Примеры вероятностных моделей

Сергей Николенко

Computer Science Club, Казань, 2014

Outline

1. Категоризация текстов: naive Bayes и LDA
   - Naive Bayes
   - Latent Dirichlet allocation
2. Рекомендательные системы
   - Ближайшие соседи
   - Вероятностные модели
3. Ещё примеры
   - Байесовские рейтинг-системы
   - Модели поведения пользователей

Категоризация текстов

- Классическая задача машинного обучения и information retrieval – категоризация текстов.
- Дан набор текстов, разделённый на категории. Нужно обучить модель и потом уметь категоризовать новые тексты.
- Атрибуты $a_1, a_2, \ldots, a_n$ – это слова, v – тема текста (или атрибут вроде «спам / не спам»).
- Bag-of-words model: забываем про порядок слов, составляем словарь. Теперь документ – это вектор, показывающий, сколько раз каждое слово из словаря в нём встречается.

Naive Bayes

- Заметим, что даже это – сильно упрощённый взгляд: для слов ещё довольно-таки важен порядок, в котором они идут...
- Но и это ещё не всё: получается, что $p(a_1, a_2, \ldots, a_n | x = v)$ – это вероятность *в точности такого набора слов* в сообщениях на разные темы. Очевидно, такой статистики взять неоткуда.
- Значит, надо дальше делать упрощающие предположения.
- Наивный байесовский классификатор – самая простая такая модель: давайте предположим, что все слова в словаре условно независимы при условии данной категории.

Naive Bayes

- Иначе говоря:

$$p(a_1, a_2, \ldots, a_n | x = v) = p(a_1 | x = v) p(a_2 | x = v) \ldots p(a_n | x = v).$$

- Итак, наивный байесовский классификатор выбирает v как

$$v_{NB}(a_1, a_2, \ldots, a_n) = \arg\max_{v \in V} p(x = v) \prod_{i=1}^{n} p(a_i | x = v).$$

- В парадигме классификации текстов мы предполагаем, что разные слова в тексте на одну и ту же тему появляются независимо друг от друга. Однако, несмотря на такие бредовые предположения, naive Bayes на практике работает очень даже неплохо (и этому есть разумные объяснения).

Naive Bayes

- Формально говоря, у наивного Байеса две разновидности:
  - многомерная (multivariate) модель: документ – это вектор бинарных атрибутов, показывающих, встретилось ли в документе то или иное слово; правдоподобие документа – произведение вероятностей того, что встретилось каждое слово из документа и того, что не встретились остальные (словарные) слова;
  - мультиномиальная (multinomial) модель: документ – это последовательность событий, случайных выборов одного слова из того самого «bag of words»; правдоподобие – произведение вероятностей того, что мы достали из мешка слова, которые встретились в документе.

LDA

- Более сложная модель – LDA (Latent Dirichlet Allocation).
- Задача: смоделировать большую коллекцию текстов (например, для information retrieval или классификации).
- Мы знаем наивный подход: скрытая переменная – тема, слова получаются из темы независимо по дискретному распределению.
- Аналогично работают и подходы, основанные на кластеризации.
- Давайте чуть усложним.

LDA

- Очевидно, что у одного документа может быть несколько тем; подходы, которые кластеризуют документы по темам, никак этого не учитывают.
- Давайте построим иерархическую байесовскую модель:
  - на первом уровне – смесь, компоненты которой соответствуют «темам»;
  - на втором уровне – мультиномиальная переменная с априорным распределением Дирихле, которое задаёт «распределение тем» в документе.

LDA

- Если формально: слова берутся из словаря $\{1, \ldots, V\}$; слово – это вектор w, $w_i \in \{0, 1\}$, где ровно одна компонента равна 1.
- Документ – последовательность из N слов $\mathbf{w}$. Нам дан корпус из M документов $\mathcal{D} = \{\mathbf{w}_d \mid d = 1..M\}$.
- Генеративная модель LDA выглядит так.
  1. Выбрать $N \sim p(N \mid \xi)$.
  2. Выбрать $\theta \sim \mathrm{Di}(\alpha)$.
  3. Для каждого из N слов w_n:
     1. выбрать тему $z_n \sim \mathrm{Mult}(\theta)$;
     2. выбрать слово $w_n \sim p(w_n \mid z_n, \beta)$ по мультиномиальному распределению.

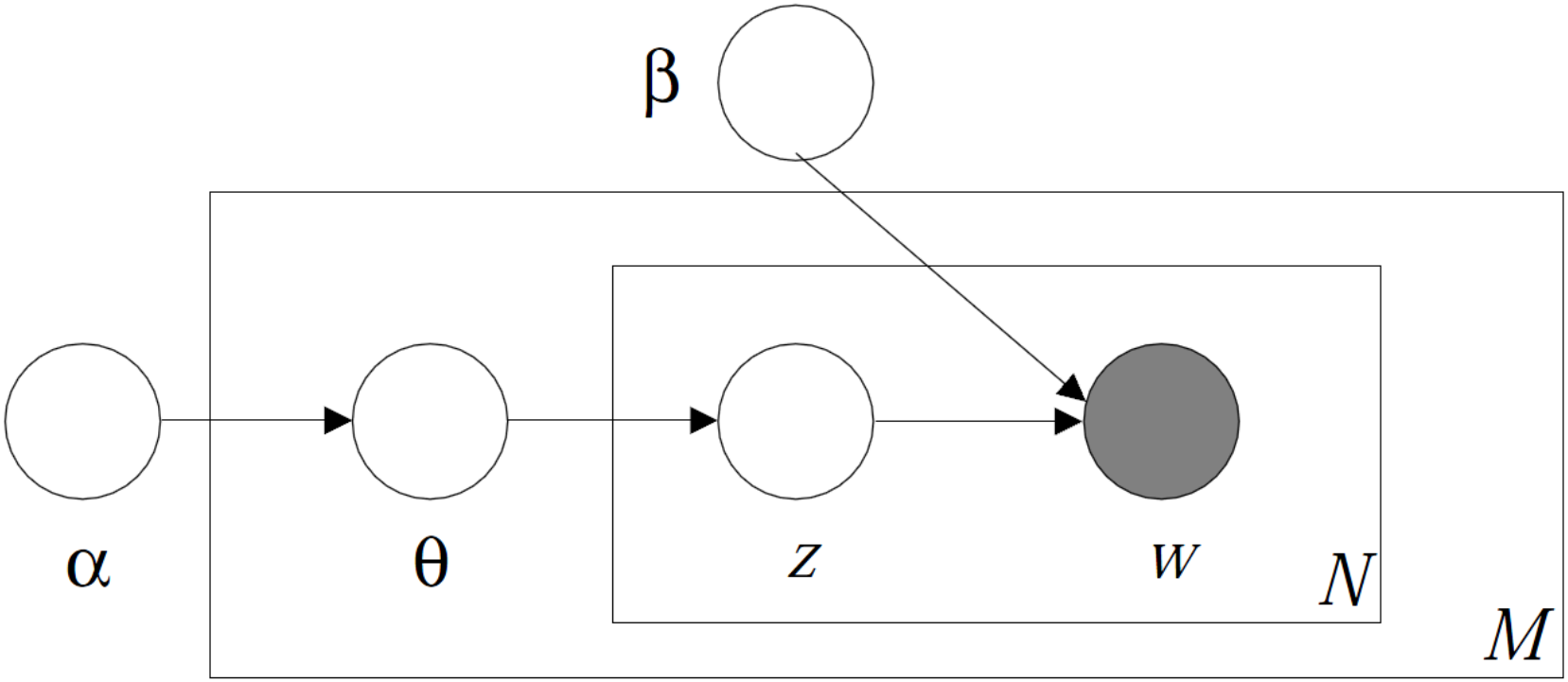
LDA

- Мы пока для простоты фиксируем число тем k, считаем, что β – это просто набор параметров $\beta_{ij} = p(w^j = 1 \mid z^i = 1)$, которые нужно оценить, и не беспокоимся о распределении на N.
- Совместное распределение тогда выглядит так:

$$p(\theta, \mathbf{z}, \mathbf{w}, N \mid \alpha, \beta) = p(N \mid \xi) p(\theta \mid \alpha) \prod_{n=1}^{N} p(z_n \mid \theta) p(w_n \mid z_n, \beta).$$

- В отличие от обычной кластеризации с априорным распределением Дирихле, мы тут не выбираем кластер один раз, а затем накидываем слова из этого кластера, а для каждого слова выбираем по распределению θ, по какой теме оно будет набросано.

LDA: графическая модель



Вывод в LDA

- Рассмотрим задачу байесовского вывода, т.е. оценки апостериорного распределения θ и z после нового документа:

$$p(\theta, \mathbf{z} \mid \mathbf{w}, \alpha, \beta) = \frac{p(\theta, \mathbf{z}, \mathbf{w} \mid \alpha, \beta)}{p(\mathbf{w} \mid \alpha, \beta)}.$$

- Правдоподобие набора слов $\mathbf{w}$ оценивается как

$$p(\mathbf{w} \mid \alpha, \beta) = \frac{\Gamma(\sum_i \alpha_i)}{\prod_i \Gamma(\alpha_i)} \int \left[\prod_{i=1}^{k} \theta_i^{\alpha_i - 1} \right] \left[\prod_{n=1}^{N} \sum_{i=1}^{k} \prod_{j=1}^{V} (\theta_i \beta_{ij})^{w_n^j} \right] d\theta,$$

и это трудно посчитать, потому что θ и β путаются друг с другом.

Вывод в LDA

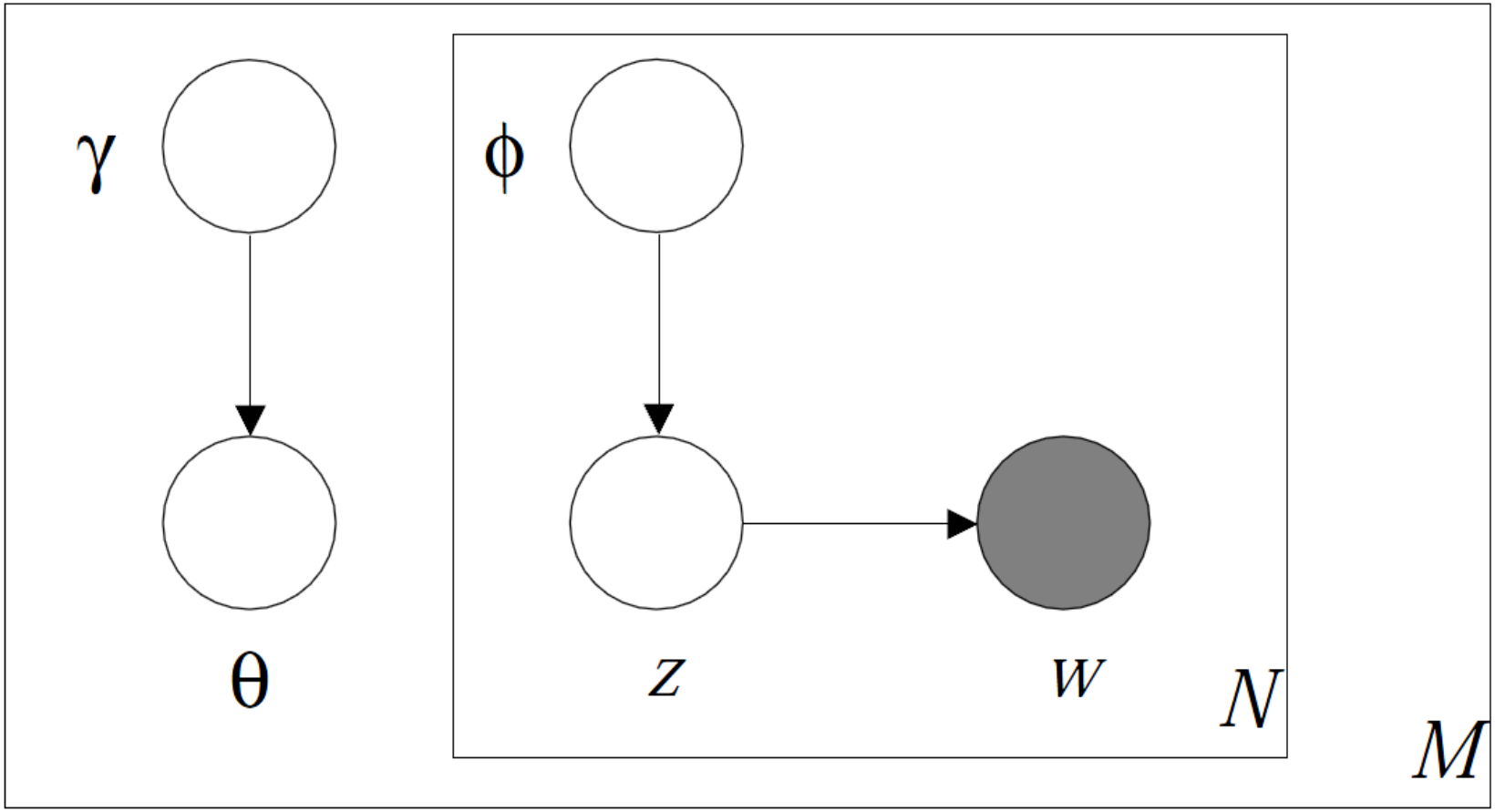
- Как делать вывод?
  1. вариационное приближение: рассмотрим семейство распределений

$$q(\theta, z \mid \mathbf{w}, \gamma, \phi) = p(\theta \mid \mathbf{w}, \gamma) \prod_{n=1}^{N} p(z_n \mid \mathbf{w}, \phi_n);$$

     тут всё расщепляется, и оптимизируем по вариационным параметрам γ (Дирихле) и ϕ (мультиномиальный);
  2. сэмплирование по Гиббсу: будем итеративно сэмплировать вектор $\mathbf{z}$, элемент за элементом, фиксируя все остальные элементы:

$$p(z_w = t \mid \mathbf{z}_{-w}, \mathbf{w}, \alpha, \beta) \propto \frac{n^{(d)}_{-w,t} + \alpha}{\sum_{t' \in T} \left(n^{(d)}_{-w,t'} + \alpha \right)} \frac{n^{(w)}_{-w,t} + \beta}{\sum_{w' \in W} \left(n^{(w')}_{-w,t} + \beta \right)}.$$

LDA: вариационное приближение



Outline

1. Категоризация текстов: naive Bayes и LDA
   - Naive Bayes
   - Latent Dirichlet allocation
2. Рекомендательные системы
   - Ближайшие соседи
   - Вероятностные модели
3. Ещё примеры
   - Байесовские рейтинг-системы
   - Модели поведения пользователей

Рекомендательные системы

- Рекомендательные системы анализируют интересы пользователей и пытаются предсказать, что именно будет наиболее интересно для конкретного пользователя в данный момент времени.
- Компании–лидеры в рекомендательных системах в основном делятся на две категории:
  1. мы продаём какие-то товары или услуги онлайн; у нас есть пользователи, которые либо явно оценивают товары, либо просто что-то покупают, а что-то нет; интересно порекомендовать товар, который данному покупателю максимально понравится; Netflix, Amazon;
  2. мы – портал, делаем деньги тем, что размещаем рекламу, надо разместить ссылки, по которым пользователи захотят переходить (и видеть ещё больше вкусной рекламы); Yahoo!, Google, Яндекс, большинство новостных сайтов.

GroupLens

- Начнём краткий обзор разных рекомендательных систем с коллаборативной фильтрации.
- Обозначения:
  - индекс i всегда будет обозначать пользователей (всего пользователей будет N, $i = 1..N$);
  - индекс a – предметы (сайты, товары, фильмы...), которые мы рекомендуем (всего M, $a = 1..M$);
  - x_i – набор (вектор) признаков (features) пользователя, x_a – набор признаков предмета;
  - когда пользователь i оценивает предмет a, он производит отклик (response, rating) $r_{i,a}$; этот отклик – случайная величина, конечно.
- Наша задача – предсказывать оценки $r_{i,a}$, зная признаки x_i и x_a для всех элементов базы и зная некоторые уже расставленные в базе $r_{i',a'}$.

GroupLens

- Начнём с непараметрического метода ближайших соседей: давайте введём расстояние между пользователями и будем рекомендовать то, что нравится вашим соседям.
- Расстояние – коэффициент корреляции (коэффициент Пирсона)

$$w_{i,j} = \frac{\sum_a (r_{i,a} - \bar{r}_a)(r_{j,a} - \bar{r}_a)}{\sqrt{\sum_a (r_{i,a} - \bar{r}_a)^2}\sqrt{\sum_a (r_{j,a} - \bar{r}_a)^2}},$$

где $\bar{r}_a$ – средний рейтинг продукта a среди всех пользователей.

GroupLens

- Простейший способ построить предсказание нового рейтинга $\hat{r}_{i,a}$ – сумма рейтингов других пользователей, взвешенная их похожестью на пользователя i:

$$\hat{r}_{i,a} = \bar{r}_a + \frac{\sum_j (r_{j,a} - \bar{r}_j)\, w_{i,j}}{\sum_j |w_{i,j}|}.$$

- Это называется GroupLens algorithm – так работал дедушка рекомендательных систем GroupLens.
- Чтобы не суммировать по всем пользователям, можно ограничиться ближайшими соседями:

$$\hat{r}_{i,a} = \bar{r}_a + \frac{\sum_{j \in \mathrm{kNN}(i)} (r_{j,a} - \bar{r}_j)\, w_{i,j}}{\sum_{j \in \mathrm{kNN}(i)} |w_{i,j}|}.$$

GroupLens

- Есть, конечно, масса расширений этого метода.
- Важное расширение – item-based collaborative filtering: давайте искать соседей не у пользователей, а у продуктов, и рекомендовать вам соседей продуктов, которые вам понравились; это позволяет значительно быстрее обновлять рекомендации.
- Другой взгляд – вероятностные модели.

Вероятностные модели

- Из чего складывается рейтинг пользователя i, который он выдал продукту a?
- Вполне может быть, что пользователь добрый и всем подряд выдаёт хорошие рейтинги; или, наоборот, злой и рейтинг зажимает.
- С другой стороны, некоторые продукты попросту лучше других.
- Поэтому мы вводим так называемые *базовые предикторы* (baseline predictors) $b_{i,a}$, которые складываются из базовых предикторов отдельных пользователей b_i и базовых предикторов отдельных продуктов b_a, а также просто общего среднего рейтинга по базе μ:

$$b_{i,a} = \mu + b_i + b_a.$$

Вероятностные модели

- Чтобы обучить предикторы, можно добавить нормально распределённый шум и получить модель линейной регрессии
$$r_{i,a} \sim \mu + b_i + b_a.$$
- Можно ввести нормальные априорные распределения, т.е. минимизировать среднеквадратическое отклонение с регуляризатором:

$$b_* = \arg\min_b \sum_{(i,a)} (r_{i,a} - \mu - b_i - b_a)^2 + \lambda_1 \left(\mu^2 + \sum_i b_i^2 + \sum_a b_a^2 \right).$$

Вероятностные модели

- С тем, чтобы напрямую обучать оставшуюся матрицу предпочтений вероятностными методами, есть одна очень серьёзная проблема – матрица X, выражающая рейтинги, содержит $N \times M$ параметров, гигантское число, которое, конечно, никак толком не обучить.
- Более того, обучать их и не надо – как мы уже говорили, данные очень разреженные, и «на самом деле» свободных параметров гораздо меньше, проблема только с тем, как их выделить.
- Поэтому обычно число независимых параметров модели необходимо уменьшать.

Вероятностные модели

- Метод SVD (singular value decomposition) – разложим матрицу X в произведение матриц маленького ранга.
- Зафиксируем некоторое число f *скрытых факторов*, которые так или иначе описывают каждый продукт и предпочтения каждого пользователя относительно этих факторов.
- Пользователь – вектором $p_i \in \mathbb{R}^f$, который показывает, насколько пользователь предпочитает те или иные факторы; продукт – вектор $q_a \in \mathbb{R}^f$, который показывает, насколько выражены те или иные факторы в этом продукте.

Вероятностные модели

- Предпочтение в итоге будем подсчитывать просто как скалярное произведение $q_a^\top p_i = \sum_{j=1}^f q_{a,j} p_{i,j}$.
- Таким образом, добавляя теперь сюда baseline-предикторы, получаем следующую модель предсказаний рейтингов:

$$\hat{r}_{i,a} \sim \mu + b_i + b_a + q_a^\top p_i.$$

- Её легко обучать градиентным спуском.

Вероятностные модели

- Можно добавлять и дополнительную информацию в эту модель. Например, введём дополнительный набор факторов для продуктов y_a, которые будут характеризовать пользователя на основе того, что он просматривал, но не оценивал.
- Модель после этого принимает вид

$$\hat{r}_{i,a} = \mu + b_i + b_a + q_a^\top \left(p_i + \frac{1}{\sqrt{|V(i)|}} \sum_{b \in V(i)} y_b \right),$$

где $V(i)$ – множество продуктов, которые просматривал этот пользователь ($\frac{1}{\sqrt{|V(i)|}}$ контролирует дисперсию).
- Это называется SVD++, и эта модель была одним из основных компонентов модели, взявшей Netflix Prize.

Вероятностное разложение матриц

- Пусть мы хотим построить разложение матрицы рейтингов на матрицы меньшего ранга:

$$\hat{R} = U^\top V.$$

- Вероятностно мы имеем правдоподобие

$$p(R \mid U, V, \sigma^2) = \prod_i \prod_a \left(\mathcal{N}(r_{i,a} \mid u_i^\top v_j, \sigma^2)\right)^{[i \text{ оценил } a]}.$$

- Добавим гауссовские априорные распределения на U и V:

$$p(U \mid \sigma_U^2) = \prod_i \mathcal{N}(U_i \mid \mathbf{0}, \sigma_U^2 \boldsymbol{I}), \quad p(V \mid \sigma_V^2) = \prod_a \mathcal{N}(V_a \mid \mathbf{0}, \sigma_V^2 \boldsymbol{I}).$$

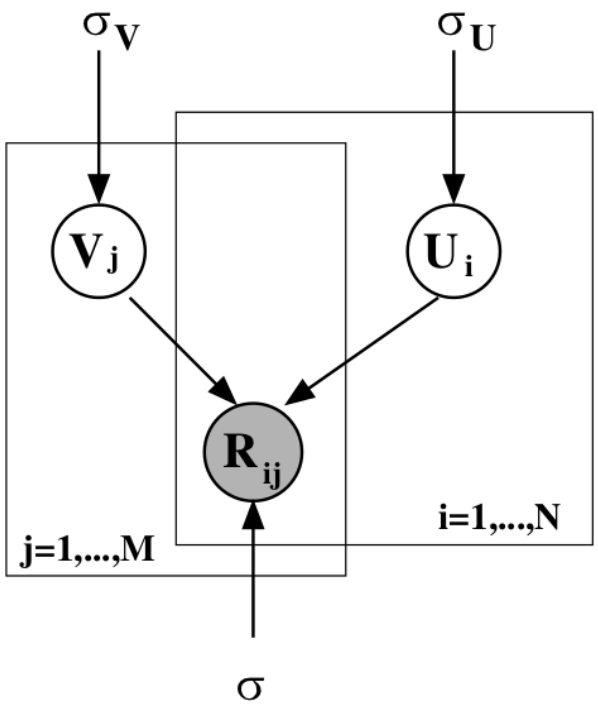
Вероятностное разложение матриц

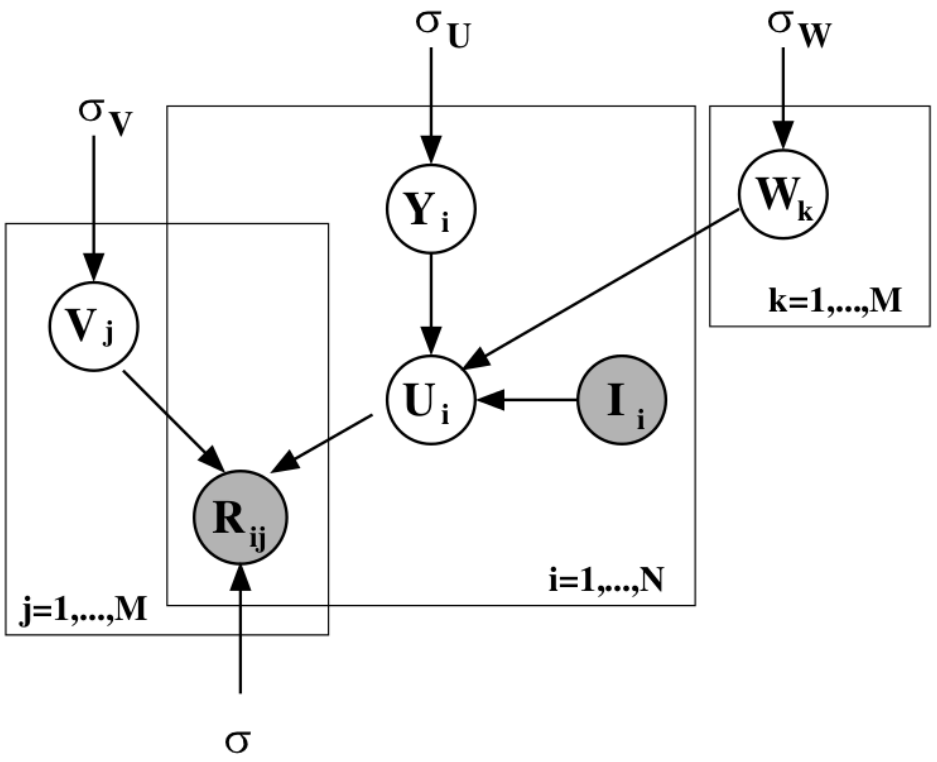
- Модификация: пользователи с малым числом оценок в PMF получат апостериорные распределения, очень похожие на «среднего пользователя».
- Чтобы на редких пользователей лучше обобщалось, добавим ещё факторы, которые меняют априорное распределение факторов у пользователя в зависимости от того, сколько и чего он оценил:

$$U_i = Y_i + \frac{\sum_a [i \text{ оценил } a] W_a}{\sum_a [i \text{ оценил } a]}.$$

- Матрица W показывает, как влияет на априорное распределение
- Тоже в качестве регуляризатора берём априорный гауссиан: $p(W \mid \sigma_W^2) = \prod_i \mathcal{N}(W_i \mid \mathbf{0}, \sigma_W^2 \boldsymbol{I})$.

Графическая модель





Outline

1. Категоризация текстов: naive Bayes и LDA
   - Naive Bayes
   - Latent Dirichlet allocation
2. Рекомендательные системы
   - Ближайшие соседи
   - Вероятностные модели
3. Ещё примеры
   - Байесовские рейтинг-системы
   - Модели поведения пользователей

Частичные сравнения

- Рейтинг-система – это модель, которая ранжирует участников (игроков) в единый линейный порядок по данным сравнений небольших подмножеств этих игроков (турниров).
- Более того, результаты турниров зашумлены (отчасти случайны).
- Соответственно, и применяются они в таких ситуациях (пример: контекстная реклама в Bing).

Рейтинг Эло

- Первая известная рейтинг-система, основанная на байсеовском подходе.
- Суть модели:
  - сила игры шахматиста в одной партии – случайная величина;
  - *рейтинг* – это ожидание этой величины; мы пытаемся оценить это ожидание;
  - исходная модель Эло – нормальное распределение силы игры вокруг рейтинга.

Рейтинг Эло

- Значит, сила игры в конкретной партии распределена как

$$p(x) = \mathcal{N}(x; s, \beta) = \frac{1}{\beta\sqrt{2\pi}} e^{-\frac{1}{2\beta^2}(x-s)^2}.$$

- Сила игры задаётся двумя параметрами: средним s (собственно рейтингом) и дисперсией β^2.
- Эло предположил, что дисперсия β^2 постоянна (и даже от игрока не зависит), а среднее – это как раз рейтинг, который мы пытаемся оценить.

Рейтинг Эло

- Значит, математически говоря, мы ищем

$$\arg\max_{s,\beta^2} p(s, \beta^2 \mid D) = \arg\max_{s,\beta^2} \frac{p(D \mid s, \beta^2)p(s, \beta^2)}{p(D)} =$$
$$= \arg\max_{s,\beta^2} p(D \mid s, \beta^2)p(s, \beta^2).$$

- Как мы знаем, нормальное распределение является самосопряжённым, поэтому если сила игры нормально распределена вокруг рейтинга, то логично взять нормальное распределение как априорное для рейтинга.
- Таким образом, рейтинг игрока складывается из двух чисел: его среднего значения μ и дисперсии σ^2.
- Значение μ отображается в таблице рейтингов, а σ^2 показывает, насколько достоверна имеющаяся оценка.

Рейтинг Эло

- Предположим, что встречаются два игрока с некоторыми априорными распределениями на рейтинги $\mathcal{N}(s_1; \mu_1, \sigma_1^2)$ и $\mathcal{N}(s_2; \mu_2, \sigma_2^2)$.
- Тогда сила игры каждого из них в этой конкретной партии имеет распределение

$$p(x \mid \mu, \sigma) = \int_{-\infty}^{\infty} p(x \mid s) p(s \mid \mu, \sigma) ds =$$

$$= \int_{-\infty}^{\infty} \mathcal{N}(x; s, \beta) \mathcal{N}(s; \mu, \sigma) ds =$$

$$= \int_{-\infty}^{\infty} \frac{1}{\beta\sqrt{2\pi}} e^{-\frac{1}{2\beta^2}(x-s)^2} \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2\sigma^2}(s-\mu)^2} ds = \mathcal{N}(x; \mu_x, \sigma_x).$$

то есть мы снова приходим к нормальному распределению, но с другими параметрами.

Рейтинг Эло

- Задача обучения заключается в том, чтобы после новой партии принять во внимание её результат и пересчитать рейтинги.
- Эло разработал специальные аппроксимации и очень простые алгоритмы для этого случая (через «ожидаемые очки в турнире»), чтобы каждый шахматист мог сам на калькуляторе свой рейтинг посчитать, но они нас сейчас не очень интересуют.

Модели Брэдли–Терри

- Следующий подход к рейтинг–системам – модели Брэдли–Терри (Bradley–Terry).
- Модель предполагает, что для участников $1, \ldots, n$ можно подобрать такие рейтинги γ_i, $i = 1..n$, что вероятность победы участника i над участником j равна

$$p(i \text{ побеждает } j) = \frac{\gamma_i}{\gamma_i + \gamma_j}.$$

- Основная задача заключается в том, чтобы найти $\boldsymbol{\gamma} = (\gamma_1, \ldots, \gamma_m)$ максимального правдоподобия из имеющихся данных D.

Модели Брэдли–Терри

- Если принять априорное распределение равномерным, можно просто максимизировать правдоподобие

$$p(D|\boldsymbol{\gamma}) = \prod_{i=1}^{m}\prod_{j=1}^{m}\left(\frac{\gamma_i}{\gamma_i+\gamma_j}\right)^{w_{ij}},$$

где w_{ij} — то, сколько раз x_i обыграл x_j при их попарном сравнении ($w_{ii} = 0$ по определению).

- Взяв логарифм, будем максимизировать

$$l(\boldsymbol{\gamma}) = \sum_{i=1}^{m}\sum_{j=1}^{m}\left(w_{ij}\log\gamma_i - w_{ij}\log(\gamma_i+\gamma_j)\right).$$

Модели Брэдли–Терри

- Максимизировать будем классическим MM-алгоритмом (minorization–maximization), фактически вариационным приближением.
- Заметим, что

$$1 + \log \frac{x}{y} - \frac{x}{y} \leq 0.$$

Упражнение. Докажите это.

Модели Брэдли–Терри

- Рассмотрим вспомогательную функцию

$$Q(\boldsymbol{\gamma}, \boldsymbol{\gamma}^{(k)}) = \sum_{i,j} w_{ij} \left[\log \gamma_i - \frac{\gamma_i + \gamma_j}{\gamma_i^{(k)} + \gamma_j^{(k)}} - \log \left(\gamma_i^{(k)} + \gamma_j^{(k)} \right) + 1 \right].$$

Упражнение. Используя предыдущее неравенство, докажите, что $Q(\boldsymbol{\gamma}, \boldsymbol{\gamma}^{(k)}) \leq l(\boldsymbol{\gamma})$.

Модели Брэдли–Терри

- Чтобы найти $\max_{\boldsymbol{\gamma}} Q\left(\boldsymbol{\gamma}, \boldsymbol{\gamma}^{(k)}\right)$, можно просто взять производные $\frac{\partial Q}{\partial \gamma_l}$:

$$\frac{\partial Q}{\partial \gamma_l} = \sum_{i,j} w_{ij} \left[\frac{\delta_{il}}{\gamma_i} - \frac{\delta_{il} + \delta_{jl}}{\gamma_i^{(k)} + \gamma_j^{(k)}} \right] =$$

$$= \frac{1}{\gamma_l} \sum_j w_{lj} - \sum_j \frac{w_{lj}}{\gamma_i^{(k)} + \gamma_j^{(k)}} - \sum_i \frac{w_i l}{\gamma_i^{(k)} + \gamma_j^{(k)}}.$$

Модели Брэдли–Терри

- Если w_l – общее количество побед игрока l ($w_l = \sum_j w_{lj}$), и N_{ij} — количество встреч между игроками i и j ($N_{ij} = w_{ij} + w_{ji}$), получаем

$$\frac{w_l}{\gamma_l} - \sum_j \frac{N_{lj}}{\gamma_i^{(k)} + \gamma_j^{(k)}} = 0.$$

- В результате правило пересчёта на одной итерации выглядит так:

$$\gamma_l^{(k+1)} := w_l \left[\sum_j \frac{N_{lj}}{\gamma_i^{(k)} + \gamma_j^{(k)}} \right]^{-1}.$$

Модели Брэдли–Терри

- Получили алгоритм оценки рейтингов. Правда, он пока работает только для ситуации, когда игроки встречаются один на один и выигрывают или проигрывают.
- Но даже в шахматах бывают ничьи. Как их учесть?

Модели Брэдли–Терри

- Если возможны ничьи, их вероятность можно описать дополнительным параметром $\theta > 1$, и это приводит к модели, в которой

$$
\begin{aligned}
p(i \text{ побеждает } j) &= \frac{\gamma_i}{\gamma_i + \theta\gamma_j}, \\
p(j \text{ побеждает } i) &= \frac{\gamma_j}{\theta\gamma_i + \gamma_j}, \\
p(i \text{ и } j \text{ играют вничью}) &= \frac{(\theta^2 - 1)\gamma_i\gamma_j}{(\gamma_i + \theta\gamma_j)\,(\theta\gamma_i + \gamma_j)}.
\end{aligned}
$$

Модели Брэдли–Терри

- Аналогично можно вводить другие обобщения.
- Например, если результат может зависеть от порядка элементов в паре (скажем, команды проводят «домашние» матчи и «гостевые»), можно ввести дополнительный параметр θ, характеризующий, насколько большое преимущество дают «родные стены», и рассмотреть модель с

$$p(i \text{ побеждает } j) = \begin{cases} \frac{\theta\gamma_i}{\theta\gamma_i+\gamma_j}, & \text{если } i \text{ играет дома,} \\ \frac{\gamma_j}{\theta\gamma_i+\gamma_j}, & \text{если } j \text{ играет дома.} \end{cases}$$

Модели Брэдли–Терри

- Можно даже обобщить на случай, когда в одном турнире встречаются несколько игроков: пусть перестановка π подмножества игроков $A = \{1, \ldots, k\}$ (результат турнира) имеет вероятность

$$p_A(\pi) = \prod_{i=1}^{k} \frac{\gamma_{\pi(i)}}{\gamma_{\pi(i)} + \gamma_{\pi(i+1)} + \ldots + \gamma_{\pi(k)}}.$$

Модели Брэдли–Терри

- Можно показать, что такая модель эквивалентна весьма естественной «аксиоме Люса»: для любой модели, в которой вероятности игроков обыграть друг друга в любой паре не равны нулю, для любых подмножеств игроков $A \subset B$ и любого игрока $i \in A$

$$p_B(i \text{ побеждает}) =$$
$$= p_A(i \text{ побеждает}) p_B(\text{побеждает кто-то из множества } A).$$

- Но для крупных турниров это перестаёт работать; и совсем трудно что-то осмысленное сделать, если игроки соревнуются не поодиночке, а в командах.

Рейтинг-система TrueSkill

- Была разработана в Microsoft Research для игровых серверов Xbox 360.
- Постановка задачи теперь становится максимально общей.
- Системf TrueSkill вычисляет рейтинги игроков, которые объединяются в команды разного размера и участвуют в матчах (турнирах) с несколькими участниками.
- Задача — после каждого из таких турниров пересчитать апостериорные рейтинги.

Рейтинг-система TrueSkill

- Начнём понемногу «разворачивать» то, что происходит в каждом из этих турниров.
- Во-первых, мы не знаем достоверных априорных значений рейтингов, у нас есть только некоторое априорное распределение (мы считаем его нормальным)

$$f(s_i) = \mathcal{N}(s; \mu_i, \sigma_i).$$

- Здесь μ_i — это собственно рейтинг игрока, а σ_i — «показатель достоверности» рейтинга, дисперсия.

Рейтинг-система TrueSkill

- Каждый «истинный» скилл является средним значением, вокруг которого распределены конкретные показатели силы игры того или иного игрока в данной конкретной партии (p_i, performance):

$$f(p_i \mid s_i) = \mathcal{N}(p_i; s_i, \beta^2).$$

- В системе TrueSkill (как в рейтинге Эло) делается предположение, что β^2 – универсальная константа.
- Тогда p_i через исходные параметры выражается как

$$f(p_i \mid \mu_i, \sigma_i) = \int_{-\infty}^{\infty} \mathcal{N}(p_i; s_i, \beta^2)\mathcal{N}(s_i; \mu_i, \sigma_i)\,ds_i.$$

Рейтинг-система TrueSkill

- Затем показатели силы игры игроков в конкретных партиях объединяются и дают оценки на силу игры команд.
- В системе TrueSkill предполагается, что сила команды равна сумме сил её игроков:

$$t_i = \sum_i p_i.$$

Рейтинг-система TrueSkill

- После этого показатели силы команд в данном турнире нужно сравнить друг с другом; их сравнение и должно порождать тот порядок, который записан в результатах турнира.
- Будем считать, что ничья между командами с силой t_1 и t_2 означает, что

$$|t_1 - t_2| < \epsilon$$

для некоторого ϵ (тоже универсальная константа).

Рейтинг-система TrueSkill

- Мы должны подсчитать апостериорные рейтинги команд после получения данных.
- Данные приходят к нам в виде перестановки команд π: упорядоченных результатов турнира (в которых могут быть ничьи между соседними командами).
- Иначе говоря, нужно подсчитать

$$p(\boldsymbol{s} \mid \pi) = \frac{p(\pi \mid \boldsymbol{s})p(\boldsymbol{s})}{\int p(\pi \mid \boldsymbol{s})p(\boldsymbol{s})d\boldsymbol{s}}.$$

Рейтинг-система TrueSkill

- В нашей системе присутствуют, кроме s_i и π, ещё переменные p_i, t_i и d_i, причём плотность распределения всей системы мы только что представили в виде произведения распределений:

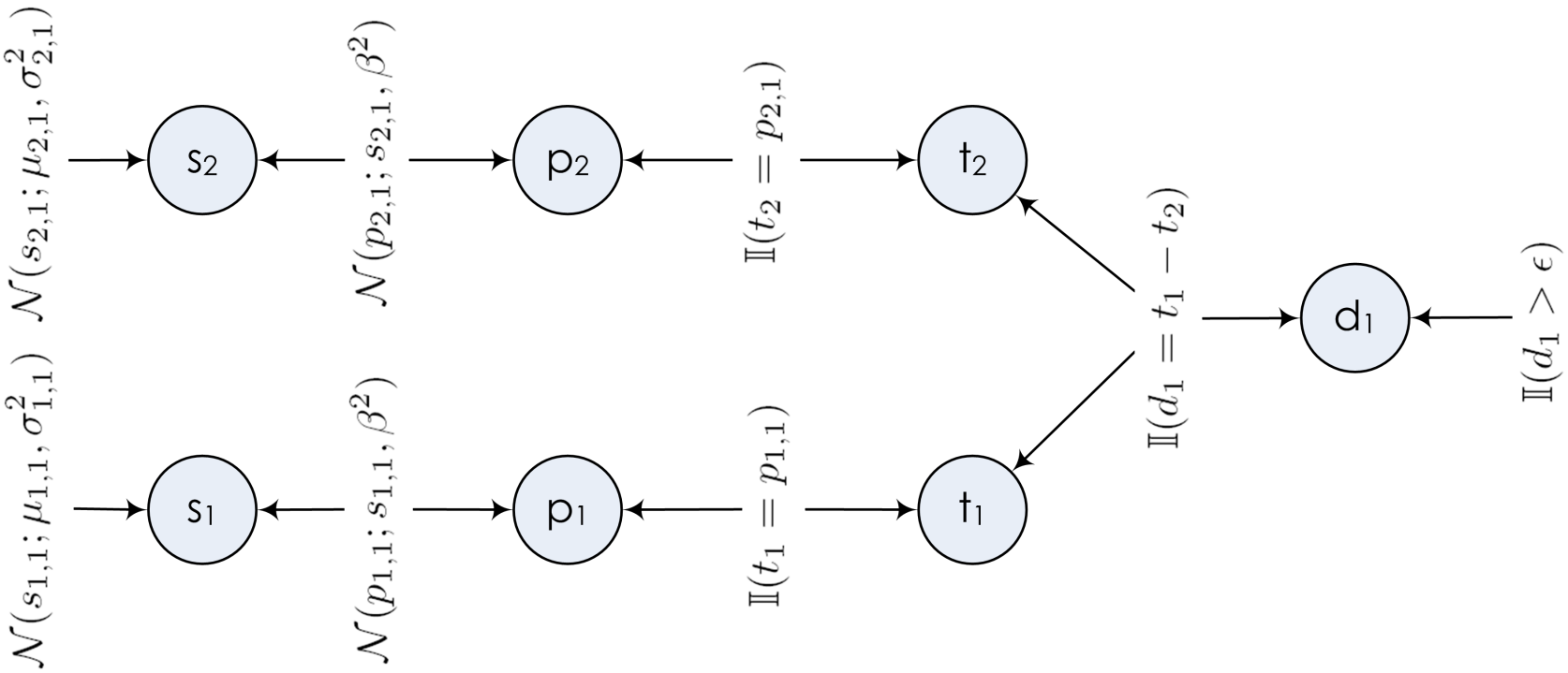
$$p(\pi, \boldsymbol{d}, \boldsymbol{t}, \boldsymbol{p}, \boldsymbol{s}) = p(\pi \mid \boldsymbol{d})p(\boldsymbol{d} \mid \boldsymbol{t})p(\boldsymbol{t} \mid \boldsymbol{p})p(\boldsymbol{p} \mid \boldsymbol{s})p(\boldsymbol{s}).$$

- А нам нужно вычислить

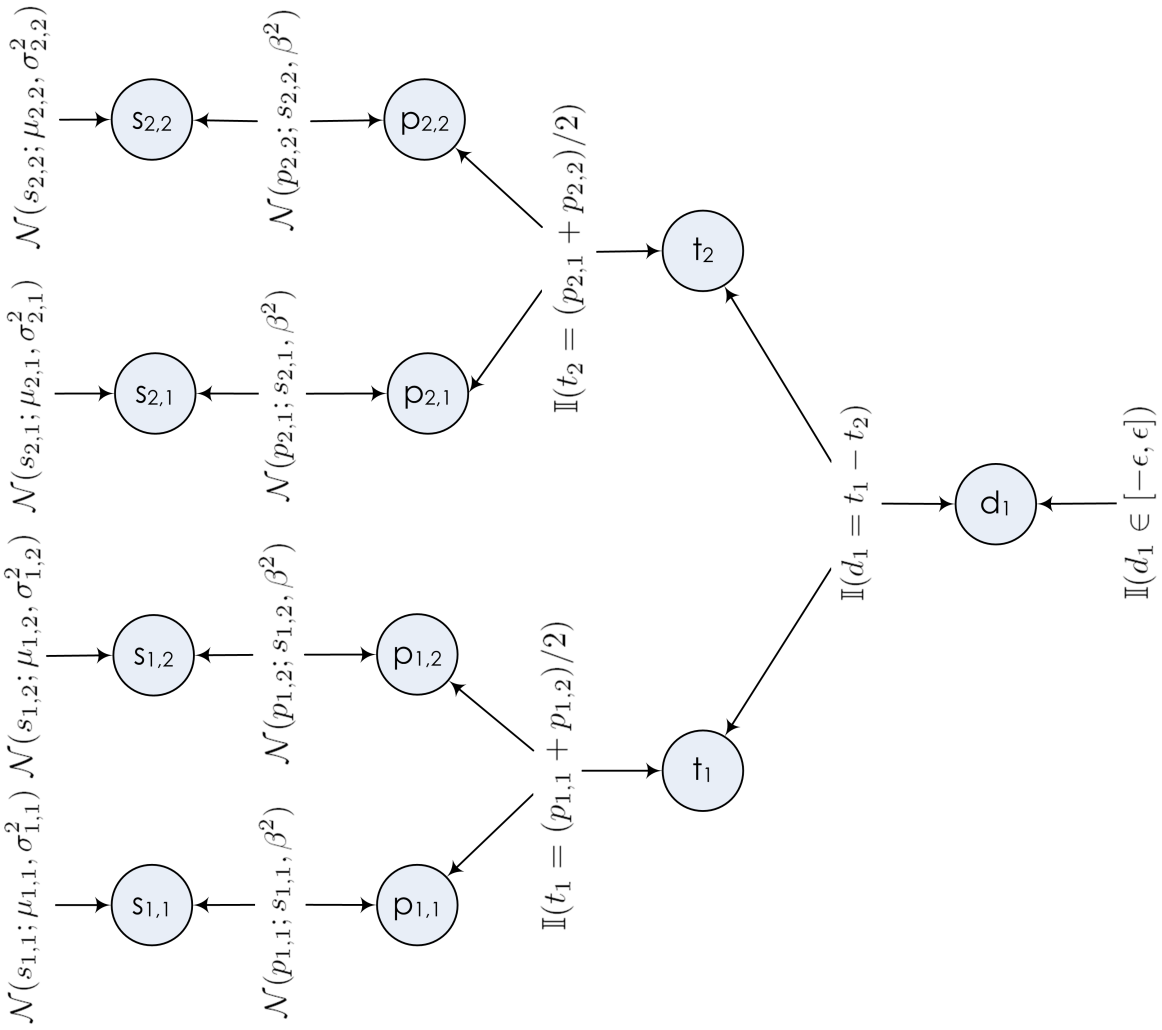
$$p(\pi \mid \boldsymbol{s}) = \int\int\int p(\pi, \boldsymbol{d}, \boldsymbol{t}, \boldsymbol{p}, \boldsymbol{s}) d\boldsymbol{d} d\boldsymbol{t} d\boldsymbol{p}.$$

- Получили обычную задачу маргинализации.

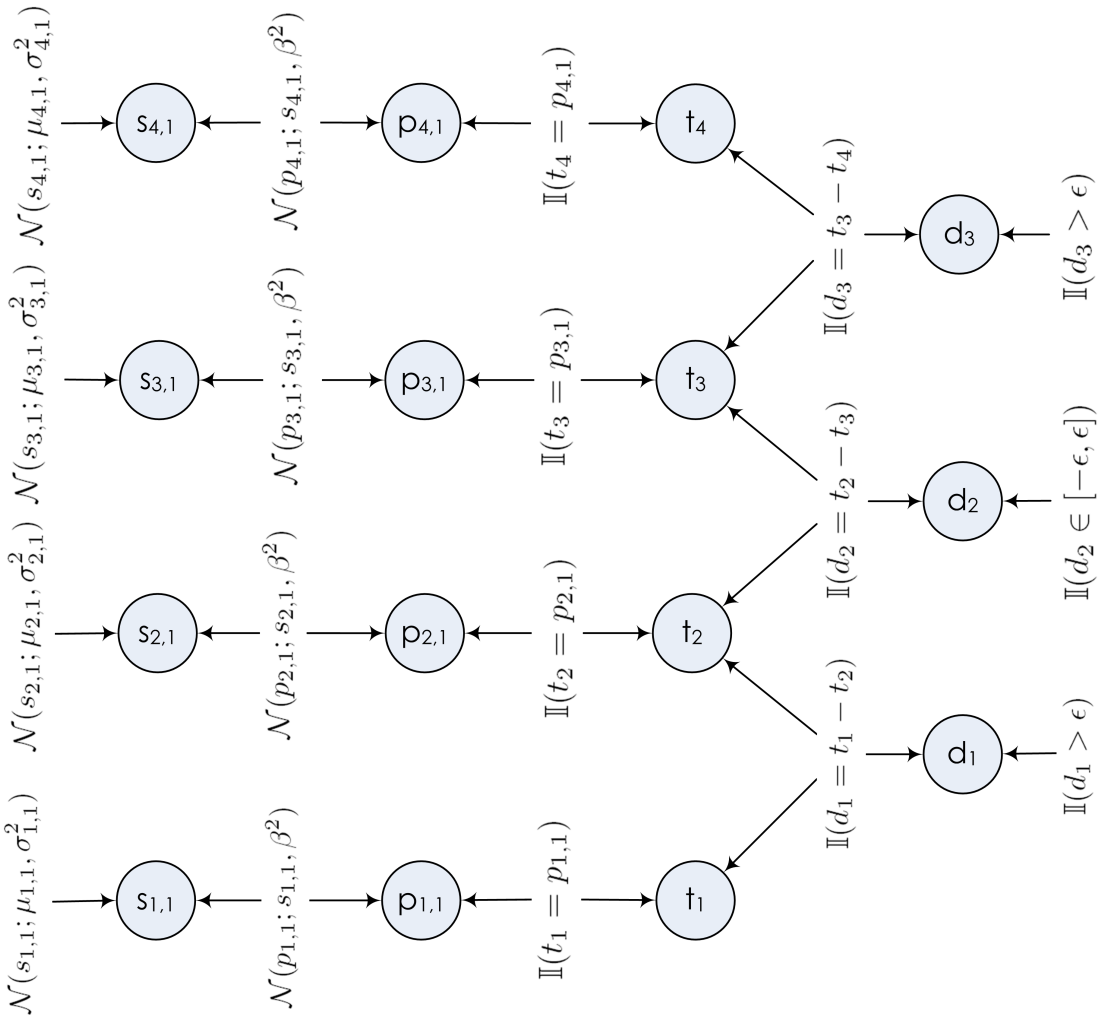
Матч двух игроков



Две команды по два игрока



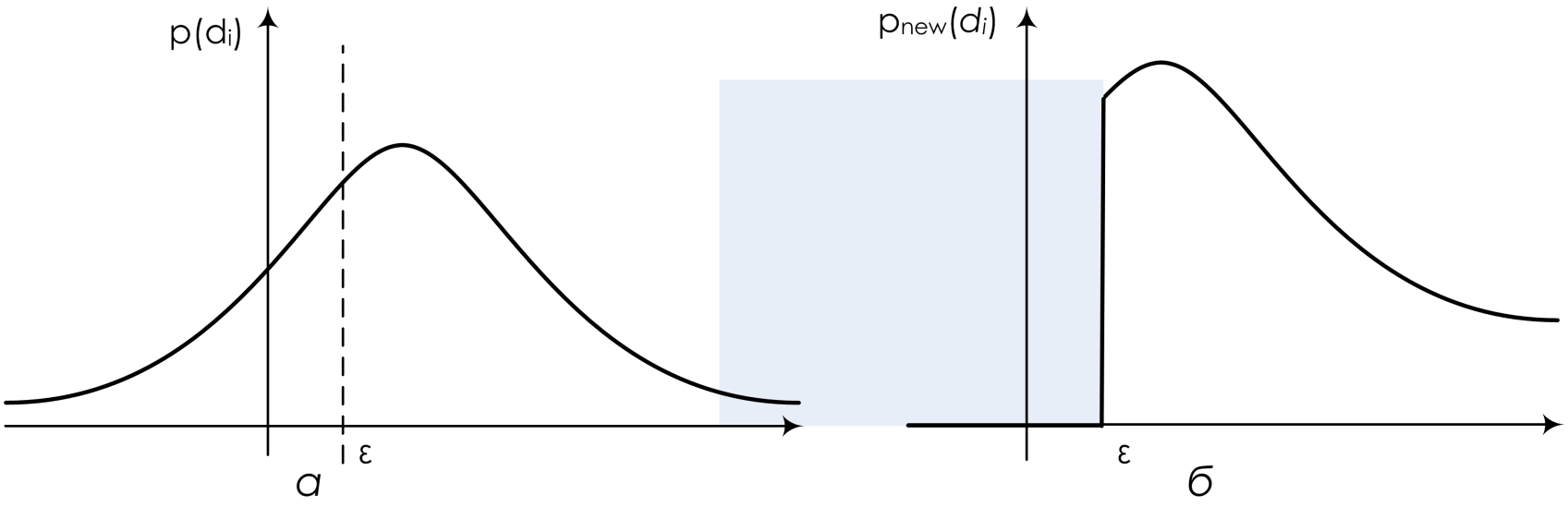
Матч четырёх игроков



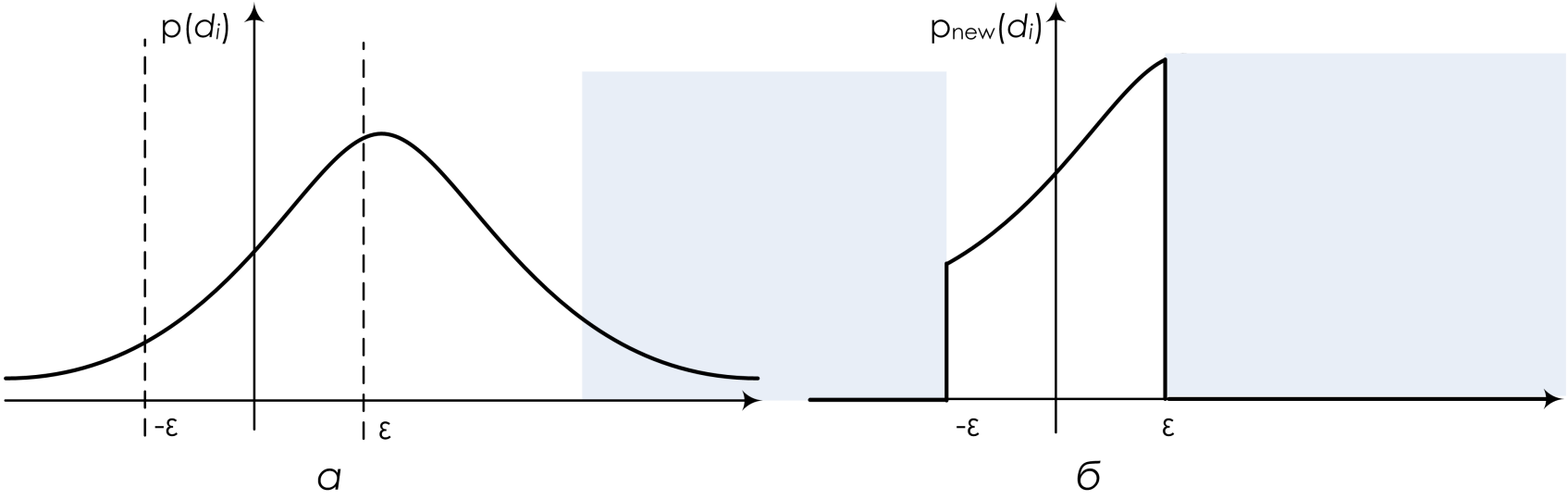
Приближённый вывод

- Казалось бы, граф – дерево, что ещё говорить.
- Но возникает проблема с выводом на нижнем уровне графа.
- Что делает функция, которая собственно определяет победу или ничью?

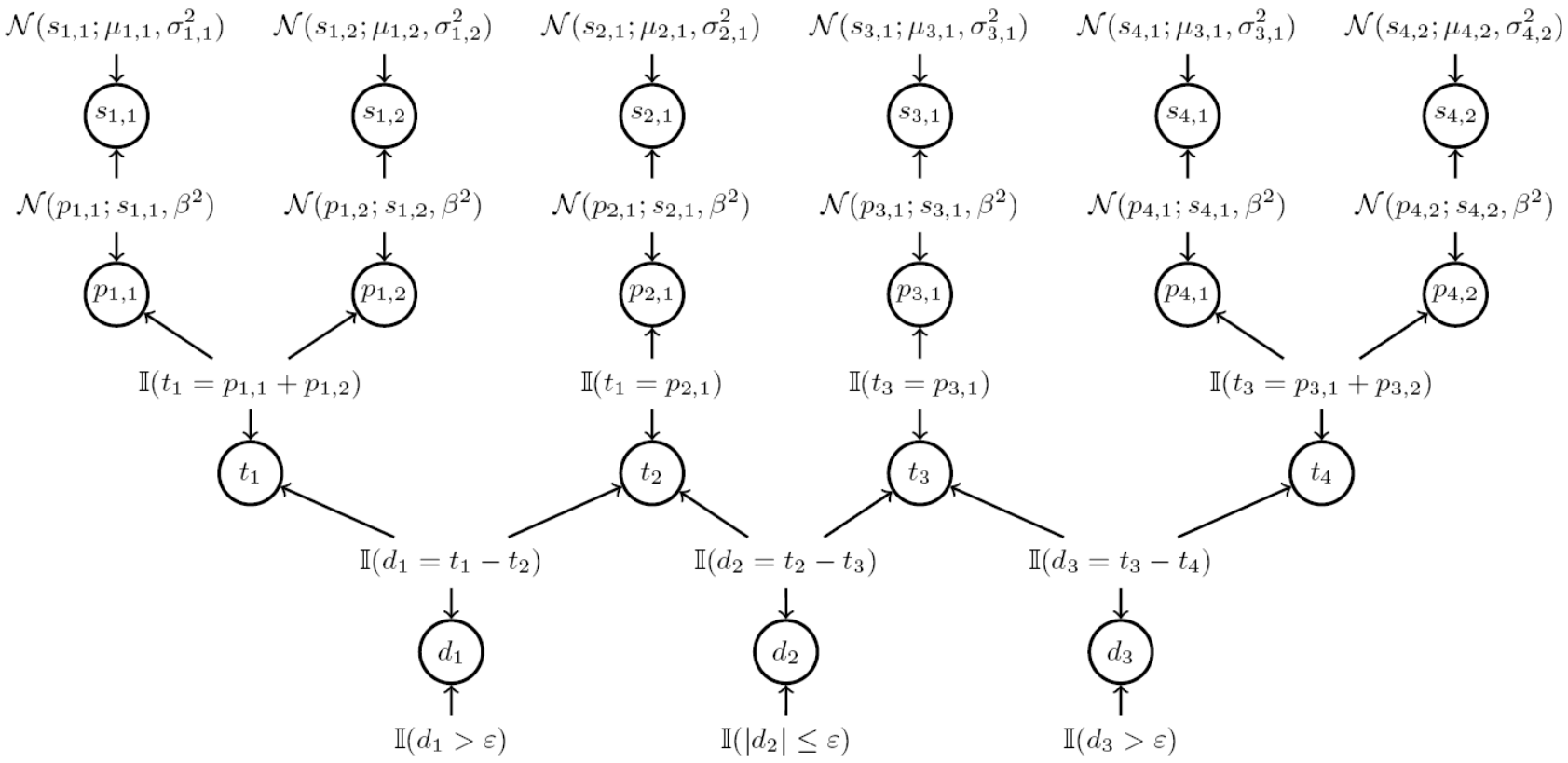
Победа



Ничья



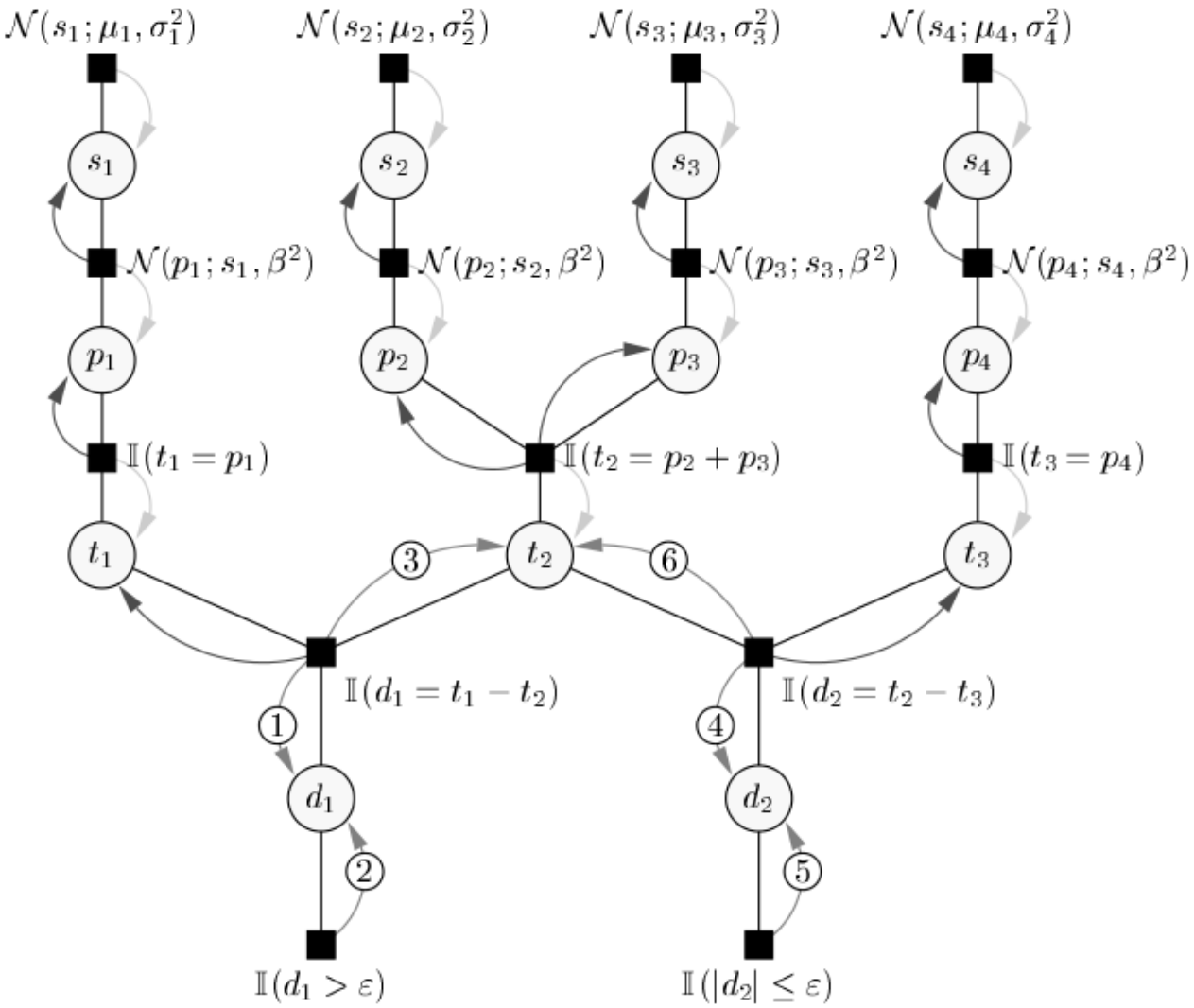
Общий случай



Приближённый вывод

- Для вывода на нижнем уровне графа применяется алгоритм Expectation Propagation [Minka, 2001]:
  - приближаем сообщение от функции к переменной (т.е. распределение p) некоторым семейством распределений $q(\lambda)$;
  - передаём сообщения взад-вперёд, пока оценки не сойдутся.
- В качестве приближённого семейства будем рассматривать семейство нормальных распределений. Тогда для поиска оптимального приближения надо просто найти ожидание и дисперсию (первые два момента) распределения p.
- Для таких «обрезанных» распределений это можно сделать явно, получатся конкретные формулы для передачи сообщений.

Пример


$\mathcal{N}(s_1; \mu_1, \sigma_1^2)$
$\mathcal{N}(s_2; \mu_2, \sigma_2^2)$
$\mathcal{N}(s_3; \mu_3, \sigma_3^2)$
$\mathcal{N}(s_4; \mu_4, \sigma_4^2)$
s_1
s_2
s_3
s_4
$\mathcal{N}(p_1; s_1, \beta^2)$
$\mathcal{N}(p_2; s_2, \beta^2)$
$\mathcal{N}(p_3; s_3, \beta^2)$
$\mathcal{N}(p_4; s_4, \beta^2)$
p_1
p_2
p_3
p_4
$\mathbb{I}(t_1 = p_1)$
$\mathbb{I}(t_2 = p_2 + p_3)$
$\mathbb{I}(t_3 = p_4)$
t_1
t_2
t_3
3
6
$\mathbb{I}(d_1 = t_1 - t_2)$
$\mathbb{I}(d_2 = t_2 - t_3)$
1
4
d_1
d_2
2
5
$\mathbb{I}(d_1 > \varepsilon)$
$\mathbb{I}(|d_2| \leq \varepsilon)$

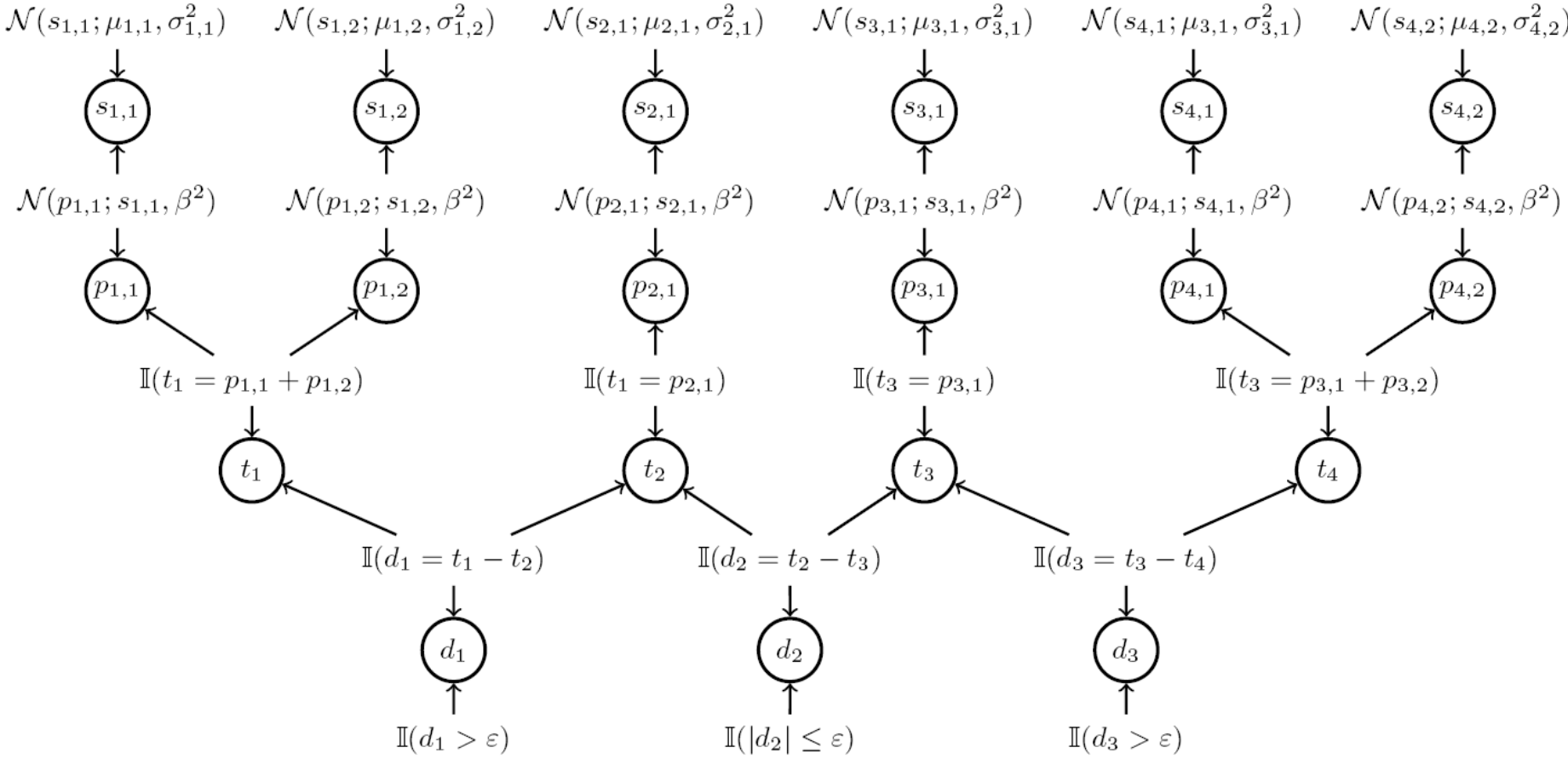
Проблемы TrueSkill

- У TrueSkill есть проблемы, с которыми мы столкнулись, когда попытались применить её на практике.
- Мы хотели сделать рейтинг спортивного «Что? Где? Когда?»:
  - участвуют команды по ≤ 6 человек, причём часто встречаются неполные команды;
  - игроки постоянно переходят между командами (поэтому TrueSkill);
  - в одном турнире могут участвовать до тысячи команд (синхронные турниры);
  - командам задаётся фиксированное число вопросов (36, 60, 90), т.е. в крупных турнирах очень много команд делят одно и то же место.

Проблемы TrueSkill

- У системы TrueSkill при этом тут же начинаются проблемы.
- Главная проблема – большие ничьи на много команд.
- Вторая проблема – сила команды как сумма сил игроков.

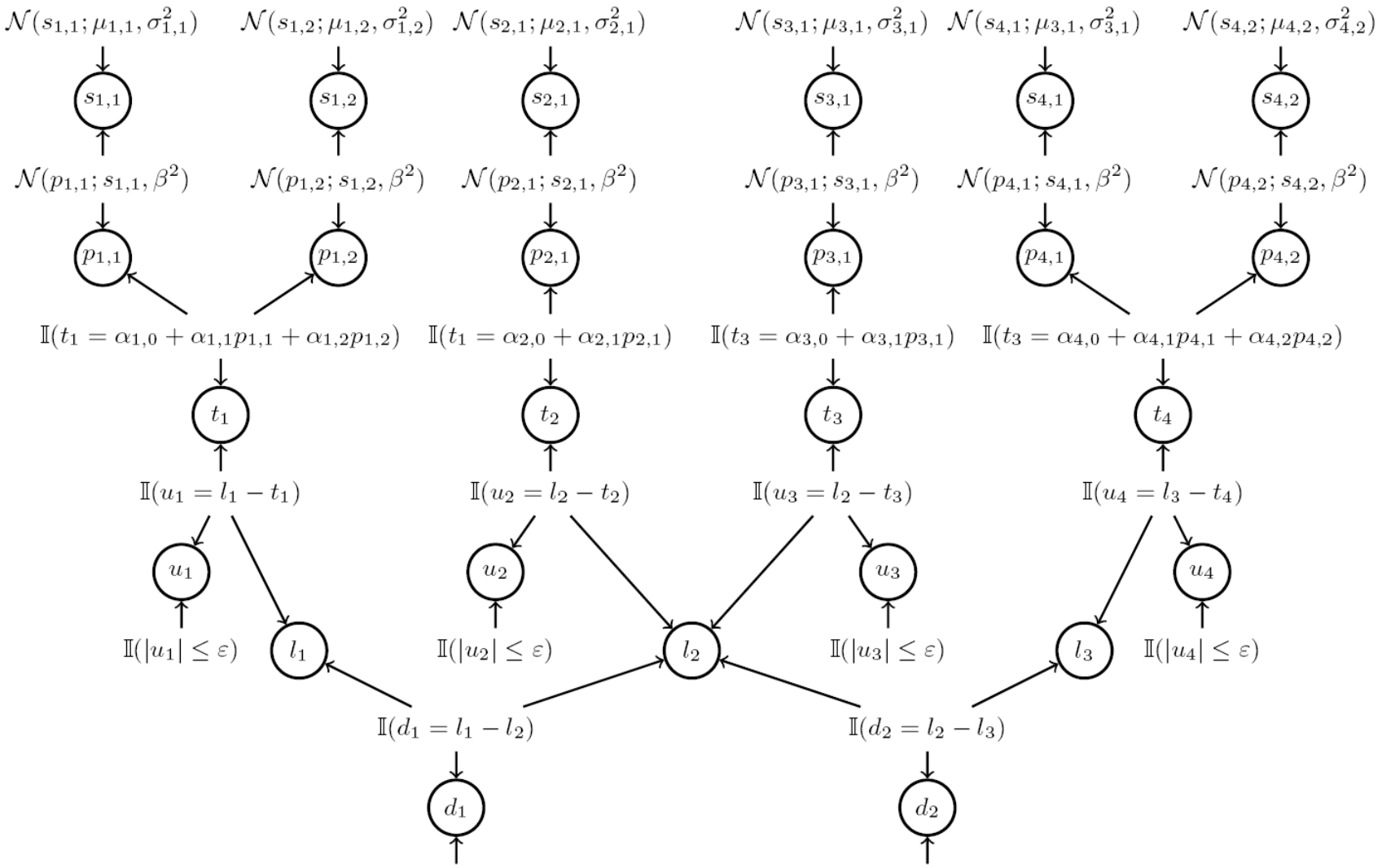
Пример


$\mathcal{N}(s_{1,1}; \mu_{1,1}, \sigma^2_{1,1})$
$\mathcal{N}(s_{1,2}; \mu_{1,2}, \sigma^2_{1,2})$
$\mathcal{N}(s_{2,1}; \mu_{2,1}, \sigma^2_{2,1})$
$\mathcal{N}(s_{3,1}; \mu_{3,1}, \sigma^2_{3,1})$
$\mathcal{N}(s_{4,1}; \mu_{3,1}, \sigma^2_{3,1})$
$\mathcal{N}(s_{4,2}; \mu_{4,2}, \sigma^2_{4,2})$
$s_{1,1}$
$s_{1,2}$
$s_{2,1}$
$s_{3,1}$
$s_{4,1}$
$s_{4,2}$
$\mathcal{N}(p_{1,1}; s_{1,1}, \beta^2)$
$\mathcal{N}(p_{1,2}; s_{1,2}, \beta^2)$
$\mathcal{N}(p_{2,1}; s_{2,1}, \beta^2)$
$\mathcal{N}(p_{3,1}; s_{3,1}, \beta^2)$
$\mathcal{N}(p_{4,1}; s_{4,1}, \beta^2)$
$\mathcal{N}(p_{4,2}; s_{4,2}, \beta^2)$
$p_{1,1}$
$p_{1,2}$
$p_{2,1}$
$p_{3,1}$
$p_{4,1}$
$p_{4,2}$
$\mathbb{I}(t_1 = p_{1,1} + p_{1,2})$
$\mathbb{I}(t_1 = p_{2,1})$
$\mathbb{I}(t_3 = p_{3,1})$
$\mathbb{I}(t_3 = p_{3,1} + p_{3,2})$
t_1
t_2
t_3
t_4
$\mathbb{I}(d_1 = t_1 - t_2)$
$\mathbb{I}(d_2 = t_2 - t_3)$
$\mathbb{I}(d_3 = t_3 - t_4)$
d_1
d_2
d_3
$\mathbb{I}(d_1 > \varepsilon)$
$\mathbb{I}(|d_2| \leq \varepsilon)$
$\mathbb{I}(d_3 > \varepsilon)$

Проблемы TrueSkill

- [Nikolenko, Sirotkin, 2011]: изменив структуру factor graph'а, получилось решить проблему с дележом мест.
- Проблема с силой команд, конечно, должна решаться индивидуально в каждом конкретном приложении.

Пример

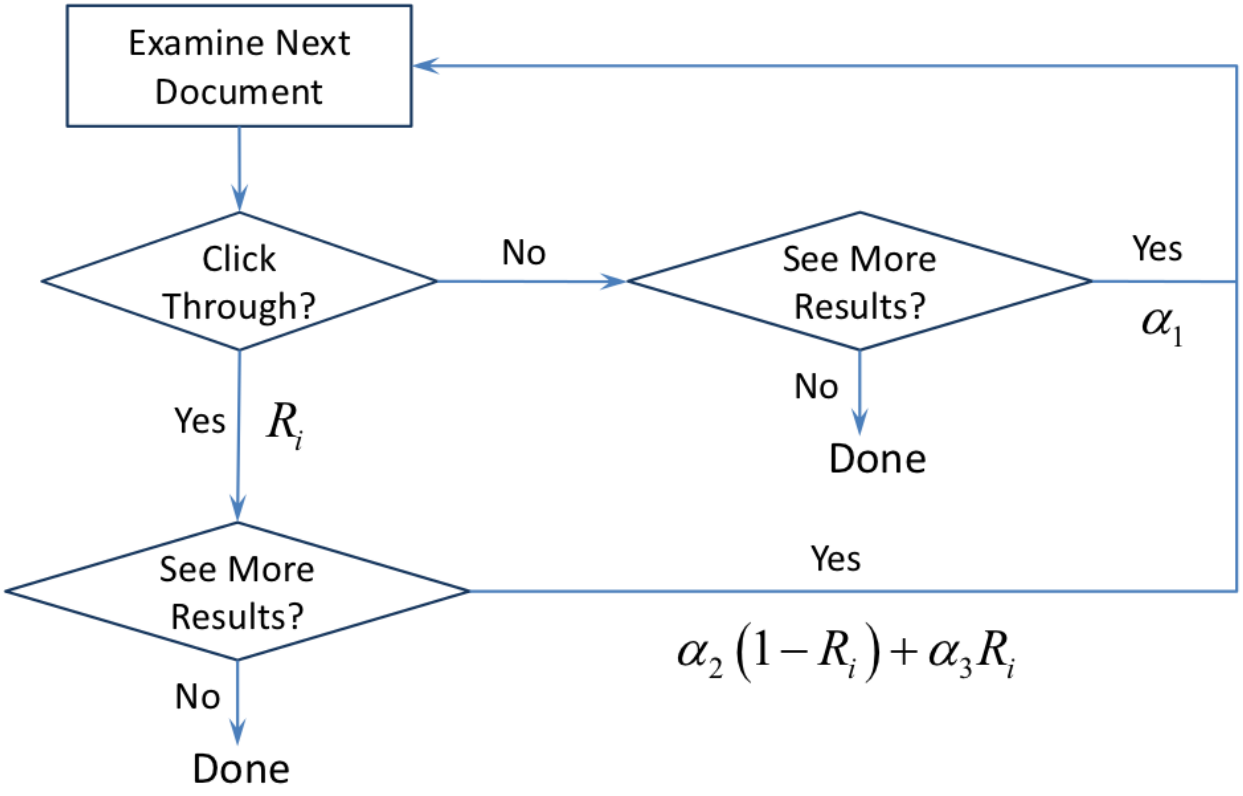

$\mathcal{N}(s_{1,1}; \mu_{1,1}, \sigma^2_{1,1})$
$\mathcal{N}(s_{1,2}; \mu_{1,2}, \sigma^2_{1,2})$
$\mathcal{N}(s_{2,1}; \mu_{2,1}, \sigma^2_{2,1})$
$\mathcal{N}(s_{3,1}; \mu_{3,1}, \sigma^2_{3,1})$
$\mathcal{N}(s_{4,1}; \mu_{3,1}, \sigma^2_{3,1})$
$\mathcal{N}(s_{4,2}; \mu_{4,2}, \sigma^2_{4,2})$
$s_{1,1}$
$s_{1,2}$
$s_{2,1}$
$s_{3,1}$
$s_{4,1}$
$s_{4,2}$
$\mathcal{N}(p_{1,1}; s_{1,1}, \beta^2)$
$\mathcal{N}(p_{1,2}; s_{1,2}, \beta^2)$
$\mathcal{N}(p_{2,1}; s_{2,1}, \beta^2)$
$\mathcal{N}(p_{3,1}; s_{3,1}, \beta^2)$
$\mathcal{N}(p_{4,1}; s_{4,1}, \beta^2)$
$\mathcal{N}(p_{4,2}; s_{4,2}, \beta^2)$
$p_{1,1}$
$p_{1,2}$
$p_{2,1}$
$p_{3,1}$
$p_{4,1}$
$p_{4,2}$
$\mathbb{I}(t_1 = \alpha_{1,0} + \alpha_{1,1}p_{1,1} + \alpha_{1,2}p_{1,2})$
$\mathbb{I}(t_1 = \alpha_{2,0} + \alpha_{2,1}p_{2,1})$
$\mathbb{I}(t_3 = \alpha_{3,0} + \alpha_{3,1}p_{3,1})$
$\mathbb{I}(t_3 = \alpha_{4,0} + \alpha_{4,1}p_{4,1} + \alpha_{4,2}p_{4,2})$
t_1
t_2
t_3
t_4
$\mathbb{I}(u_1 = l_1 - t_1)$
$\mathbb{I}(u_2 = l_2 - t_2)$
$\mathbb{I}(u_3 = l_2 - t_3)$
$\mathbb{I}(u_4 = l_3 - t_4)$
u_1
u_2
u_3
u_4
$\mathbb{I}(|u_1| \le \varepsilon)$
l_1
$\mathbb{I}(|u_2| \le \varepsilon)$
l_2
$\mathbb{I}(|u_3| \le \varepsilon)$
l_3
$\mathbb{I}(|u_4| \le \varepsilon)$
$\mathbb{I}(d_1 = l_1 - l_2)$
$\mathbb{I}(d_2 = l_2 - l_3)$
d_1
d_2

Постановка задачи

- Предположим, что мы – Google или Яндекс, и мы хотим улучшить качество поиска.
- Мы показываем пользователю документы по запросу и хотим показывать *релевантные* документы.
- Конечно, для этого полезно знать содержание документа, иначе никак не начать выдачу.
- Но когда мы уже накопили много терабайт логов, мы можем попробовать оценить качество выдачи по тому, как пользователи на неё реагировали.

Модель поведения пользователя

Модель поведения из [Guo et al., 2009].



Особенности

- То есть, по-русски если:
  1. пользователь читает сверху вниз;
  2. иногда кликает, когда кликнул, может вернуться обратно и читать дальше;
  3. по идее, самый хороший документ – это тот, на который пользователь кликнул последним.

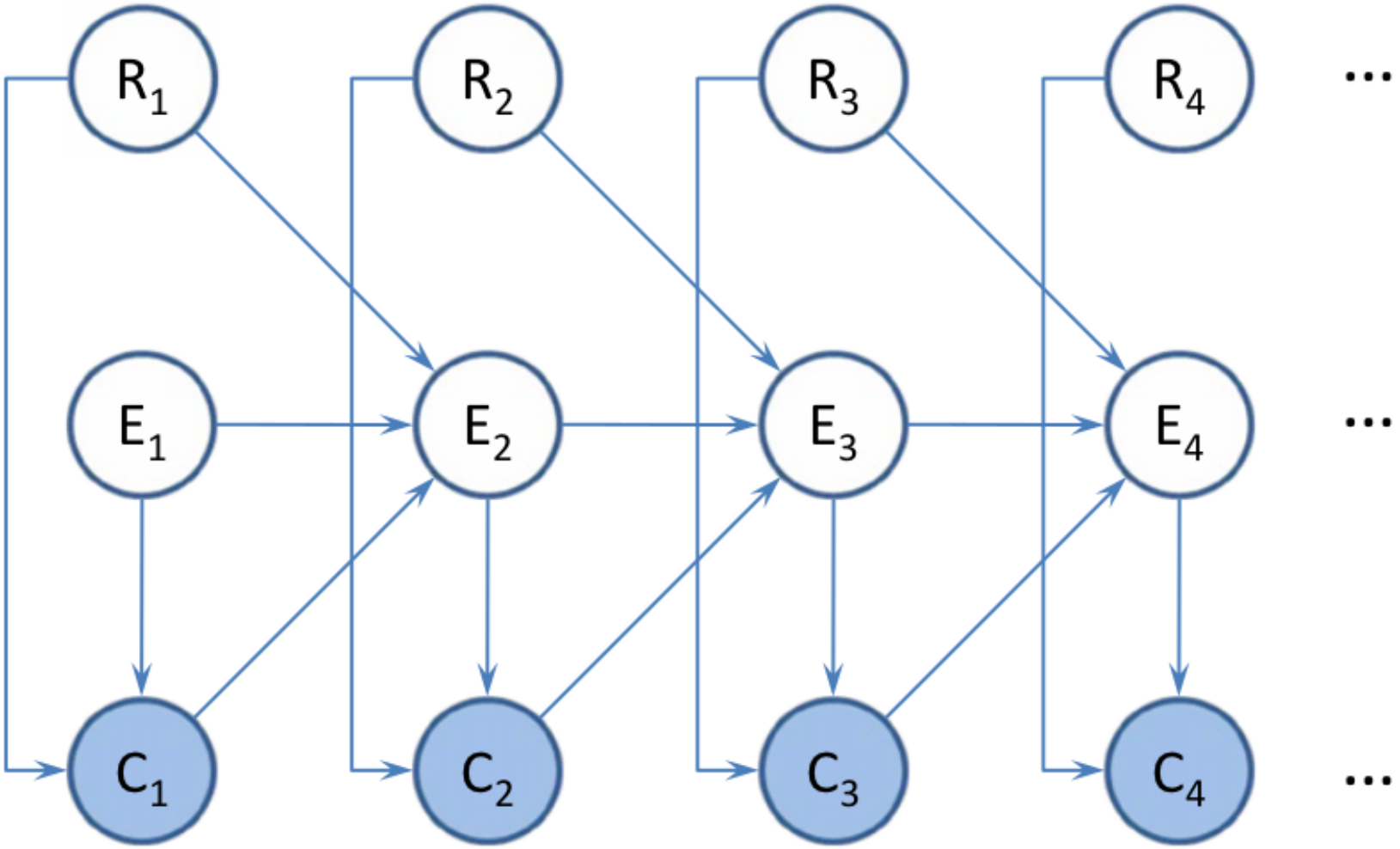
Структура модели

- Для каждого документа выдачи (1..10 обычно) введём переменные E_i (посмотрел ли; $E_1 = 1$ всегда) и C_i (кликнул ли).
- В CCM релевантность R_i – это вероятность клика, если посмотрел:

$$p(C_i = 1 \mid E_i = 0) = 0, \qquad p(C_i = 1 \mid E_i = 1, R_i) = R_i.$$

- Каскадная модель просмотра:
  - смотрим сверху вниз: $p(E_{i+1} = 1 \mid E_i = 0) = 0$, $p(E_1 = 1) = 1$;
  - если не кликнул, то вероятность посмотреть следующий – константа: $p(E_{i+1} = 1 \mid E_i = 1, C_i = 0) = \alpha_1$;
  - если кликнул, то вероятность посмотреть следующий зависит от релевантности линейно в каком-то интервале: $p(E_{i+1} = 1 \mid E_i = 1, C_i = 1) = \alpha_2(1 - R_i) + \alpha_3 R_i$.

Графическая модель



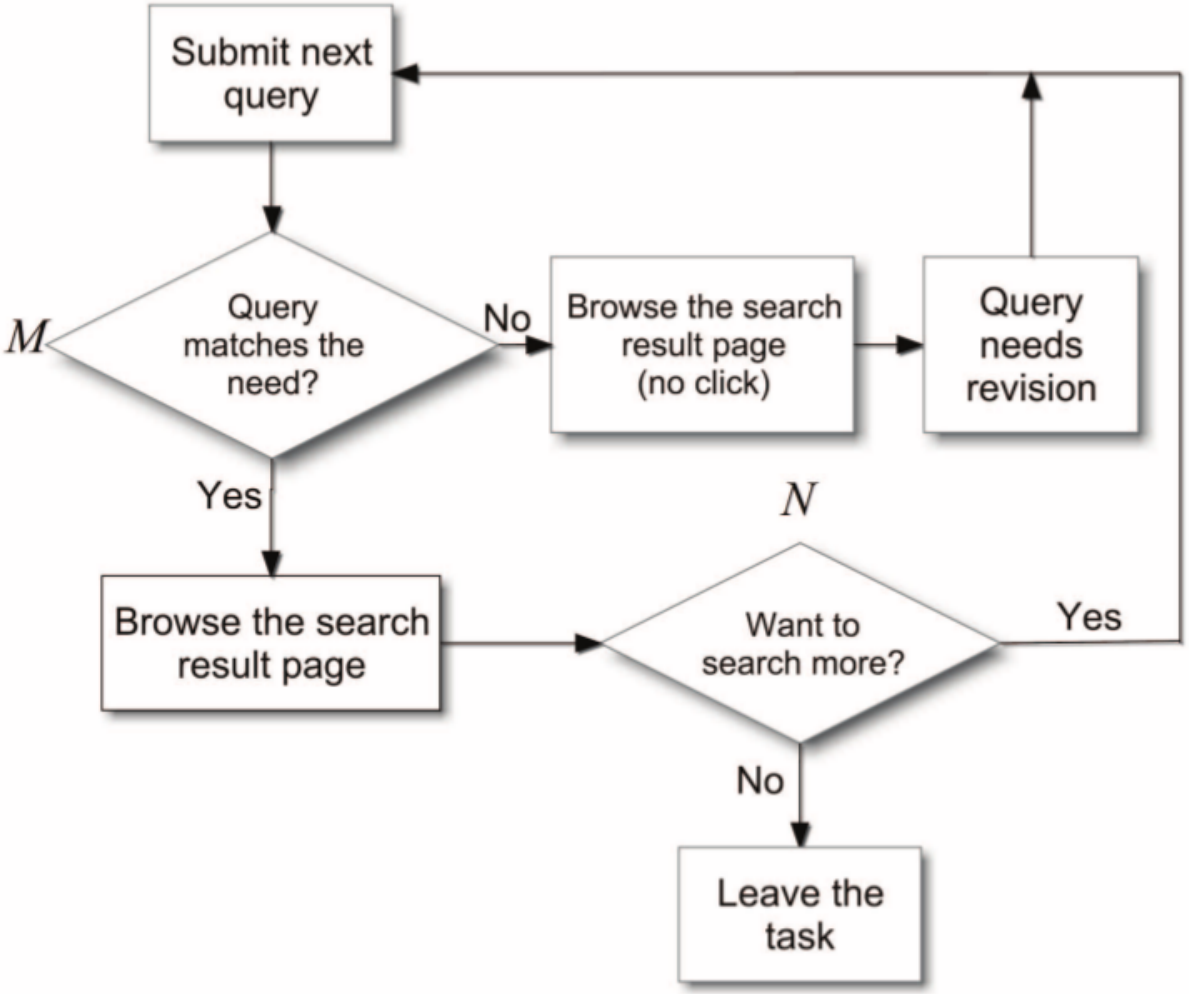
Суть идеи

- На самом деле разные клики в истории не независимы.
- Если пользователю не нравится выдача, он никуда кликать не будет, а переформулирует запрос.
- Если пользователь уже видел документ, то кликать на него в последующих выдачах он будет с меньшей вероятностью.
- Можно идентифицировать *сессии*, в течение которых пользователь что-то делал, задавал запросы, куда-то кликал, куда-то нет.

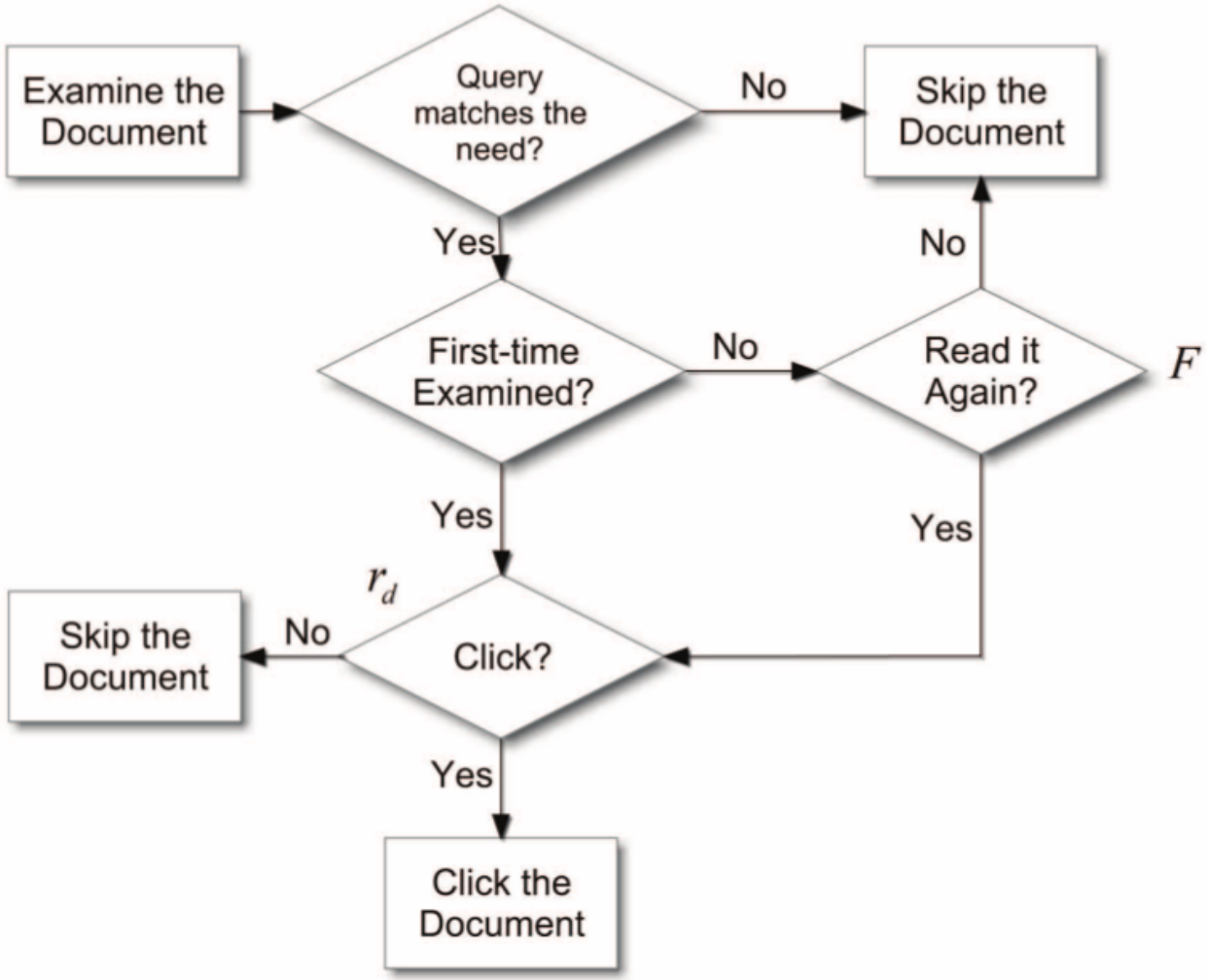
TCM

- TCM (task-centric click model) – это модель с двумя логическими «уровнями» [Zhang et al., 2011].
- На макроуровне пользователь переходит от запроса к запросу внутри сессии.
- Микроуровень, внутри запроса, похож на CCM, только ещё дополнительно обрабатываются повторы.

Макроуровень



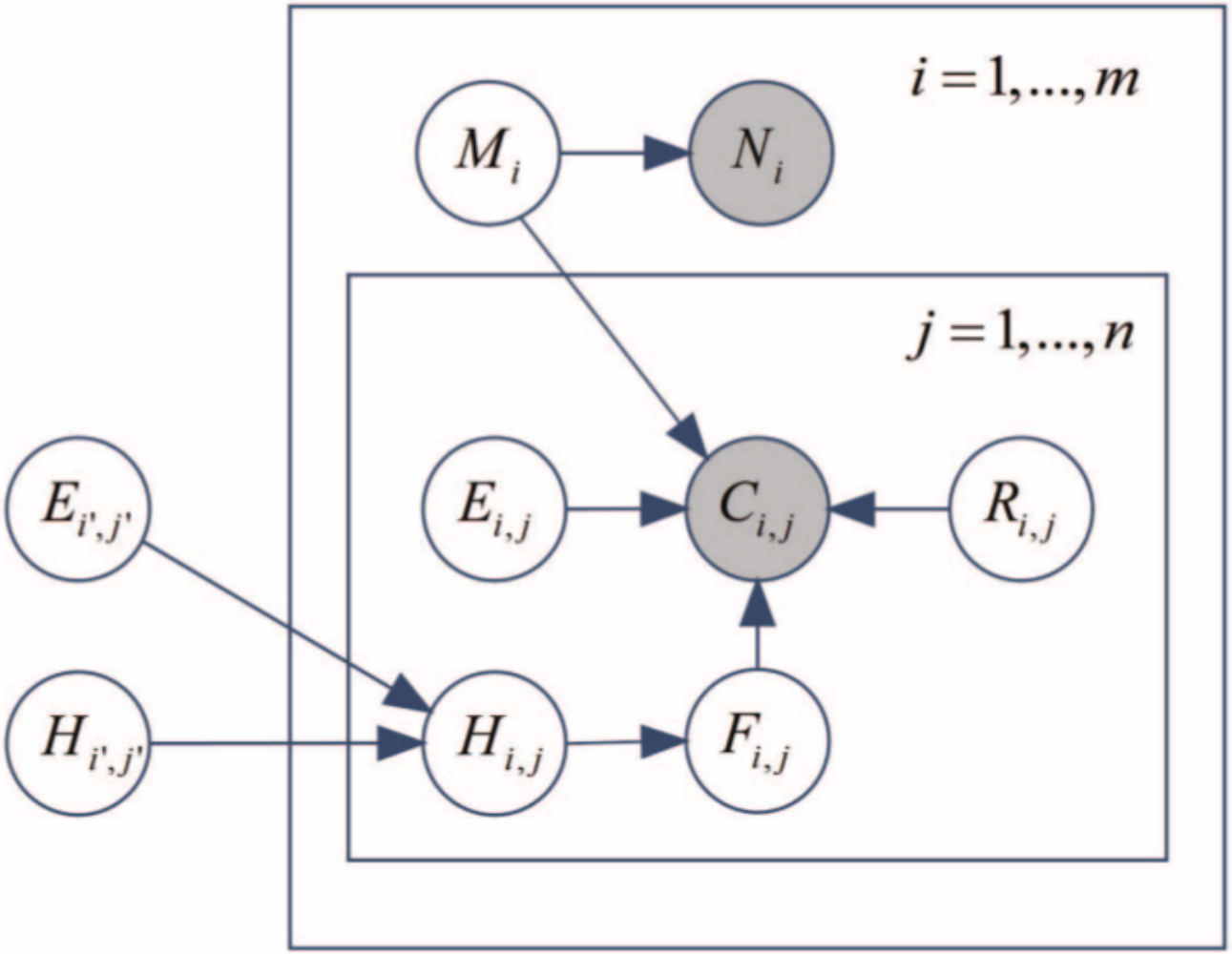
Микроуровень



Переменные TCM

- Вот какие переменные участвуют в TCM:
  - M_i – подходит ли i-й запрос для того, чего хотел пользователь;
  - N_i – будет ли пользователь ещё запрос подавать после i-го;
  - $E_{i,j}$ – просмотрит ли j-й документ в i-й выдаче;
  - $H_{i,j}$ – просматривал ли уже;
  - $F_{i,j}$ – «свежий» ли документ ($H_{i,j} = 0 \Rightarrow F_{i,j} = 1$, но свежим может быть и повтор);
  - $C_{i,j}$ – кликнет ли;
  - (i', j') – последнее вхождение того же самого документа в другой сессии.

Графическая модель



Thank you!

Спасибо за внимание!

LDA: вариационный вывод

- Теперь можно искать минимум KL-расстояния:

$$(\gamma^*, \phi^*) = \arg\min_{(\gamma,\phi)} \mathrm{KL}(q(\theta, z \mid \mathbf{w}, \gamma\phi) \| p(\theta, \mathbf{z} \mid \mathbf{w}, \alpha, \beta)).$$

- Для этого сначала воспользуемся уже известной оценкой из неравенства Йенсена:

$$\log p(\mathbf{w} \mid \alpha, \beta) = \log \int_\theta \sum_{\mathbf{z}} p(\theta, \mathbf{z}, \mathbf{w} \mid \alpha, \beta) d\theta =$$

$$= \log \int_\theta \sum_{\mathbf{z}} \frac{p(\theta, \mathbf{z}, \mathbf{w} \mid \alpha, \beta) q(\theta, \mathbf{z})}{q(\theta, \mathbf{z})} d\theta \geq$$

$$\geq E_q [\log p(\theta, \mathbf{z}, \mathbf{w} \mid \alpha, \beta)] - E_q [\log q(\theta, \mathbf{z})] =: \mathcal{L}(\gamma, \phi; \alpha, \beta).$$

LDA: вариационный вывод

- Распишем произведения:

$$\mathcal{L}(\gamma, \phi; \alpha, \beta) = E_q\left[p(\theta \mid \alpha)\right] + E_q\left[p(\mathbf{z} \mid \theta)\right] + E_q\left[p(\mathbf{w} \mid \mathbf{z}, \beta)\right] - \\ - E_q\left[\log q(\theta)\right] - E_q\left[\log q(\mathbf{z})\right].$$

- Свойство распределения Дирихле: если $X \sim \mathrm{Di}(\alpha)$, то

$$E[\log(X_i)] = \Psi(\alpha_i) - \Psi(\sum_i \alpha_i),$$

где $\Psi(x) = \frac{\Gamma'(x)}{\Gamma(x)}$ – дигамма-функция.

- Теперь можно выписать каждый из пяти членов.

LDA: вариационный вывод

$$\mathcal{L}(\gamma, \phi; \alpha, \beta) = \log \Gamma(\sum_{i=1}^{k} \alpha_i) - \sum_{i=1}^{k} \log \Gamma(\alpha_i) + \sum_{i=1}^{k} (\alpha_i - 1) \left[\Psi(\gamma_i) - \Psi\left(\sum_{j=1}^{k} \gamma_j \right) \right] +$$

$$+ \sum_{n=1}^{N} \sum_{i=1}^{k} \phi_{ni} \left[\Psi(\gamma_i) - \Psi\left(\sum_{j=1}^{k} \gamma_j \right) \right] +$$

$$+ \sum_{n=1}^{N} \sum_{i=1}^{k} \sum_{j=1}^{V} w_n^j \phi_{ni} \log \beta_{ij} -$$

$$- \log \Gamma(\sum_{i=1}^{k} \gamma_i) + \sum_{i=1}^{k} \log \Gamma(\gamma_i) - \sum_{i=1}^{k} (\gamma_i - 1) \left[\Psi(\gamma_i) - \Psi\left(\sum_{j=1}^{k} \gamma_j \right) \right] -$$

$$- \sum_{n=1}^{N} \sum_{i=1}^{k} \phi_{ni} \log \phi_{ni}.$$

LDA: вариационный вывод

- Теперь осталось только брать частные производные этого выражения.
- Сначала максимизируем его по ϕ_{ni} (вероятность того, что n-е слово было порождено темой i); надо добавить λ-множители Лагранжа, т.к. $\sum_{j=1}^{k} \phi_{nj} = 1$.
- В итоге получится:

$$\phi_{ni} \propto \beta_{iv} e^{\Psi(\gamma_i) - \Psi\left(\sum_{j=1}^{k} \gamma_j\right)},$$

где v – номер того самого слова, т.е. единственная компонента $w_n^v = 1$.

LDA: вариационный вывод

- Потом максимизируем по γ_i, i-й компоненте апостериорного Дирихле-параметра.
- Получится
$$\gamma_i = \alpha_i + \sum_{n=1}^{N} \phi_{ni}.$$
- Соответственно, для вывода нужно просто пересчитывать ϕ_{ni} и γ_i друг через друга, пока оценка не сойдётся.

LDA: оценка параметров

- Теперь давайте попробуем оценить параметры α и β по корпусу документов $\mathcal{D}$.
- Мы хотим найти α и β, которые максимизируют

$$\ell(\alpha, \beta) = \sum_{d=1}^{M} \log p(\mathbf{w}_d \mid \alpha, \beta).$$

- Подсчитать $p(\mathbf{w}_d \mid \alpha, \beta)$ мы не можем, но у нас есть нижняя оценка $\mathcal{L}(\gamma, \phi; \alpha, \beta)$, т.к.

$$p(\mathbf{w}_d \mid \alpha, \beta) =$$
$$= \mathcal{L}(\gamma, \phi; \alpha, \beta) + \mathrm{KL}(q(\theta, z \mid \mathbf{w}_d, \gamma\phi) \| p(\theta, \mathbf{z} \mid \mathbf{w}_d, \alpha, \beta)).$$

LDA: оценка параметров

- EM-алгоритм:
  1. найти параметры $\{\gamma_d, \phi_d \mid d \in \mathcal{D}\}$, которые оптимизируют оценку (как выше);
  2. зафиксировать их и оптимизировать оценку по α и β.

LDA: оценка параметров

- Для β это тоже делается нехитро:

$$\beta_{ij} \propto \sum_{d=1}^{M} \sum_{n=1}^{N_d} \phi_{dni} w_n^j.$$

- Для α_i получается система уравнений, которую можно решить методом Ньютона.