

Как объединять модели

Сергей Николенко

Академический Университет, 2012

Outline

- 1 Простые методы
 - Усреднение
 - Bagging
- 2 Бустинг
 - AdaBoost
 - Расширения: RankBoost

Как объединять модели

- 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?

Основные подходы

- 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).

Комбинация моделей и байесовское усреднение

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

$$p(\mathbf{x}) = \sum_k \pi_k \mathcal{N}(\mathbf{x} \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k).$$

- Если её обучать, мы обучим коэффициенты смеси, и результат будет порождён

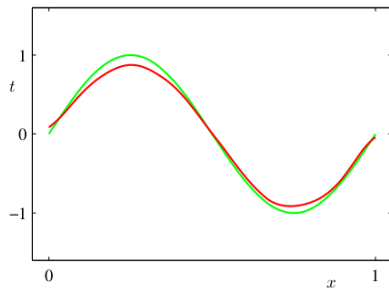
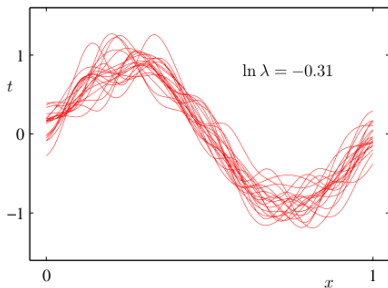
Комбинация моделей и байесовское усреднение

- А байесовское усреднение будет выглядеть как

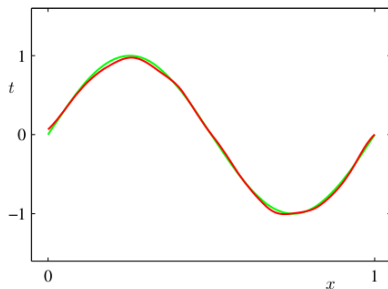
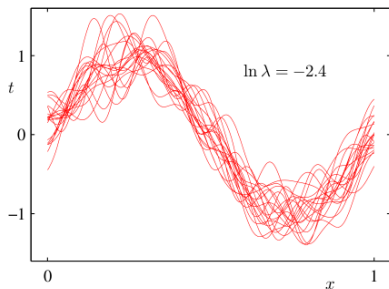
$$p(\mathbf{X}) = \sum_{h=1}^H p(\mathbf{X} | h)p(h).$$

- 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



Bagging

- 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?

Bagging

- 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*.

Bagging

- 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...

Bagging

- 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

$$\mathbb{E}_{\mathbf{x}} \left[(y_m(\mathbf{x}) - h(\mathbf{x}))^2 \right] = \mathbb{E}_{\mathbf{x}} \left[\epsilon_m(\mathbf{x})^2 \right].$$

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

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

Bagging

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

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

- А ошибка комитета – это

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

Bagging

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

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

Bagging

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

Outline

- 1 Простые методы
 - Усреднение
 - Bagging
- 2 Бустинг
 - AdaBoost
 - Расширения: RankBoost

AdaBoost

- 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

- AdaBoost: самый простой вариант. Рассмотрим задачу бинарной классификации; данные – это x_1, \dots, 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 $y(x) \in \{-1, 1\}$, на *взвешенных* данных (минимизируя взвешенную ошибку).

AdaBoost

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

AdaBoost

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

Weak learners: деревья принятия решений

- А что за weak learners применяются в реальных приложениях?
- Обычно бустинг применяется, когда есть набор уже посчитанных фич (посчитанных из каких-то более сложных моделей), и нужно объединить их в единую модель.
- Часто слабые классификаторы совершенно тупые: берём одну координату и ищем по ней оптимальное разбиение.
- Могут быть чуть посложнее: деревья принятия решений.

Weak learners: деревья принятия решений

- Дерево принятия решений — это дерево. На нём есть метки:
 - в узлах, не являющиеся листьями: атрибуты (фичи), по которым различаются случаи;
 - в листьях: значения целевой функции;
 - на рёбрах: значения атрибута, из которого исходит ребро.
- Чтобы классифицировать новый случай, нужно спуститься по дереву до листа и выдать соответствующее значение.

Weak learners: деревья принятия решений

- Конечно, перебрать все деревья нельзя, их строят жадно.
 - 1 Выбираем очередной атрибут Q , помещаем его в корень.
 - 2 Выбираем оптимальное разбиение по атрибуту. Для всех интервалов разбиения:
 - оставляем из тестовых примеров только те, у которых значение атрибута Q попало в этот интервал;
 - рекурсивно строим дерево в этом потомке.
- Остались три вопроса:
 - 1 как проводить разбиение?
 - 2 как выбирать новый атрибут?
 - 3 когда останавливаться?

Weak learners: деревья принятия решений

- Если атрибут бинарный или дискретный с небольшим числом значений, разбиение тривиальное (просто по значениям).
- Если непрерывный – можно брать просто среднее арифметическое (тем самым минимизируя сумму квадратов).
- Выбирают атрибут, оптимизируя целевую функцию. Для задачи регрессии просто минимизируем среднеквадратическую ошибку по отношению к текущему предсказателю

$$y_{\tau} = \frac{1}{|\mathbf{X}_{\tau}|} \sum_{\mathbf{x}_n \in \mathbf{X}_{\tau}} t_n.$$

Weak learners: деревья принятия решений

- Предположим, что мы решаем задачу классификации на K классов.
- Тогда «сложность» подмножества данных \mathbf{X}_τ относительно целевой функции $f(\mathbf{x}) : \mathbf{X} \rightarrow \{1, \dots, K\}$ характеризуется *перекрёстной энтропией*:

$$Q(\mathbf{X}_\tau) = \sum_{k=1}^K p_{\tau,k} \ln p_{\tau,k}.$$

- Иногда ещё используют *индекс Джини* (Gini index):
 $G(\mathbf{X}_\tau) = \sum_{k=1}^K p_{\tau,k} (1 - p_{\tau,k}).$

Weak learners: деревья принятия решений

- Когда останавливаться? Недоучиться плохо и переучиться плохо.
- Останавливаться, когда ошибка перестанет меняться, тоже плохо (она может опять начать меняться ниже).
- Поэтому делают так: выращивают большое дерево, чтобы наверняка, а потом *обрезают* его (pruning): поддереву t схлопывают в корень, а правило предсказания в корне считают как $y_t = \frac{1}{|\mathbf{X}_t|} \sum_{\mathbf{x}_n \in \mathbf{X}_t} t_n$.
- Обрезают, оптимизируя функцию ошибки с регуляризатором: $\sum_{\tau=1}^{|T|} Q(\mathbf{X}_\tau) + \lambda |T|$ (для классификаторов здесь можно использовать долю ошибок классификации).

Теоретические свойства AdaBoost

- 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

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

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

Теоретические свойства AdaBoost

- 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 α_l и параметрам $y_l(\mathbf{x})$.

Теоретические свойства AdaBoost

- 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.

Теоретические свойства AdaBoost

- 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

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

Теоретические свойства AdaBoost

- Ну а когда мы обучим $y_m(\mathbf{x})$, из

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

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

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

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

RankBoost

- Предположим, что нам нужно не классифицировать, а *упорядочить* элементы какого-то множества.
- Задача *ранжирования*: например, мы – поисковая система, и мы хотим ранжировать выдачу по какому-то запросу от более релевантных к менее.
- У нас может быть масса разных фич, характеризующих эти документы.
- Это тоже можно сделать бустингом.

RankBoost

- Формально говоря, нам надо обучить функцию $F(\mathbf{x}) : \mathbb{R}^n \rightarrow \mathbb{R}$ по входу $\{\mathbf{x}_1, \dots, \mathbf{x}_N\}$ и функции частичных предпочтений $\Phi : \mathbf{X} \times \mathbf{X} \rightarrow \mathbb{R}$ ($\Phi(\mathbf{x}_i, \mathbf{x}_j) > 0$, если \mathbf{x}_i лучше \mathbf{x}_j , и т.д.).
- Обычно в качестве функции Φ подаётся просто разбиение на «хорошие» (например, релевантные) и «плохие»: $\Phi(\mathbf{x}, \mathbf{y}) = 1$ для хорошего \mathbf{x} и плохого \mathbf{y} и 0, если они из одного множества.

RankBoost

- RankBoost по сути работает примерно как AdaBoost, но раньше веса давали распределение на примерах, а теперь – на парах примеров: инициализируем распределение $D^{(1)} = D(\mathbf{x}, \mathbf{y})$ на $\mathbf{X} \times \mathbf{X}$, а потом для $m = 1..M$:
 - обучаем слабую ранжирующую функцию $h_m(\mathbf{x}) : \mathbf{X} \rightarrow \mathbb{R}$ по распределению $D^{(m)}$;
 - выбираем $\alpha_m \in \mathbb{R}$ (потом скажу как);
 - пересчитываем новое распределение

$$D^{(m+1)}(\mathbf{x}, \mathbf{y}) = \frac{1}{Z_m} D^{(m)}(\mathbf{x}, \mathbf{y}) e^{\alpha_m (h_m(\mathbf{x}) - h_m(\mathbf{y}))}.$$

- После обучения ранжируем как $H_M(\mathbf{x}) = \sum_{m=1}^M \alpha_m h_m(\mathbf{x})$.

RankBoost

- Тогда получится такая теорема: если вернуться от $D^{(m+1)}(\mathbf{x}, \mathbf{y})$ к $D(\mathbf{x}, \mathbf{y})$, будет

$$D^{(m+1)}(\mathbf{x}, \mathbf{y}) = \frac{1}{\prod_m Z_m} D(\mathbf{x}, \mathbf{y}) e^{H_M(\mathbf{x}) - H_M(\mathbf{y})}.$$

- Значит, ошибку можно оценить как

$$\begin{aligned} J_M &= \sum_{\mathbf{x}, \mathbf{y}} D(\mathbf{x}, \mathbf{y}) [H_M(\mathbf{x}) \geq H_M(\mathbf{y})] \leq \\ &\leq \sum_{\mathbf{x}, \mathbf{y}} D(\mathbf{x}, \mathbf{y}) e^{H_M(\mathbf{x}) - H_M(\mathbf{y})} = \sum_{\mathbf{x}, \mathbf{y}} D^{(m+1)}(\mathbf{x}, \mathbf{y}) \prod_m Z_m = \prod_m Z_m. \end{aligned}$$

RankBoost

- И выбирать α_m можно (и нужно) так, чтобы минимизировать $\prod_m Z_m$, т.е. на шаге m минимизировать

$$Z_m = \sum_{\mathbf{x}, \mathbf{y}} D^{(m)}(\mathbf{x}, \mathbf{y}) e^{\alpha_m (h_m(\mathbf{x}) - h_m(\mathbf{y}))}.$$

- Формально для нас h_m – чёрный ящик, но на практике мы часто выбираем алгоритм обучения и для h_m , так что его тоже можно выбирать так, чтобы минимизировать Z_m .

RankBoost

- Теорема: для любого слабого ранжирования h $Z(\alpha)$ имеет единственный минимум, так что можно просто бинарным поиском.
- Если $h \in \{0, 1\}$, можно и аналитически: обозначим $W_b = \sum_{\mathbf{x}, \mathbf{y}} D(\mathbf{x}, \mathbf{y}) [h(\mathbf{x}) - h(\mathbf{y}) = b]$. Тогда

$$\alpha_{\text{opt}} = \frac{1}{2} \ln \left(\frac{W_{-1}}{W_{+1}} \right), \quad Z = W_0 + 2\sqrt{W_{-1}W_{+1}}.$$

Упражнение. Проверьте это.

RankBoost

- А для $h \in [0, 1]$ можно приблизить:
 $e^{\alpha x} \leq \left(\frac{1+x}{2}\right) e^{\alpha} + \left(\frac{1-x}{2}\right) e^{-\alpha}$, $x \in [-1, 1]$, так что

$$Z \leq \left(\frac{1-r}{2}\right) e^{\alpha} + \left(\frac{1+r}{2}\right) e^{-\alpha}, \quad r = \sum_{\mathbf{x}, \mathbf{y}} D(\mathbf{x}, \mathbf{y}) (h(\mathbf{x}) - h(\mathbf{y})),$$

и можно выбирать

$$\alpha_{\text{opt}} = \frac{1}{2} \ln \left(\frac{1+r}{1-r} \right),$$

а h можно обучать так, чтобы максимизировать $|r|$.

Упражнение. Проверьте это.

Оценки доверия

- Предположим, что наши слабые классификаторы выдают не только один бит, а ещё какую-то оценку доверия своему результату (обычно ведь так и бывает).
- Т.е. слабый классификатор – это слабая гипотеза $h(\mathbf{x}) : \mathbf{X} \rightarrow \mathbb{R}$.

Оценки доверия

- Для этого слегка обобщим исходный AdaBoost, представив его как алгоритм, пересчитывающий распределения:
 - 1 обучаем слабую гипотезу $h_m(\mathbf{x}) : \mathbf{X} \rightarrow \mathbb{R}$ по распределению $D^{(m)}$;
 - 2 выбираем $\alpha_m \in \mathbb{R}$ (потом скажу как);
 - 3 пересчитываем новое распределение

$$D^{(m+1)}(\mathbf{x}) = \frac{1}{Z_m} D^{(m)}(\mathbf{x}) e^{-\alpha_m t_m h_m(\mathbf{x})}.$$

- 4 После обучения классифицируем как $H_M(\mathbf{x}) = \text{sign} \left(\sum_{m=1}^M \alpha_m h_m(\mathbf{x}) \right)$.

Оценки доверия

- Теперь будет верна аналогичная теорема:

$$\frac{1}{N} |\{n : H(\mathbf{x}_n) \neq t_n\}| \leq \prod_{m=1}^M Z_m.$$

Упражнение. Докажите это.

- И, значит, мы тоже можем минимизировать Z_m жадно на каждом шаге по α_m и слабой гипотезе h_m .

Оценки доверия

- В частности, исходный AdaBoost получится как одно из приближений к такой минимизации; обозначим $u_n = t_n y(\mathbf{x}_n)$:

$$Z = \sum_n D(\mathbf{x}_n) e^{-\alpha u_n} \leq \sum_n D(\mathbf{x}_n) \left(\frac{1+u_i}{2} e^{-\alpha} + \frac{1-u_i}{2} e^{\alpha} \right),$$

причём эта оценка точна для $u_n \in \{-1, 1\}$ (т.е. для исходного AdaBoost).

- Тогда для $r = \sum_n D(\mathbf{x}_n) u_n$ мы получим аналитический минимум

$$\alpha_{\text{opt}} = \frac{1}{2} \ln \left(\frac{1+r}{1-r} \right), \quad Z \leq \sqrt{1-r^2}.$$

- И мы доказали исходную теорему про AdaBoost.

Оценки доверия

- А для общего случая получится

$$Z = \sum_n D(\mathbf{x}_n) e^{-\alpha u_n},$$

$$Z'_\alpha = - \sum_n D(\mathbf{x}_n) u_n e^{-\alpha u_n} = -Z \sum_n D^{(m+1)}(\mathbf{x}_n) e^{-\alpha u_n}.$$

- Т.е. надо выбирать такой α , чтобы $Z'_\alpha = 0$, а значит,

$$\sum_n D^{(m+1)}(\mathbf{x}_n) u_n = \mathbb{E}_{\mathbf{x} \sim D^{(m+1)}} [t_n y_m(\mathbf{x}_n)] = 0,$$

т.е. выбирать так, чтобы по $D^{(m+1)}$ гипотеза h_m была бы некоррелирована с метками t_n .

- Это можно делать, численно решая уравнение $Z'_\alpha = 0$; у него не больше одного корня (проверьте, что $Z''_\alpha > 0$ для всех α).

Thank you!

Спасибо за внимание!