mu_N) = \mu_N^\top \phi(\mathbf{x}) = \beta \phi(\mathbf{x})^\top \Sigma_N \Phi^\top \mathbf{t} =$$
$$= \sum_{n=1}^{N} \beta \phi(\mathbf{x})^\top \Sigma_N \phi(\mathbf{x}_n) t_n.$$

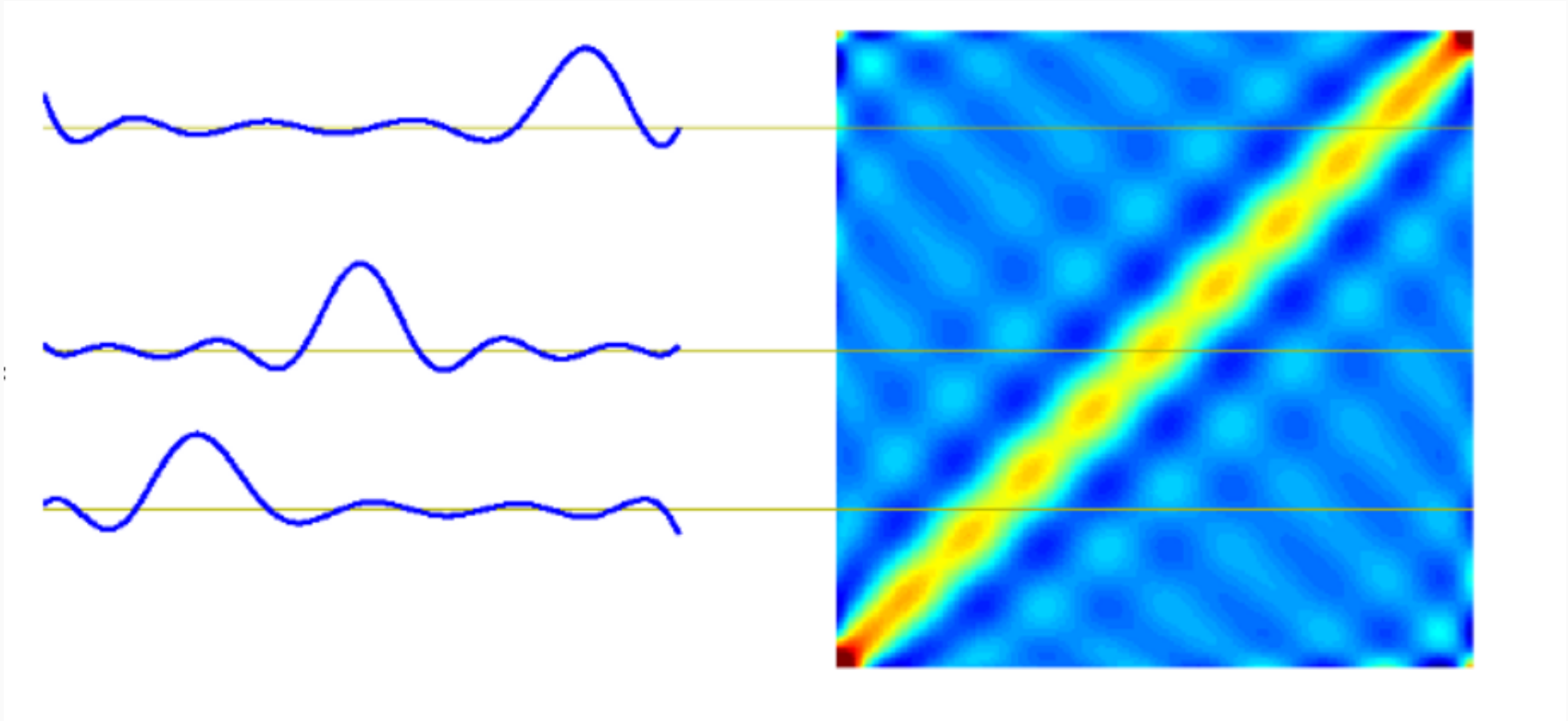
Эквивалентное ядро

- $y(\mathbf{x}, \mu_N) = \sum_{n=1}^{N} \beta\phi(\mathbf{x})^\top \Sigma_N \phi(\mathbf{x}_n) t_n$.
- Это значит, что предсказание можно переписать как

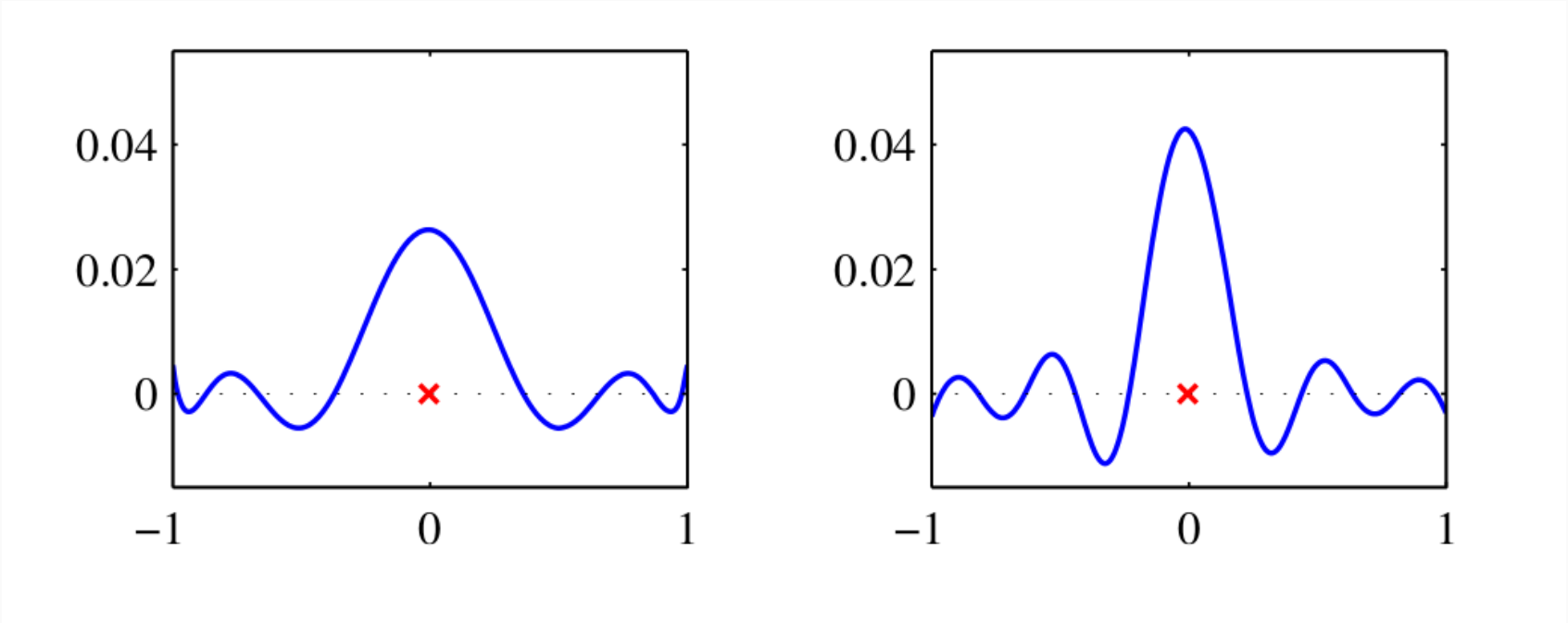
$$y(\mathbf{x}, \mu_N) = \sum_{n=1}^{N} k(\mathbf{x}, \mathbf{x}_n) t_n.$$

- Т.е. мы предсказываем следующую точку как линейную комбинацию значений в известных точках.
- Функция $k(\mathbf{x}, \mathbf{x}') = \beta\phi(\mathbf{x})^\top \Sigma_N \phi(\mathbf{x}')$ называется *эквивалентным ядром* (equivalent kernel).

Эквивалентное ядро



Эквивалентное ядро



Выводы про эквивалентное ядро

- Эквивалентное ядро $k(\mathbf{x}, \mathbf{x}')$ локализовано вокруг $\mathbf{x}$ как функция $\mathbf{x}'$, т.е. каждая точка оказывает наибольшее влияние около себя и затухает потом.
- Можно было бы с самого начала просто определить ядро и предсказывать через него, безо всяких базисных функций ϕ – такой подход мы ещё будем рассматривать.

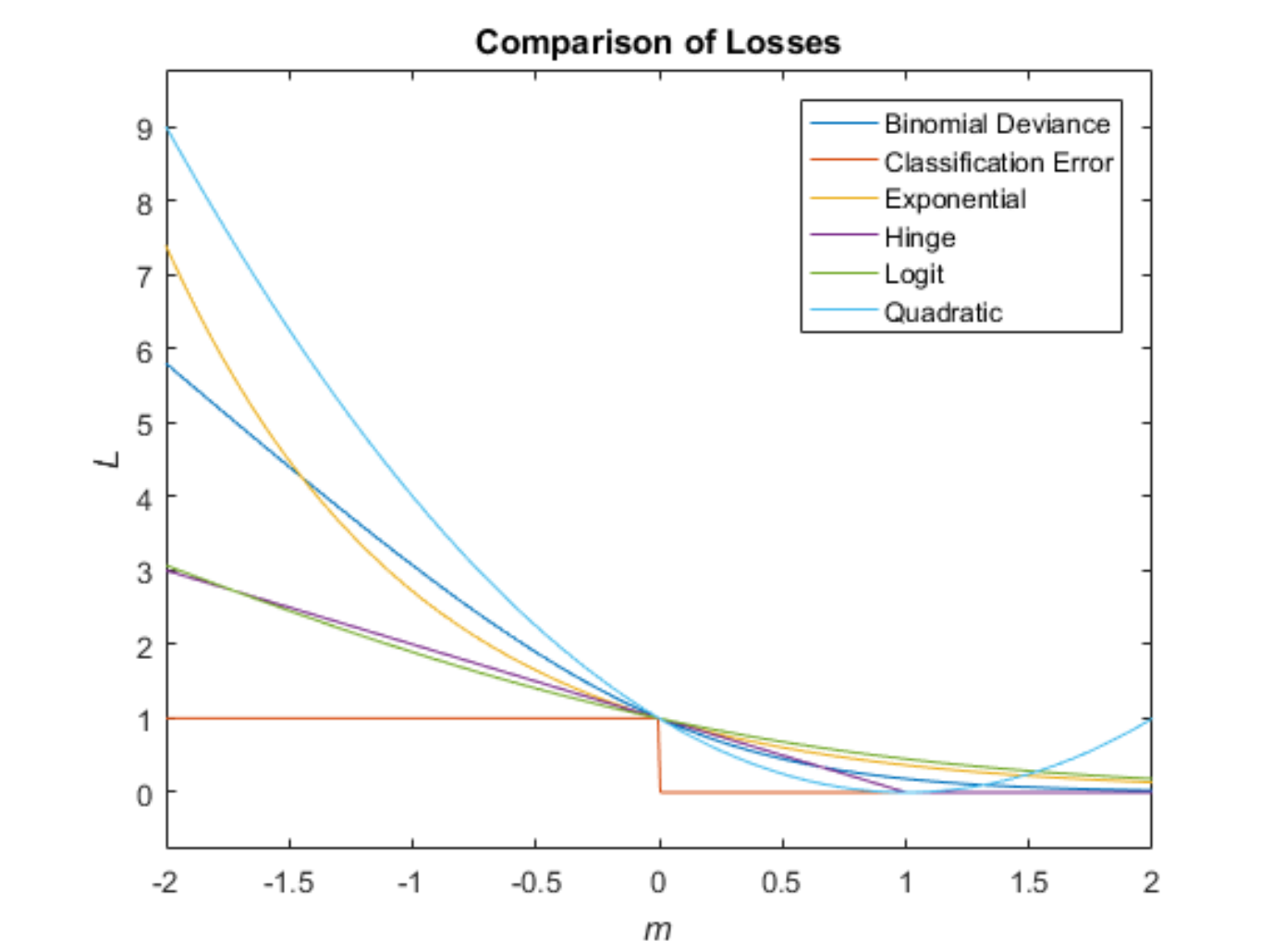
Упражнение. Докажите, что $\sum_{n=1}^{N} k(\mathbf{x}, \mathbf{x}_n) = 1$.

RVM

Функции потерь в классификации

- Сначала – ещё один важный другой взгляд: разные методы классификации отличаются друг от друга тем, какую функцию ошибки они оптимизируют.
- У классификации проблема с «правильной» функцией ошибки, то есть ошибкой собственно классификации:
  - она и не везде дифференцируема,
  - и производная её никому не нужна.
- Давайте посмотрим на разные функции потерь (loss functions); мы уже несколько видели, ещё кое-что осталось.

Функции потерь в классификации



Постановка задачи

- SVM – отличный метод. Но и у него есть недостатки.
  1. Выходы SVM – решения, а апостериорные вероятности непонятно как получить.
  2. SVM – для двух классов, обобщить на несколько проблематично.
  3. Есть параметр C (или ν, или ещё вдобавок ϵ), который надо подбирать.
  4. Предсказания – линейные комбинации ядер, которым необходимо быть положительно определёнными и которые центрированы на точках из датасета.
- Сейчас мы рассмотрим байесовский аналог SVM – *relevance vector machines* (RVM).

RVM для регрессии

- RVM удобнее сразу формулировать для регрессии.
- Вспомним обычную нашу линейную модель:

$$p(t \mid \mathbf{x}, \mathbf{w}, \beta) = N(t \mid y(\mathbf{x}), \beta^{-1}), \text{ где}$$

$$y(\mathbf{x}) = \sum_{i=1}^{M} w_i \phi_i(\mathbf{x}) = \mathbf{w}^\top \phi(\mathbf{x}).$$

RVM для регрессии

- RVM – это вариант такой модели, который старается работать как SVM.
- Рассмотрим

$$y(\mathbf{x}) = \sum_{n=1}^{N} w_n k(\mathbf{x}, \mathbf{x}_n) + b.$$

- Т.е. мы сразу ищем решение в форме линейной комбинации значений ядра (вспомним «эквивалентное ядро» для линейной регрессии), но, в отличие от SVM, теперь ядро никак не ограничивается.

RVM для регрессии

- Для N наблюдений вектора $\mathbf{x}$ (обозначим через $\mathbf{X}$) со значениями $\mathbf{t}$ получим правдоподобие

$$p(\mathbf{t} \mid \mathbf{X}, \mathbf{w}, \beta) = \prod_{n=1}^{N} p(t_n \mid \mathbf{x}_n, \mathbf{w}, \beta^{-1}).$$

- Априорное распределение тоже будет нормальное, но вместо единого гиперпараметра для всех весов мы введём отдельный гиперпараметр для каждого:

$$p(\mathbf{w} \mid \alpha) = \prod_{i=1}^{M} N(w_i \mid 0, \alpha_i^{-1}).$$

RVM для регрессии

- Отдельные гиперпараметры:

$$p(\mathbf{w} \mid \alpha) = \prod_{i=1}^{M} N(w_i \mid 0, \alpha_i^{-1}).$$

- Идея здесь в том, что при максимизации апостериорной вероятности большая часть α_i просто уйдёт на бесконечность, и соответствующие веса будут нулевыми.
- Сейчас увидим, как это получается.

RVM для регрессии

- Апостериорное распределение нам знакомо:

$$p(\mathbf{w} \mid \mathbf{t}, \mathbf{X}, \alpha, \beta) = N(\mathbf{w} \mid \mathbf{m}, \Sigma), \text{ где}$$

$$\mathbf{m} = \beta \Sigma \Phi^\top \mathbf{t},$$
$$\Sigma = \left(\mathbf{A} + \beta \Phi^\top \Phi\right)^{-1},$$

где $\mathbf{A} = \mathrm{diag}(\alpha_1, \dots, \alpha_M)$, а Φ в нашем случае – это $\mathbf{K}$, симметрическая матрица с элементами $k(\mathbf{x}_n, \mathbf{x}_m)$.

RVM для регрессии

- Как найти α и β? Нужно максимизировать маргинальное правдоподобие датасета

$$p(\mathbf{t} \mid \mathbf{X}, \alpha, \beta) = \int p(\mathbf{t} \mid \mathbf{X}, \mathbf{w}, \beta) p(\mathbf{w} \mid \alpha) d\mathbf{w}.$$

- Это свёртка двух гауссианов, тоже гауссиан:

$$\ln p(\mathbf{t} \mid \mathbf{X}, \alpha, \beta) = \ln N(\mathbf{t} \mid 0, \mathbf{C}) =$$

$$= -\frac{1}{2}\left[N \ln(2\pi) + \ln|\mathbf{C}| + \mathbf{t}^\top \mathbf{C}^{-1} \mathbf{t}\right], \text{ где } \mathbf{C} = \beta^{-1}\mathbf{I} + \Phi \mathbf{A}^{-1} \Phi^\top.$$

- Как это оптимизировать?

RVM для регрессии

- Можно подсчитать производные и получить

$$\alpha_i = \frac{\gamma_i}{m_i^2},$$
$$\beta^{-1} = \frac{\|\mathbf{t} - \Phi\mathbf{m}\|^2}{N - \sum_i \gamma_i},$$

  где $\gamma_i = 1 - \alpha_i \Sigma_{ii}$.

- Теперь можно просто итеративно пересчитывать α, β из $\mathbf{m}, \Sigma$, потом наоборот, потом опять наоборот, и до сходимости.

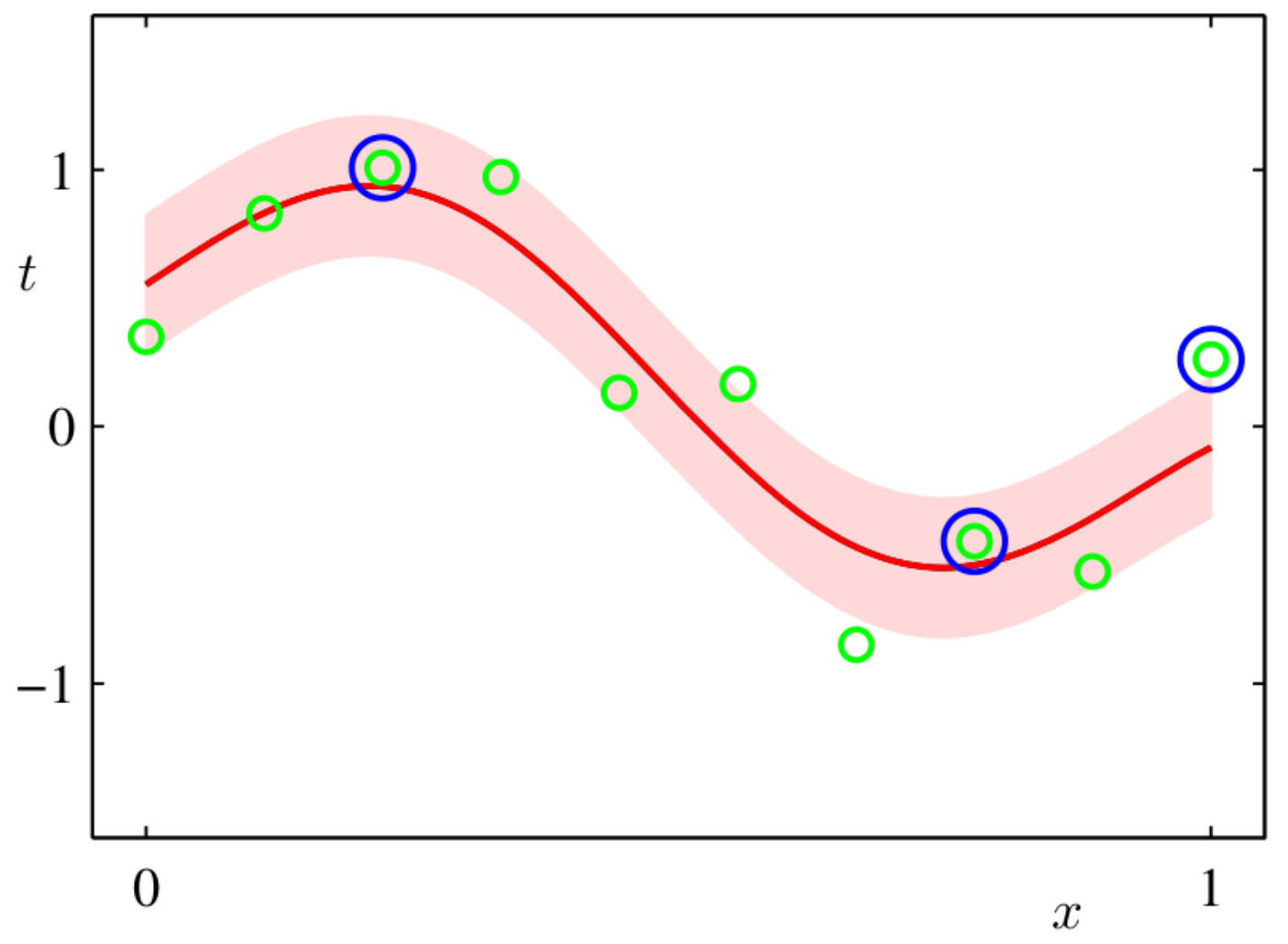
RVM для регрессии

- В результате получается обычно, что большинство α_i неограниченно растут, и соответствующие веса можно считать нулевыми.
- Оставшиеся называются *relevance vectors*, их обычно мало.
- Если теперь мы найдём α^*, β^*, то предсказывать в новых точках можно как

$$p(t \mid \mathbf{x}, \mathbf{X}, \mathbf{t}, \alpha^*, \beta^*) = \int p(t \mid \mathbf{x}, \mathbf{w}, \beta^*) p(\mathbf{w} \mid \mathbf{X}, \mathbf{t}, \alpha^*, \beta^*) d\mathbf{w} = \\ = N(t \mid \mathbf{m}^\top \phi(\mathbf{x}), \sigma^2(\mathbf{x})),$$

где $\sigma^2(\mathbf{x}) = (\beta^*)^{-1} + \phi(\mathbf{x})^\top \Sigma \phi(\mathbf{x})$.

RVM для регрессии



RVM для классификации

- Можно сделать то же самое и для классификации. Рассмотрим классификацию с двумя классами, $t \in \{0, 1\}$:

$$y(\mathbf{x}, \mathbf{w}) = \sigma(\mathbf{w}^\top \phi(\mathbf{x})).$$

- И добавим сюда, опять же, априорное распределение с разными α_i для каждого веса:

$$p(\mathbf{w} \mid \alpha) = \prod_{i=1}^{M} N(w_i \mid 0, \alpha_i^{-1}).$$

- Идея: инициализируем α, считаем лапласовское приближение к апостериорному распределению, максимизируем, получаем новое α, и т.д.

RVM для классификации

- Апостериорное распределение:

$$\ln p(\mathbf{w} \mid \mathbf{t}, \alpha) = \ln \left(p(\mathbf{t} \mid \mathbf{w}) p(\mathbf{w} \mid \alpha)\right) - \ln p(\mathbf{t} \mid \alpha) =$$
$$= \sum_{n=1}^{N} \left[t_n \ln y_n + (1 - t_n) \ln(1 - y_n)\right] - \frac{1}{2}\mathbf{w}^\top \mathbf{A}\mathbf{w} + \text{const}.$$

- Мы уже обсуждали, как его максимизировать – IRLS; для этого подсчитаем

$$\nabla \ln p(\mathbf{w} \mid \mathbf{t}, \alpha) = \Phi^\top (\mathbf{t} - \mathbf{y}) - \mathbf{A}\mathbf{w},$$
$$\nabla\nabla \ln p(\mathbf{w} \mid \mathbf{t}, \alpha) = -\left(\Phi^\top \mathbf{B} \Phi + \mathbf{A}\right),$$

где $\mathbf{B}$ – диагональная матрица с элементами $b_n = y_n(1 - y_n)$.

RVM для классификации

- Лапласовское приближение получится из $\nabla \ln p(\mathbf{w} \mid \mathbf{t}, \alpha)$, и получится

$$\mathbf{w}^* = \mathbf{A}^{-1}\Phi^\top (\mathbf{t} - \mathbf{y}),$$
$$\Sigma = \left(\Phi^\top \mathbf{B}\Phi + \mathbf{A}\right)^{-1},$$

и распределение для предсказания получится

$$p(\mathbf{t} \mid \alpha) = \int p(\mathbf{t} \mid \mathbf{w}) p(\mathbf{w} \mid \alpha) d\mathbf{w} \approx$$
$$\approx p(\mathbf{t} \mid \mathbf{w}^*) p(\mathbf{w}^* \mid \alpha) (2\pi)^{M/2} |\Sigma|^{1/2}.$$

RVM для классификации

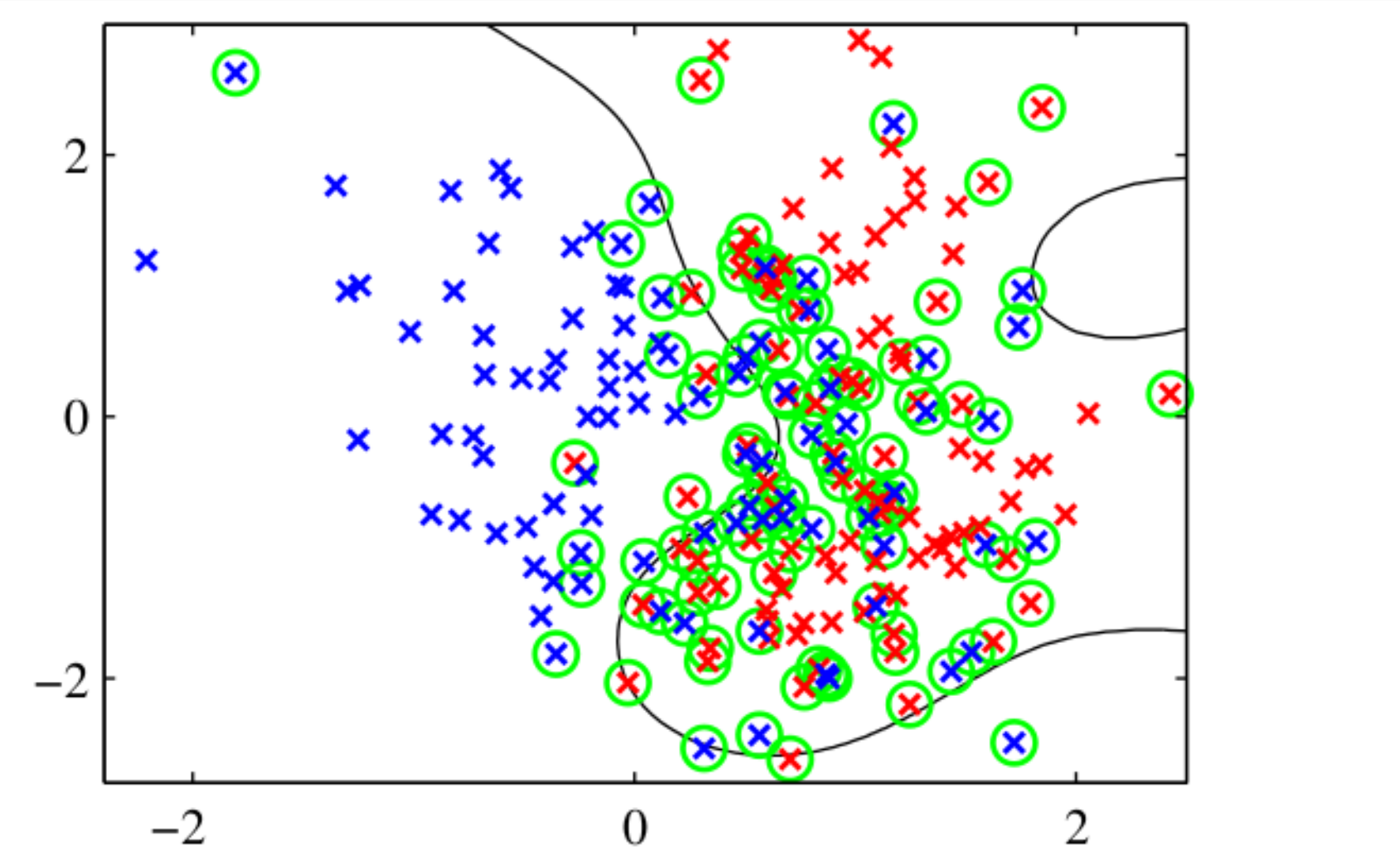
- $p(\mathbf{t} \mid \alpha) = \int p(\mathbf{t} \mid \mathbf{w}) p(\mathbf{w} \mid \alpha) d\mathbf{w} \approx$
  $p(\mathbf{t} \mid \mathbf{w}^*) p(\mathbf{w}^* \mid \alpha) (2\pi)^{M/2} |\Sigma|^{1/2}$.
- Теперь мы оптимизируем это по α: берём производную, получаем

$$-\frac{1}{2}(w_i^*)^2 + \frac{1}{2\alpha_i} - \frac{1}{2}\Sigma_{ii} = 0, \text{ т.е.}$$

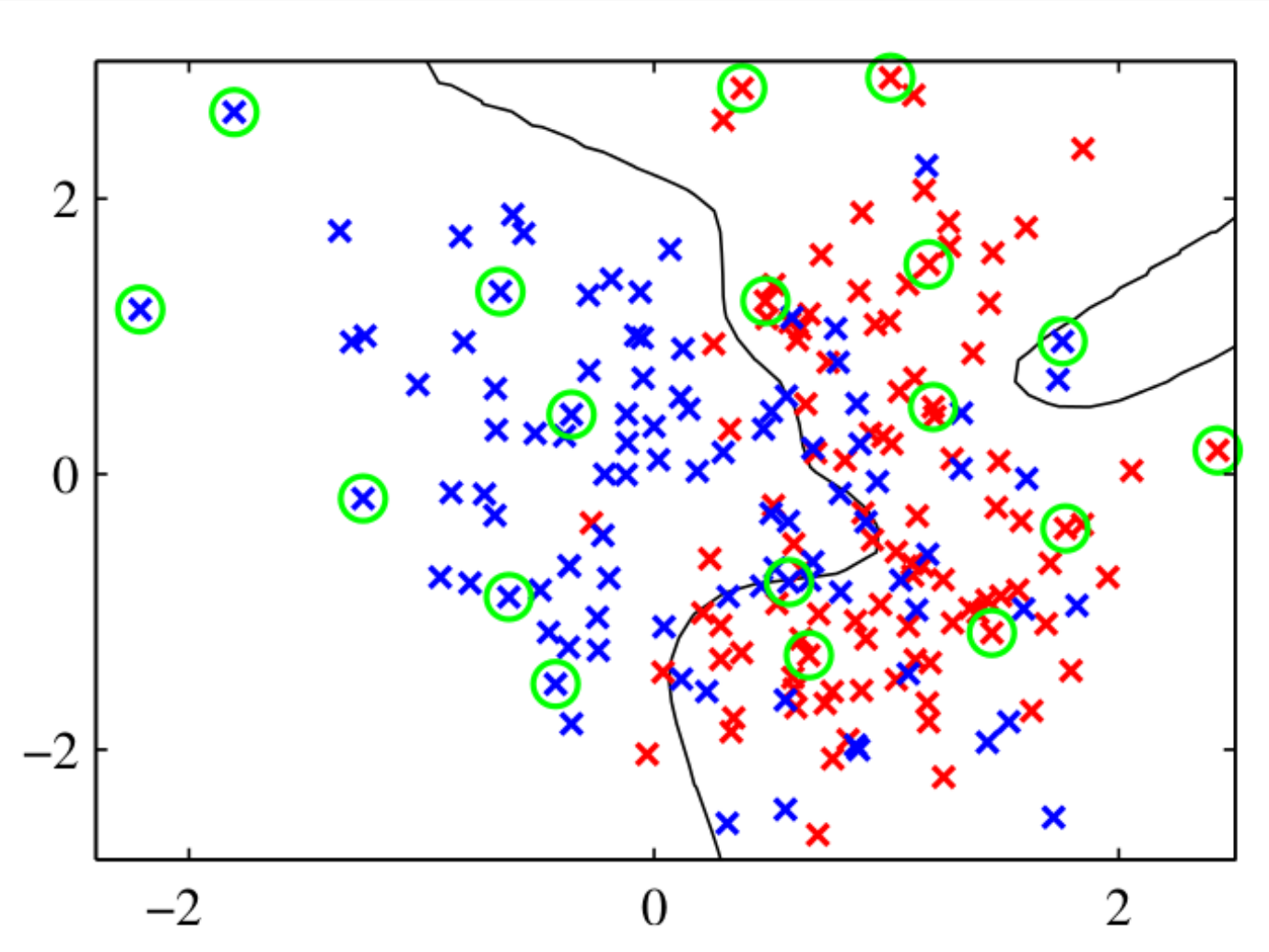
$$\alpha_i = \frac{\gamma_i}{(w_i^*)^2}, \ \gamma_i = 1 - \alpha_i \Sigma_{ii}.$$

- Т.е. формула получилась точно такая же, как в случае регрессии.

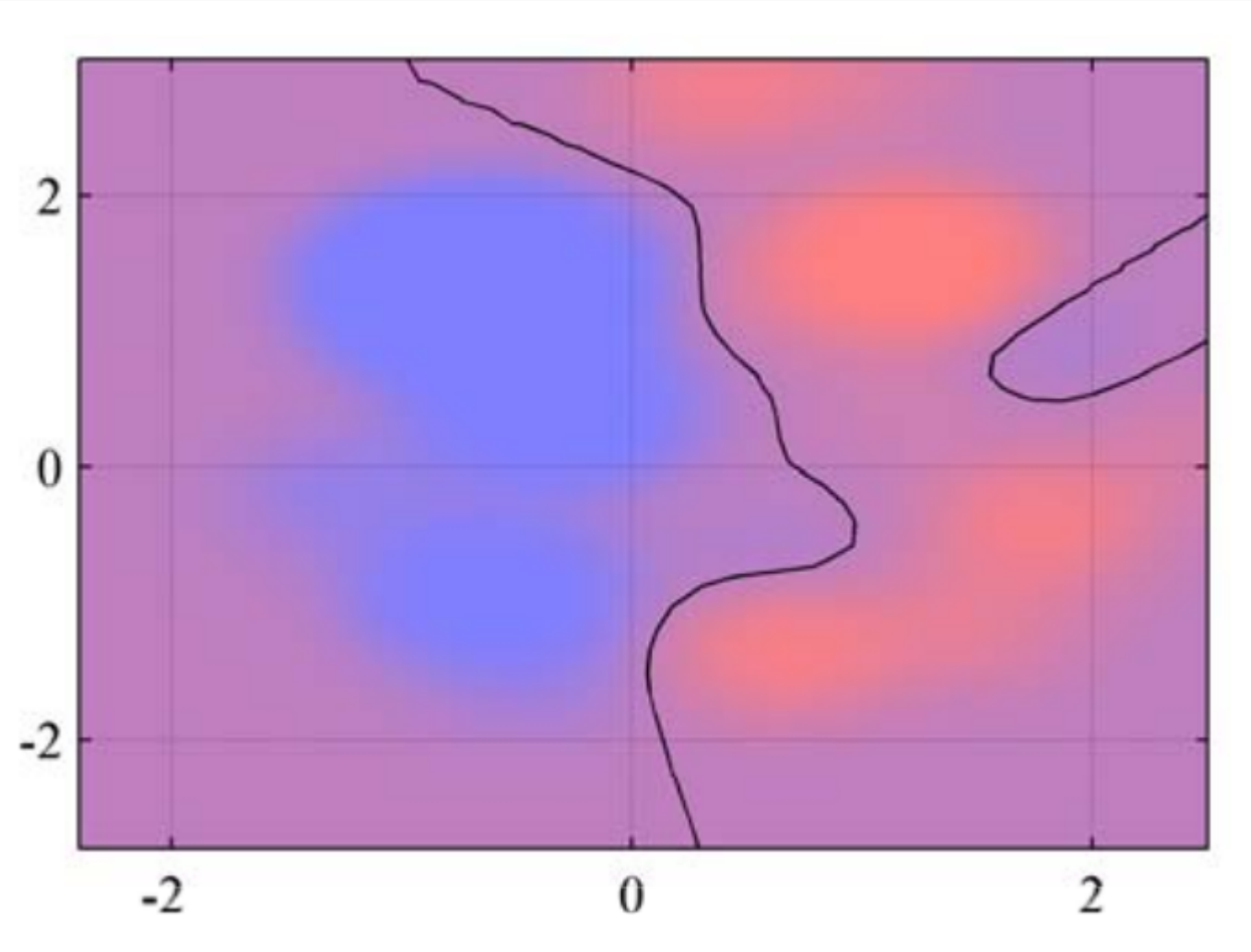
Было: SVM



Стало: RVM



Стало: RVM



RVM для нескольких классов

- На несколько классов теперь обобщается естественным образом:

$$a_k = \mathbf{w}_k^\top \mathbf{x}, \quad y_k(\mathbf{x}) = \frac{e^{a_k}}{\sum_j e^{a_j}}.$$

- И дальше всё то же самое.

Сравнение SVM и RVM

- RVM как-то получилось лучше со всех сторон.
- Главный минус – в RVM обучение гораздо дольше (хотя есть алгоритмы и побыстрее, чем мы рассматривали, но всё равно дольше).
- Но даже это не то чтобы минус, потому что в SVM нужна кросс-валидация для подбора параметров, т.е. на самом деле обучение SVM дольше, чем кажется.
- Есть и (даже более важный) плюс с точки зрения скорости – в RVM гораздо быстрее применение модели к новым точкам, потому что опорных векторов гораздо меньше.

Быстрый алгоритм обучения RVM

- В RVM для регрессии получается правдоподобие

$$\ln p(\mathbf{t} \mid \mathbf{X}, \alpha, \beta) = \ln N(\mathbf{t} \mid 0, \mathbf{C}) =$$

$$= -\frac{1}{2}\left[N \ln(2\pi) + \ln|\mathbf{C}| + \mathbf{t}^\top \mathbf{C}^{-1}\mathbf{t}\right], \text{ где } \mathbf{C} = \beta^{-1}\mathbf{I} + \Phi \mathbf{A}^{-1}\Phi^\top.$$

Быстрый алгоритм обучения RVM

- Выделим вклад в $\mathbf{C}$ одного компонента α_i:

$$\mathbf{C} = \beta^{-1}\mathbf{I} + \sum_{j \neq i} \alpha_j^{-1}\varphi_j\varphi_j^\top + \alpha_i^{-1}\varphi_i\varphi_i^\top = \\ = \mathbf{C}_{-i} + \alpha_i^{-1}\varphi_i\varphi_i^\top,$$

где φ_i – i-я строка Φ (ϕ_n был n-м столбцом).

Быстрый алгоритм обучения RVM

- $\mathbf{C} = \mathbf{C}_{-i} + \alpha_i^{-1}\varphi_i\varphi_i^\top$.
- Верны следующие тождества для определителя и обратной матрицы:

$$|\mathbf{C}| = |\mathbf{C}_{-i}||1 + \alpha_i^{-1}\varphi_i^\top \mathbf{C}_{-i}^{-1}\varphi_i|,$$
$$\mathbf{C}^{-1} = \mathbf{C}_{-i}^{-1} - \frac{\mathbf{C}_{-i}^{-1}\varphi_i\varphi_i^\top\mathbf{C}_{-i}^{-1}}{\alpha_i + \varphi_i^\top\mathbf{C}_{-i}^{-1}\varphi_i}.$$

Упражнение. Докажите это.

Быстрый алгоритм обучения RVM

- Значит, $L(\alpha)$ можно переписать в виде

$$L(\alpha) = L(\alpha_{-i}) + \lambda(\alpha_i), \text{ где}$$

$$\lambda(\alpha_i) = \frac{1}{2}\left[\ln \alpha_i - \ln(\alpha_i + s_i) + \frac{q_i^2}{\alpha_i + s_i}\right].$$

- Здесь

$$\begin{aligned} s_i &= \varphi_i^\top \mathbf{C}_{-i}^{-1} \varphi_i && \text{sparsity } \varphi_i \\ q_i &= \varphi_i^\top \mathbf{C}_{-i}^{-1} \mathbf{t} && \text{quality } \varphi_i. \end{aligned}$$

Быстрый алгоритм обучения RVM

- $s_i = \varphi_i^\top \mathbf{C}_{-i}^{-1} \varphi_i$, $q_i = \varphi_i^\top \mathbf{C}_{-i}^{-1} \mathbf{t}$.
- Sparsity – то, насколько φ_i перекрывается с остальными векторами модели.
- Quality – то, насколько φ_i сонаправлен с ошибкой между $\mathbf{t}$ и $\mathbf{y}_{-i}$ (ошибкой модели без φ_i).
- Чем больше sparsity и чем меньше quality, тем более вероятно, что этот базисный вектор из модели исключат (т.е. $\alpha_i \to \infty$).

Быстрый алгоритм обучения RVM

- $\lambda(\alpha_i) = \frac{1}{2}\left[\ln \alpha_i - \ln(\alpha_i + s_i) + \frac{q_i^2}{\alpha_i + s_i}\right]$.
- Возьмём производную, приравняем нулю, получим (т.к. $\alpha_i \geq 0$)

$$\alpha_i = \begin{cases} \infty, & q_i^2 \leq s_i, \\ \frac{s_i^2}{q_i^2 - s_i}, & q_i^2 > s_i. \end{cases}$$

- Как мы и ожидали.

Быстрый алгоритм обучения RVM

- И алгоритм теперь получается такой:
  1. инициализировать β, φ_1, $\alpha_1 = s_1^2/(q_1^2 - s_1)$, остальные $\alpha_j = \infty$;
  2. вычислить Σ, $\mathbf{m}$, q_i и s_i для всех i;
  3. выбрать i, проапдейтить α_i, проапдейтить β;
  4. goto 2 и так пока не сойдётся.

Спасибо за внимание!