

# МЕТОД ОПОРНЫХ ВЕКТОРОВ

---

Сергей Николенко

uData School — Киев

15 июня 2018 г.

---

## *Random facts:*

- 15 июня 1215 г. Иоанн Безземельный встретился в долине Раннимед на берегу Темзы с мятежными баронами и приложил королевскую печать к баронским статьям, на основе которых была чуть позже составлена Великая хартия вольностей
- 15 июня 1667 г. Жан-Батист Дени перелил около 12 унций крови ягнёнка 15-летнему юноше, 15 июня 1752 г. Бенджамин Франклин запустил своего знаменитого воздушного змея, 15 июня 1844 г. Чарльз Гудьир запатентовал способ вулканизации резины, 15 июня 1851 г. Джекоб Фуссель основал в Балтиморе первую в мире фабрику мороженого, а 15 июня 1869 г. Джон Хайат запатентовал целлулоид
- 15 июня 1985 г. литовец Брюнос Майгис пришёл в зал Рембрандта в Эрмитаже и спросил у работниц музея, где здесь самая ценная картина; указали на «Даная»

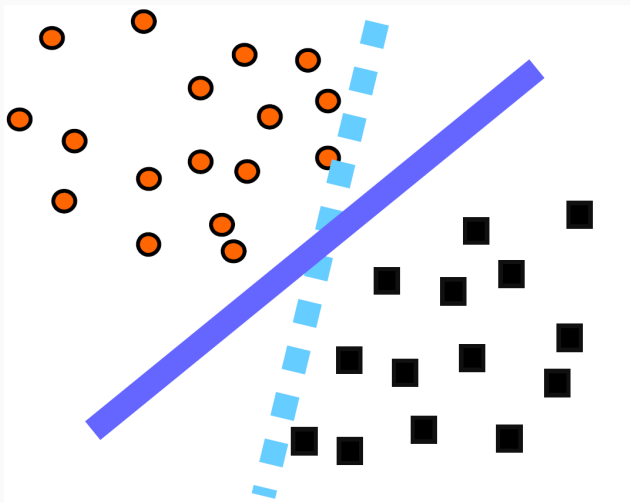
# SVM и ЗАДАЧА ЛИНЕЙНОЙ КЛАССИФИКАЦИИ

---

## 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?

- 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.

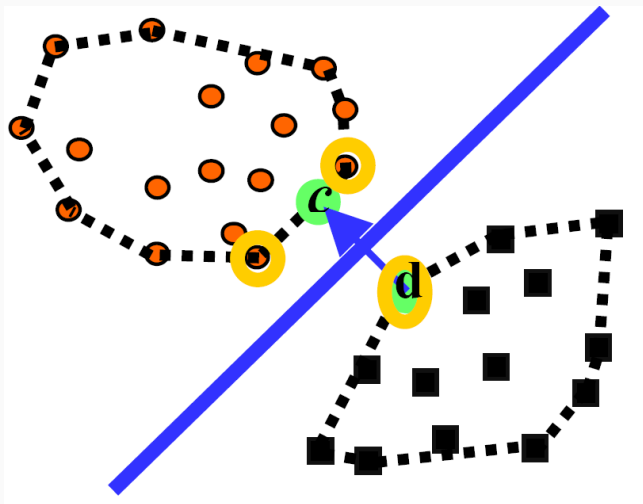


- Один подход: найти две ближайшие точки в выпуклых оболочках данных, а затем провести разделяющую гиперплоскость через середину отрезка.
- Формально это превращается в задачу квадратичной оптимизации:

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

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.



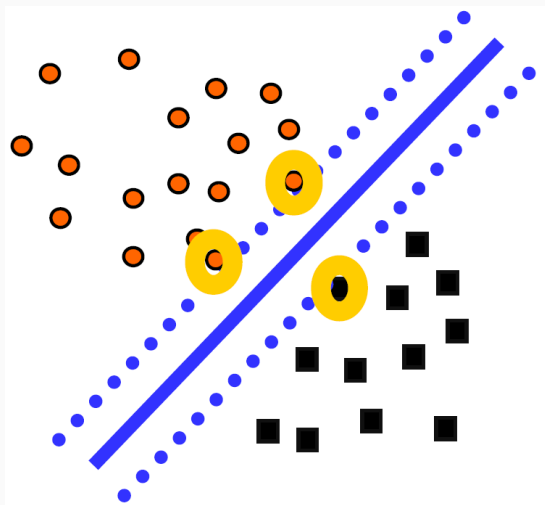
- Другой подход: максимизировать зазор (margin) между двумя параллельными опорными плоскостями, затем провести им параллельную на равных расстояниях от них.
- Гиперплоскость называется *опорной* для множества точек  $X$ , если все точки из  $X$  лежат под одну сторону от этой гиперплоскости.
- Формально: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



- Расстояние от точки до гиперплоскости  $y(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + w_0 = 0$  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

$$\begin{aligned} \arg \max_{\mathbf{w}, w_0} \min_n \frac{t_n y(\mathbf{x}_n)}{\|\mathbf{w}\|} &= \\ &= \arg \max_{\mathbf{w}, w_0} \left\{ \frac{1}{\|\mathbf{w}\|} \min_n [t_n (\mathbf{w}^T \mathbf{x}_n + w_0)] \right\}. \end{aligned}$$

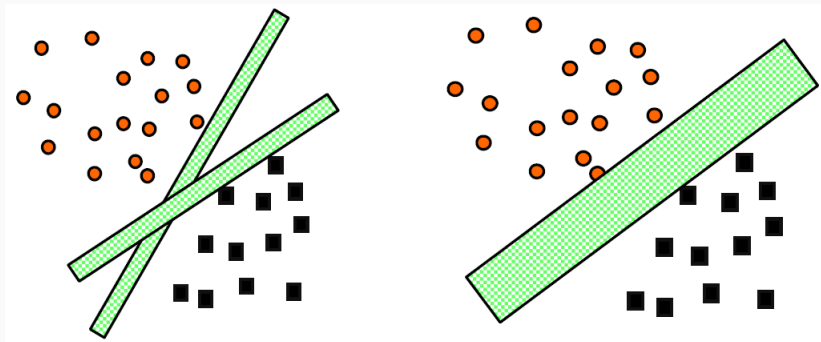
- $\arg \max_{\mathbf{w}, w_0} \left\{ \frac{1}{\|\mathbf{w}\|} \min_n [t_n(\mathbf{w}^\top \mathbf{x}_n + w_0)] \right\}$ . Сложно.
- Но если перенормировать  $\mathbf{w}$ , гиперплоскость не изменится.
- Давайте перенормируем так, чтобы  $\min_n [t_n(\mathbf{w}^\top \mathbf{x}_n + w_0)] = 1$ .



- 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 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).



- 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  $\lambda$ , соответствующие равенствам, и  $\mu$ , соответствующие неравенствам.

- Прямая задача оптимизации:

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

- Дуальная задача оптимизации:

$$\min \{\phi(\lambda, \mu)\} \text{ при условии } \mu \geq 0,$$

$$\text{где } \phi(\lambda, \mu) = \inf_{x \in X} \{f(x) + \lambda^\top h(x) + \mu^\top g(x)\}.$$



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

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

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

- 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:

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

- Для линейного программирования прямая задача:

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

- Дуальная задача:

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

- Для квадратичного программирования прямая задача:

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

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

- Дуальная задача (проверьте):

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

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

- В случае SVM надо ввести множители Лагранжа:

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

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

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

$$0 = \sum_n \alpha_n t_n.$$

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

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

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

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

- 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  $\mathbf{x}_n$ ...

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

$$\alpha_n \geq 0,$$

$$t_n y(\mathbf{x}_n) - 1 \geq 0,$$

$$\alpha_n (t_n y(\mathbf{x}_n) - 1) = 0.$$

- 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).

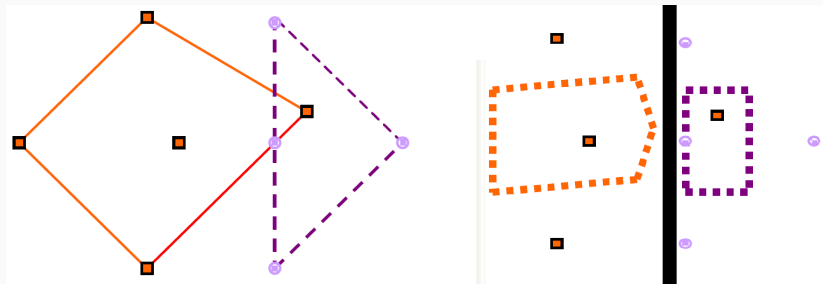


- Все эти методы работают, когда данные действительно линейно делимы.
- А что делать, когда их всё-таки немножко не получается разделить?
- Первый вопрос: что делать для первого метода, метода выпуклых оболочек?

- 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.



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

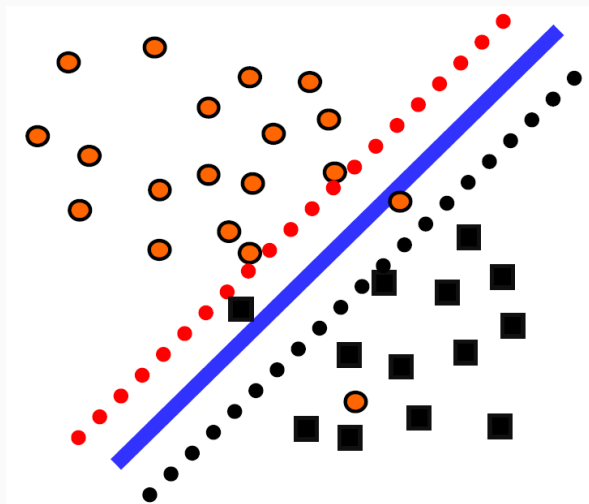
## Для МЕТОДА ОПОРНЫХ ВЕКТОРОВ

- 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...



- ...а вот дуальная:

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

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

- 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.



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

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

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

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

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

- ...поэтому заменим на оценку сверху:

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

- А потом ещё добавим регуляризатор для стабильности:

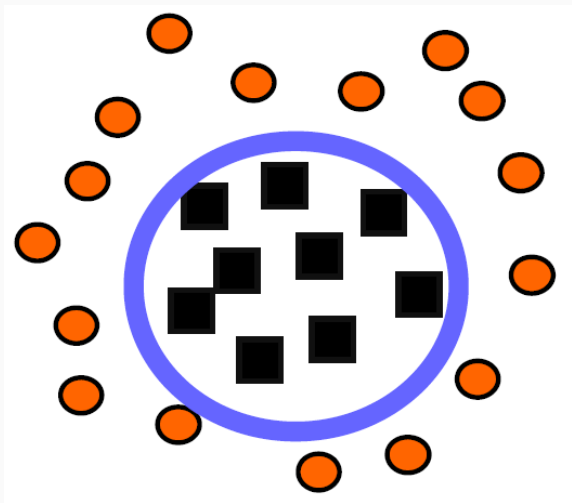
$$\sum_n [M_i < 0] \leq \sum_n (1 - M_i) + \frac{1}{2C} \|\mathbf{w}\|^2 \rightarrow \min_{\mathbf{w}}.$$

- И это снова получилась задача SVM!

# SVM и РАЗДЕЛЕНИЕ НЕЛИНЕЙНЫМИ ФУНКЦИЯМИ

---

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



- 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.

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

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

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

- 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.



- Тривиальная схема алгоритма классификации такова:
  - входной вектор  $\vec{x}$  трансформируется 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.

- 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. :)

- Напомним, что наша задача поставлена следующим образом:

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

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

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

- 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:

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

- 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  $\theta_i$  — собственные векторы интегрального оператора с ядром  $k$ , т.е.

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

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

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

для всех  $g$  таких, что  $\int g^2(\vec{u}) d\vec{u} < \infty$ .

- Вот, собственно и всё. Теперь мы можем вместо подсчёта  $\theta(\vec{u}) \cdot \theta(\vec{v})$  в задаче квадратичного программирования просто использовать подходящее ядро  $k(\vec{u}, \vec{v})$ .

- 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.



- Рассмотрим ядро

$$k(\vec{u}, \vec{v}) = (\vec{u} \cdot \vec{v})^2.$$

- Какое пространство ему соответствует?

- После выкладок получается:

$$\begin{aligned}k(\vec{u}, \vec{v}) &= (\vec{u} \cdot \vec{v})^2 = \\ &= (u_1^2, u_2^2, \sqrt{2}u_1u_2) \cdot (v_1^2, v_2^2, \sqrt{2}v_1v_2).\end{aligned}$$

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

- 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?

- Поверхность, описываемая 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.

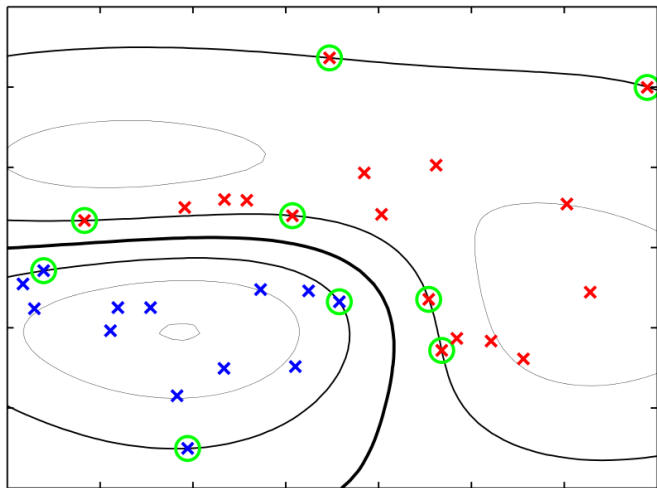
- Нормальное распределение (radial basis function):

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

- Двухуровневая нейронная сеть:

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

где  $o$  — СИГМОИД.



- 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

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

- Другой вариант для неразделимых данных –  $\nu$ -SVM [Schölkopf et al., 2000].
- Максимизируем

$$L(\mathbf{a}) = -\frac{1}{2} \sum_n \sum_m a_n a_m t_n t_m k(\mathbf{x}_n, \mathbf{x}_m)$$

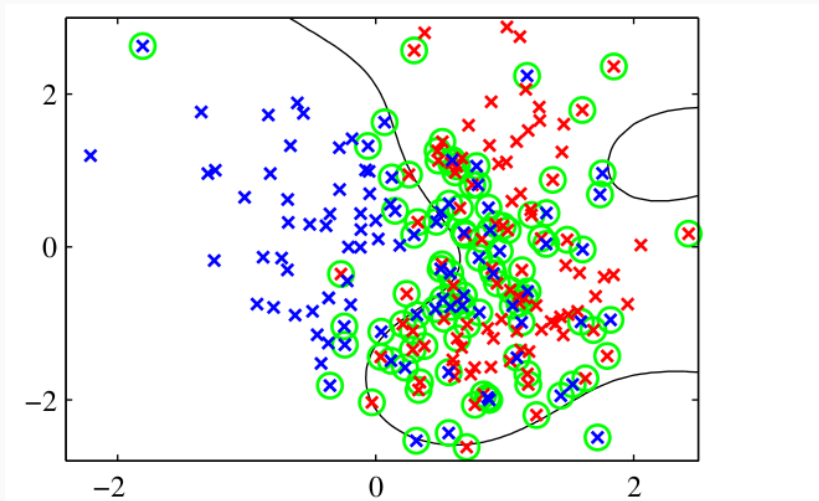
с ограничениями

$$0 \leq a_n \leq \frac{1}{N}, \quad \sum_n a_n t_n = 0, \quad \sum_n a_n \geq \nu.$$

- Параметр  $\nu$  можно интерпретировать как верхнюю границу на долю ошибок.



## SVM для КЛАССИФИКАЦИИ



- В случае SVM с возможными ошибками мы минимизируем

$$C \sum_{n=1}^N \xi_n + \frac{1}{2} \|\mathbf{w}\|^2.$$

- Для точек с правильной стороны  $\xi_n = 0$ , с неправильной –  $\xi_n = 1 - y_n t_n$ .
- Так что можно записать *hinge error function*  $E_{SV}(y_n t_n) = [1 - y_n t_n]_+$  и переписать как задачу с регуляризацией

$$\sum_{n=1}^N E_{SV}(y_n t_n) + \lambda \|\mathbf{w}\|^2.$$

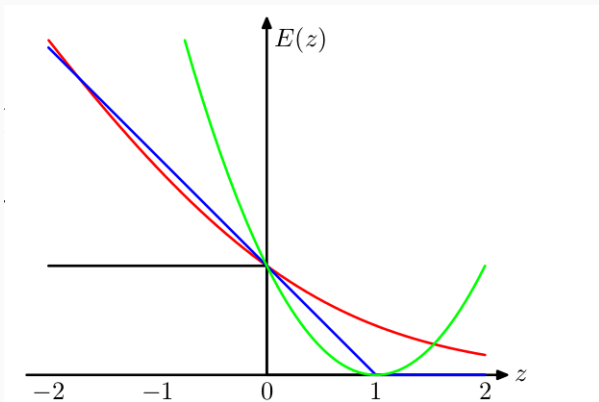
- Вспомним логистическую регрессию и переформулируем её для целевой переменной  $t \in \{-1, 1\}$ :  $p(t = 1 | y) = \sigma(y)$ , значит,  $p(t = -1 | y) = 1 - \sigma(y) = \sigma(-y)$ , и  $p(t | y) = \sigma(yt)$ .
- И логистическая регрессия – это минимизация

$$\sum_{n=1}^N E_{LR}(y_n t_n) + \lambda \|\mathbf{w}\|^2,$$

где  $E_{LR}(y_n t_n) = \ln(1 + e^{-yt})$ .

## СВЯЗЬ С ЛОГИСТИЧЕСКОЙ РЕГРЕССИЕЙ

- График hinge error function, вместе с функцией ошибки для логистической регрессии:



- Как обобщить SVM на несколько классов? Варианты (без подробностей):
  1. обучить одну против всех и классифицировать  $y(\mathbf{x}) = \max_k y_k(\mathbf{x})$  (нехорошо, потому что задача становится несбалансированной, и  $y_k(\mathbf{x})$  на самом деле несравнимы);
  2. можно сформулировать единую функцию для всех  $K$  SVM одновременно, но обучение становится гораздо медленнее;
  3. можно обучить попарно  $K(K - 1)/2$  классификаторов, а потом считать голоса – кто победит;
  4. DAGSVM: организуем попарные классификаторы в граф и будем идти по графу, для классификации выбирая очередной вопрос;
  5. есть даже методы, основанные на кодах, исправляющих ошибки.

- SVM также можно использовать с *одним* классом.
- Как и зачем?

- SVM также можно использовать с *одним* классом.
- Как и зачем?
- Можно при помощи SVM очертить границу области высокой плотности.
- Тем самым найдём выбросы данных (outliers).
- Задача будет такая: найти наименьшую поверхность (сферу, например), которая содержит все точки, кроме доли  $\nu$ .

- SVM можно использовать для регрессии, сохраняя свойство разреженности (т.е. то, что SVM зависит только от опорных векторов).
- В обычной линейной регрессии мы минимизировали

$$\frac{1}{2} \sum_{n=1}^N (y_n - t_n)^2 + \frac{\lambda}{2} \|\mathbf{w}\|^2.$$

- В SVM мы сделаем так: если мы попадаем в  $\epsilon$ -окрестность предсказания, то ошибки, будем считать, совсем нет.



- $\epsilon$ -insensitive error function:

$$E_{\epsilon}(y(\mathbf{x}) - t) = \begin{cases} 0, & |y(\mathbf{x}) - t| < \epsilon, \\ |y(\mathbf{x}) - t| - \epsilon & \text{иначе.} \end{cases}$$

- И задача теперь выглядит как минимизация

$$C \sum_{n=1}^N E_{\epsilon}(y(\mathbf{x}_n) - t_n) + \frac{\lambda}{2} \|\mathbf{w}\|^2.$$

- Чтобы переформулировать, нужны по две slack переменные, для обеих сторон «трубки»:

$$y(\mathbf{x}_n) - \epsilon \leq t_n \leq y(\mathbf{x}_n) + \epsilon$$

превращается в

$$t_n \leq y(\mathbf{x}_n) + \epsilon + \xi_n,$$

$$t_n \geq y(\mathbf{x}_n) - \epsilon - \hat{\xi}_n,$$

и мы оптимизируем

$$C \sum_{n=1}^N E_{\epsilon} (\xi_n + \hat{\xi}_n) + \frac{\lambda}{2} \|\mathbf{w}\|^2.$$

- Если же теперь пересчитать дуальную задачу, то получится

$$L(\mathbf{a}, \hat{\mathbf{a}}) = -\frac{1}{2} \sum_n \sum_m (a_n - \hat{a}_n) (a_m - \hat{a}_m) k(\mathbf{x}_n, \mathbf{x}_m) - \\ - \epsilon \sum_{n=1}^n (a_n + \hat{a}_n) + \sum_{n=1}^N (a_n - \hat{a}_n) t_n,$$

и мы её минимизируем по  $a_n, \hat{a}_n$  с условиями

$$0 \leq a_n \leq C,$$

$$0 \leq \hat{a}_n \leq C,$$

$$\sum_{n=1}^N (a_n - \hat{a}_n) = 0.$$

- Когда решим эту задачу, сможем предсказывать новые значения как

$$y(\mathbf{x}) = \sum_{n=1}^N (a_n - \hat{a}_n) k(\mathbf{x}, \mathbf{x}_n) + b,$$

где  $b$  можно найти как

$$\begin{aligned} b = t_n - \epsilon - \mathbf{w}^\top \phi(\mathbf{x}_n) &= \\ &= t_n - \epsilon - \sum_{m=1}^N (a_m - \hat{a}_m) k(\mathbf{x}_n, \mathbf{x}_m). \end{aligned}$$

- А условия ККТ превращаются в

$$\begin{aligned}a_n (\epsilon + \xi_n + y(\mathbf{x}_n) - t_n) &= 0, \\ \hat{a}_n (\epsilon + \hat{\xi}_n - y(\mathbf{x}_n) + t_n) &= 0, \\ (C - a_n)\xi_n &= 0, \\ (C - \hat{a}_n)\hat{\xi}_n &= 0.\end{aligned}$$

- Отсюда очевидно, что либо  $a_n$ , либо  $\hat{a}_n$  всегда равны 0, и хотя бы один из них не равен, только если точка лежит на или за границей «трубки».
- Опять получили решение, зависящее только от «опорных векторов».

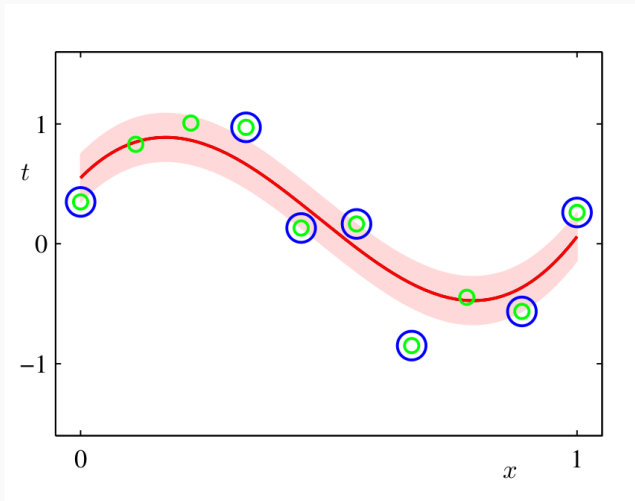
- Но снова можно переформулировать в виде  $\nu$ -SVM, в котором параметр более интуитивно ясен: вместо ширины трубки  $\epsilon$  рассмотрим  $\nu$  – долю точек, лежащих вне трубки; тогда минимизировать надо

$$L(\mathbf{a}) = -\frac{1}{2} \sum_n \sum_m (a_n - \hat{a}_n) (a_m - \hat{a}_m) k(\mathbf{x}_n, \mathbf{x}_m) + \sum_{n=1}^N (a_n - \hat{a}_n) t_n$$

при условиях

$$\begin{aligned} 0 \leq a_n \leq \frac{C}{N}, & \quad \sum_{n=1}^N (a_n - \hat{a}_n) = 0, \\ 0 \leq \hat{a}_n \leq \frac{C}{N}, & \quad \sum_{n=1}^N (a_n + \hat{a}_n) \leq \nu C. \end{aligned}$$

# SVM для РЕГРЕССИИ



- На практике:
  - маленький  $C$  – гладкая разделяющая поверхность, мало опорных векторов;
  - большой  $C$  – сложная разделяющая поверхность, много опорных векторов.
- Для RBF ядра:
  - маленькое  $\gamma$  – опорные векторы влияют далеко, модель более простая;
  - большое  $\gamma$  – опорные векторы влияют только непосредственно рядом, модель более сложная.



# ОБЪЕДИНЕНИЕ МОДЕЛЕЙ

---

- До сих пор мы разрабатывали (и потом ещё будем разрабатывать) модели, которые делают предсказания (в основном для задач регрессии и классификации).
- Таким образом, мы можем попробовать обучить сразу много разных моделей!
- Model selection – это о том, как выбрать из них лучшую.
- Но, может быть, можно не выбирать, а использовать все сразу?

- Комитет: обучаем  $L$  разных моделей, а потом так или иначе усредняем-комбинируем их результаты.
- Альтернатива: обучаем  $L$  разных моделей, а потом обучаем отдельную модель о том, какую из них использовать для предсказания (например, дерево принятия решений).

- Начнём с самого простого – байесовского усреднения.
- Мы уже знаем, что такое комбинация моделей – например, линейная смесь гауссианов:

$$p(\mathbf{x}) = \sum_k \pi_k \mathcal{N}(\mathbf{x} \mid \mu_k, \Sigma_k).$$

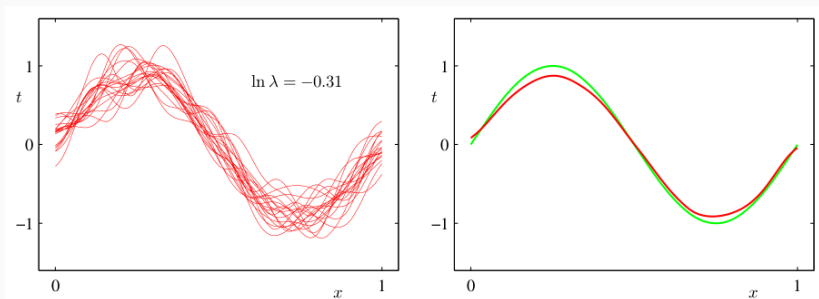
- Если её обучать, мы обучим коэффициенты смеси, и результат будет порождён как бы двухуровневым процессом.

- А байесовское усреднение будет выглядеть как

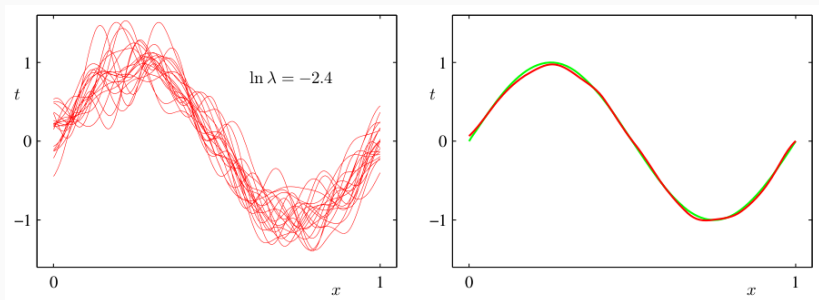
$$p(\mathbf{X}) = \sum_{h=1}^H p(\mathbf{X} | h)p(h).$$

- Смысл теперь в том, что генерирует  $\mathbf{X}$  только одна модель, но мы просто не знаем какая именно; когда  $\mathbf{X}$ , апостериорные распределения  $p(h | \mathbf{X})$  сужаются, и мы выбираем то, что надо.
- Но метод очень простой: взять много моделей и усреднить.
- Где-то мы это уже видели...

# ГДЕ-ТО МЫ ЭТО УЖЕ ВИДЕЛИ



# ГДЕ-ТО МЫ ЭТО УЖЕ ВИДЕЛИ



- На этих картинках – модели с высоким bias, которые обучены по разным датасетам, сгенерированным одним и тем же распределением.
- И если их усреднить, получится как раз то, что надо.
- Но в жизни у нас нет возможности генерировать много датасетов: сколько данных есть, столько есть.
- Просто разбивать датасет на части – не поможет. Что делать?



- Пусть у нас есть датасет  $\mathbf{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_N\}$ .
- Давайте сгенерируем много датасетов так: будем выбирать из  $\mathbf{X}$   $N$  точек с замещением, т.е. в новом датасете некоторые точки будут повторяться.
- Этот метод называется *bootstrapping*.

- Мы сделаем так  $M$  датасетов размера  $N$  (с повторяющимися точками), потом обучим  $M$  моделей, а потом образуем из них комитет и будем предсказывать как

$$y(\mathbf{x}) = \frac{1}{M} \sum_{m=1}^M y_m(\mathbf{x}).$$

- Это называется *bagging* (bootstrap aggregation).
- На первый взгляд кажется, что это какая-то ерунда: мы пытаемся получить что-то из ничего...

- Пусть настоящая функция, которую мы пытаемся предсказать –  $h(\mathbf{x})$ , т.е. модели наши выглядят как

$$y_m(\mathbf{x}) = h(\mathbf{x}) + \epsilon_m(\mathbf{x}).$$

- Тогда средняя ошибка модели – это

$$E_{\mathbf{x}} [(y_m(\mathbf{x}) - h(\mathbf{x}))^2] = E_{\mathbf{x}} [\epsilon_m(\mathbf{x})^2].$$

- И средняя ошибка тех моделей, которые мы обучаем, получается

$$E_{\text{avg}} = \frac{1}{M} \sum_{m=1}^M E_{\mathbf{x}} [\epsilon_m(\mathbf{x})^2].$$

- И средняя ошибка тех моделей, которые мы обучаем, получается

$$E_{\text{avg}} = \frac{1}{M} \sum_{m=1}^M E_{\mathbf{x}} [\epsilon_m(\mathbf{x})^2].$$

- А ошибка комитета – это

$$\begin{aligned} E_{\text{com}} &= E_{\mathbf{x}} \left[ \left( \frac{1}{M} \sum_{m=1}^M (y_m(\mathbf{x}) - h(\mathbf{x})) \right)^2 \right] = \\ &= E_{\mathbf{x}} \left[ \left( \frac{1}{M} \sum_{m=1}^M \epsilon_m(\mathbf{x}) \right)^2 \right]. \end{aligned}$$

- $E_{\text{avg}} = \frac{1}{M} \sum_{m=1}^M E_{\mathbf{x}} [\epsilon_m(\mathbf{x})^2]$ ,  $E_{\text{com}} = E_{\mathbf{x}} \left[ \left( \frac{1}{M} \sum_{m=1}^M \epsilon_m(\mathbf{x}) \right)^2 \right]$ .
- Если предположить, что  $E_{\mathbf{x}} [\epsilon_m(\mathbf{x})] = 0$ , и ошибки некоррелированы:  $E_{\mathbf{x}} [\epsilon_m(\mathbf{x})\epsilon_l(\mathbf{x})] = 0$ , мы получим

$$E_{\text{com}} = \frac{1}{M} E_{\text{avg}}.$$

- $E_{\text{com}} = \frac{1}{M} E_{\text{avg}}!$
- Это кажется совершенно невероятным. На самом деле всё не так хорошо – конечно, ошибки на самом деле сильно коррелированы.
- И, конечно, на самом деле обычно уменьшение ошибки не такое большое.
- Но можно показать, что в любом случае  $E_{\text{com}} \leq E_{\text{avg}}$ , так что хуже от этого не будет, а лучше стать может.

## БУСТИНГ: ADA BOOST

---

- Следующая идея объединения моделей: предположим, что у нас есть возможность обучать какую-нибудь простую модель (weak learner) на подмножестве данных.
- Тогда можно делать так: обучили модель, посмотрели, где она хорошо работает, обучили следующую модель на том подмножестве, где она работает плохо, повторили.
- Этот метод называется *бустинг* (boosting).



- AdaBoost: самый простой вариант. Рассмотрим задачу бинарной классификации; данные – это  $\mathbf{x}_1, \dots, \mathbf{x}_N$  с ответами  $t_1, \dots, t_N, t_i \in \{-1, 1\}$ .
- Снабдим каждый тестовый пример весом  $w_i$ ; изначально положим  $w_i = \frac{1}{N}$ .
- Предположим, что у нас есть процедура, которая обучает некоторый классификатор, выдающий  $y(\mathbf{x}) \in \{-1, 1\}$ , на взвешенных данных (минимизируя взвешенную ошибку).

- Тогда в алгоритме AdaBoost мы инициализируем  $w_n^{(1)} := 1/N$ , а потом для  $m = 1..M$ :

1. обучаем классификатор  $y_m(\mathbf{x})$ , который минимизирует функцию ошибки

$$J_m = \sum_{n=1}^N w_n^{(m)} [y_m(\mathbf{x}_n) \neq t_n];$$

2. вычисляем

$$\epsilon_m = \frac{\sum_{n=1}^N w_n^{(m)} [y_m(\mathbf{x}_n) \neq t_n]}{\sum_{n=1}^N w_n^{(m)}}, \quad \alpha_m = \ln \left( \frac{1 - \epsilon_m}{\epsilon_m} \right);$$

3. пересчитываем новые веса

$$w_n^{(m+1)} = w_n^{(m)} e^{\alpha_m [y_m(\mathbf{x}_n) \neq t_n]}.$$

- После обучения предсказываем как

$$Y_M(\mathbf{x}) = \text{sign} \left( \sum_{m=1}^M \alpha_m y_m(\mathbf{x}) \right).$$

- Смысл именно такой, как мы говорили: сначала тренируем абстрактно лучший классификатор. Потом увеличиваем веса неправильно классифицированным примерам, обучаем новый классификатор, и т.д.

- Изначально, когда AdaBoost придумали [Freund, Shapire, 1997], мотивация была такая: предположим, что ошибка каждого слабого классификатора  $h_t$  не превышает  $\epsilon_t = \frac{1}{2} - \gamma_t$ .
- Тогда можно показать, что окончательная ошибка не превосходит

$$\prod_t (2\sqrt{\epsilon_t(1 - \epsilon_t)}) = \prod_t \sqrt{1 - 4\gamma_t^2} \leq e^{-2\sum_t \gamma_t^2}.$$

- Однако на самом деле гарантий на  $\gamma_t$  обычно нету, и практические результаты AdaBoost лучше, чем можно было бы ожидать из этой оценки.

- Основная идея [Friedman et al., 2000]: давайте определим экспоненциальную ошибку

$$E = \sum_{n=1}^N e^{-t_n f_m(\mathbf{x}_n)},$$

где  $f_m$  – линейная комбинация базовых классификаторов:

$$f_m(\mathbf{x}) = \frac{1}{2} \sum_{l=1}^m \alpha_l y_l(\mathbf{x}).$$

- Мы хотим минимизировать  $E$  по  $\alpha_l$  и параметрам  $y_l(\mathbf{x})$ .

- Минимизируем  $E = \sum_{n=1}^N e^{-t_n f_m(\mathbf{x}_n)}$ .
- Вместо глобальной оптимизации будем действовать жадно: пусть  $y_1(\mathbf{x}), \dots, y_{m-1}(\mathbf{x})$  и  $\alpha_1, \dots, \alpha_{m-1}$  уже зафиксированы. Тогда ошибка получается

$$E = \sum_{n=1}^N e^{-t_n f_{m-1}(\mathbf{x}_n) - \frac{1}{2} t_n \alpha_m y_m(\mathbf{x})} = \sum_{n=1}^N w_n^{(m)} e^{-\frac{1}{2} t_n \alpha_m y_m(\mathbf{x})},$$

где  $w_N^{(m)} = e^{-t_n f_{m-1}(\mathbf{x}_n)}$  – это как раз и есть наши веса, и их теперь можно считать константами.

- На правильных классификациях произведение  $-1$ , на неправильных  $+1$ :

$$\begin{aligned} E &= e^{-\frac{\alpha_m}{2}} \sum_{\text{correct}} w_n^{(m)} + e^{\frac{\alpha_m}{2}} \sum_{\text{wrong}} w_n^{(m)} = \\ &= \left( e^{\frac{\alpha_m}{2}} - e^{-\frac{\alpha_m}{2}} \right) \sum_{n=1}^N w_n^{(m)} [y_m(\mathbf{x}_n) \neq t_n] + e^{-\frac{\alpha_m}{2}} \sum_{n=1}^N w_n^{(m)}, \end{aligned}$$

и достаточно минимизировать  $J_m = \sum_{n=1}^N w_n^{(m)} [y_m(\mathbf{x}_n) \neq t_n]$ .

- Ну а когда мы обучим  $y_m(\mathbf{x})$ , из  $E = \sum_{n=1}^N w_n^{(m)} e^{-\frac{1}{2}t_n \alpha_m y_m(\mathbf{x})}$  получится

$$w_n^{(m+1)} = w_n^{(m)} e^{-\frac{1}{2}t_n \alpha_m y_m(\mathbf{x}_n)} = w_n^{(m)} e^{-\frac{1}{2}\alpha_m} e^{\alpha_m [y_m(\mathbf{x}_n) \neq t_n]},$$

и на  $e^{-\frac{1}{2}\alpha_m}$  можно все веса сократить.

- Таким образом, бустинг можно рассматривать как оптимизацию экспоненциальной ошибки.



Спасибо за внимание!