

Large-Scale Machine Learning (LSML)

О курсе

Лекции

1. Онлайн-обучение и линейные модели
2. Введение в Apache Spark
3. Рекомендательные системы
4. Градиентный бустинг для больших данных
5. Введение в TensorFlow
6. Нейросети для классификации картинок
7. Нейросети для классификации текстов
8. LSH для поиска похожих объектов
9. Кластеризация больших данных

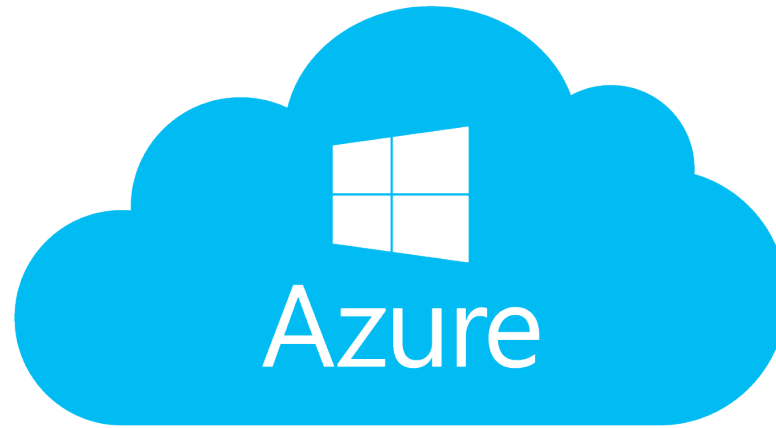


Домашние задания

1. Рекомендательная система на Apache Spark
2. Реализация градиентного бустинга над решающими деревьями на Apache Spark
3. Аннотирование картинок на TensorFlow

Домашние задания

- Выполняются на виртуальных машинах в облаке Azure



- У каждого студента:
 - Свой логин/пароль в Azure, будут выданы позже
 - Свой Hadoop-кластер с Apache Spark
 - Своя машина с GPU и TensorFlow
 - Дается 2 недели использования ресурсов на каждое задание

Формула оценки

- Результирующая оценка по дисциплине рассчитывается по формуле

$$O_{\text{итог}} = \mathbf{0.7} O_{\text{накопл}} + \mathbf{0.3} O_{\text{экз}}$$

- Накопленная оценка рассчитывается по формуле

$$O_{\text{накопл}} = \mathbf{0.3} O_{\text{дз1}} + \mathbf{0.3} O_{\text{дз2}} + \mathbf{0.4} O_{\text{дз3}}$$

- Накопленная и итоговая оценки округляются арифметически.

Экзамен

- Устный экзамен
- Задачи на листочке
- Лучше хорошо делать ДЗ и получить автомат ($O_{\text{накопл}} > \text{порога}$)

LSML #1

Онлайн-обучение и линейные модели

Онлайн-обучение

Онлайн-обучение – данные поступают последовательно, улучшаем модель после каждого нового примера

Когда используется:

- Невозможно обучаться на всей выборке
- Вся выборка не помещается в память
- Нужно быстро адаптироваться к новым зависимостям в данных

На примере линейной регрессии

- Обучение на всей выборке:

$$w^* = (X^T X)^{-1} X^T Y$$

- Можно обновлять веса рекурсивно:

$$\begin{aligned} w_0 &= 0 \in \mathbb{R}^d & \Gamma_i &= \Gamma_{i-1} - \frac{\Gamma_{i-1} x_i x_i^T \Gamma_{i-1}}{1 + x_i^T \Gamma_{i-1} x_i} \\ \Gamma_0 &= I \in \mathbb{R}^{d \times d} & w_i &= w_{i-1} - \Gamma_i x_i (x_i^T w_{i-1} - y_i) \end{aligned}$$

- Стохастический градиентный спуск (SGD):

$$w_i = w_{i-1} - \gamma_i x_i (x_i^T w_{i-1} - y_i)$$

Похожи



Vowpal Wabbit (VW)

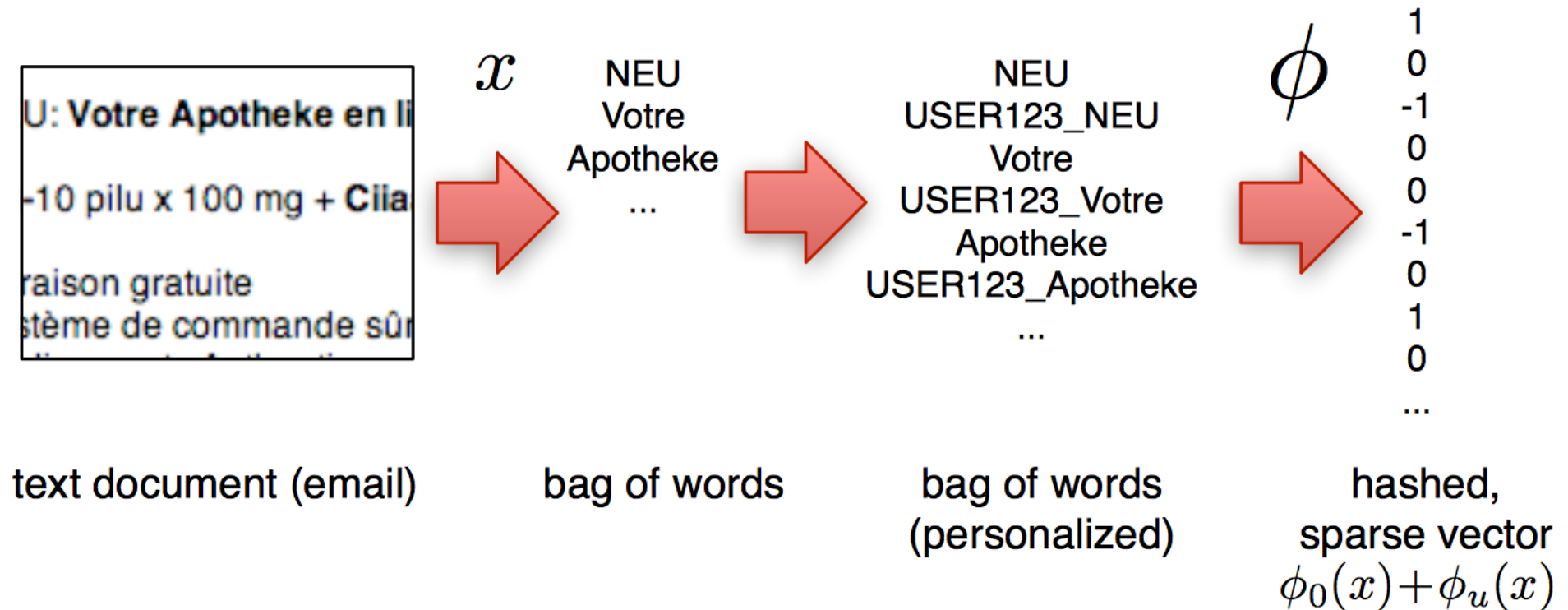
- Открытая реализация алгоритмов онлайн-обучения
- Не только линейные модели (парные взаимодействия, нейросети)
- Много функций потерь
- Обычная строка текста – это валидное описание объекта. Применяет хэширование признаков.
 - 1 | The dog ate my homework
- Эффективно масштабируется на 1000 машин при помощи AllReduce.
- Может работать в режиме с обратной связью (Contextual Bandit)
- Может работать в режиме активного обучения
- ...

Хэширование признаков

- Требуется one-hot кодирование признаков
 - Категориальные признаки
 - Слова текста в модели мешка слов
- Храним “word” \rightarrow index
 - Словарь должен быть общим для всех машин
 - Может не поместиться в RAM
- Считаём “word” \rightarrow hash(“word”)
 - Большое число корзинок хэш-функции ($\sim 2^{24}$), качество растёт по $\log(\text{hash bits})$
 - Значительно быстрее, не занимает памяти и легко распараллеливается

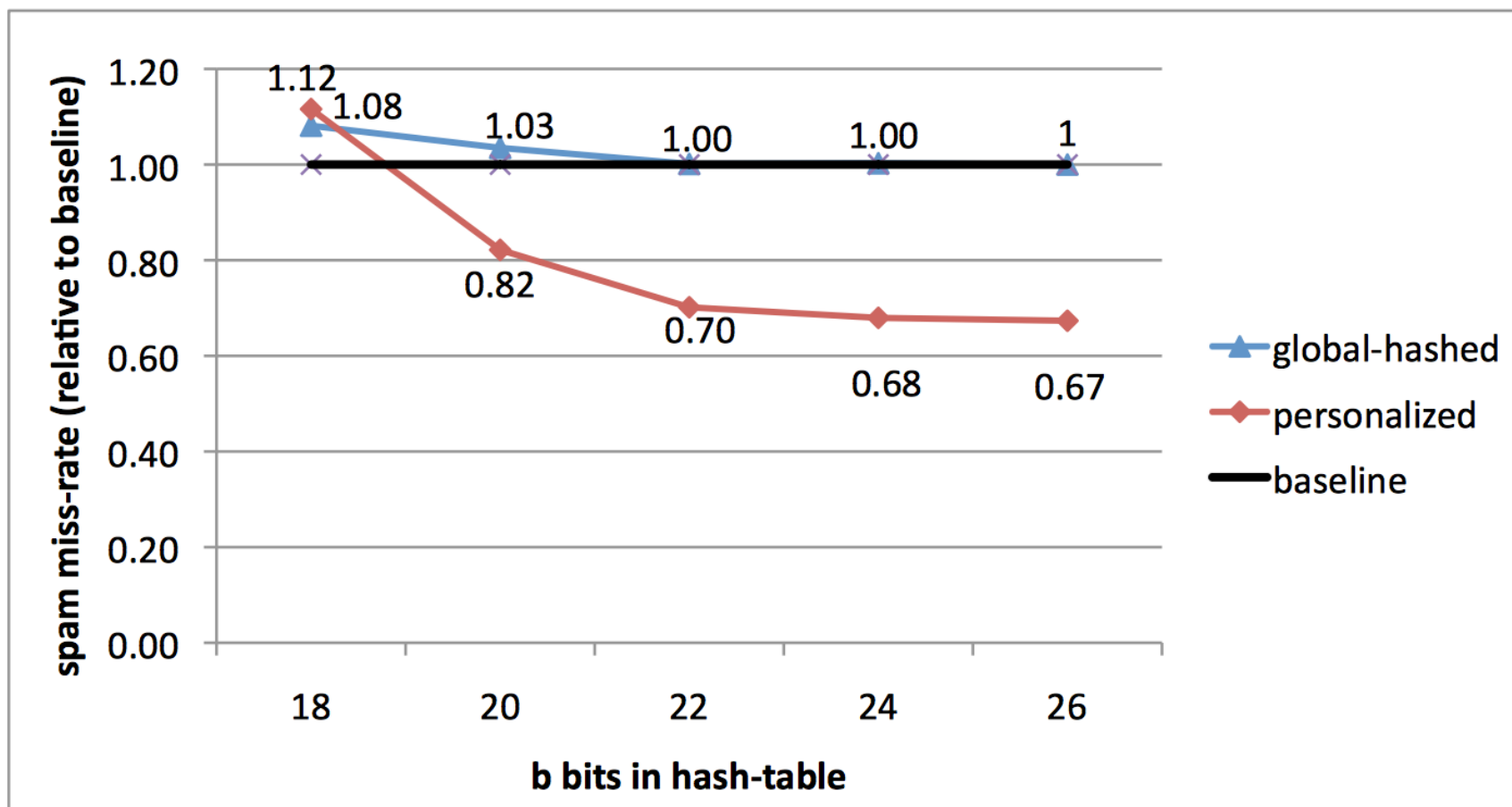
Хэширование в детекции спама

- 0.4 млн пользователей, 3.2 млн писем, 40 млн слов
- 16 трлн пар (пользователь, слово) – поможет только хэширование



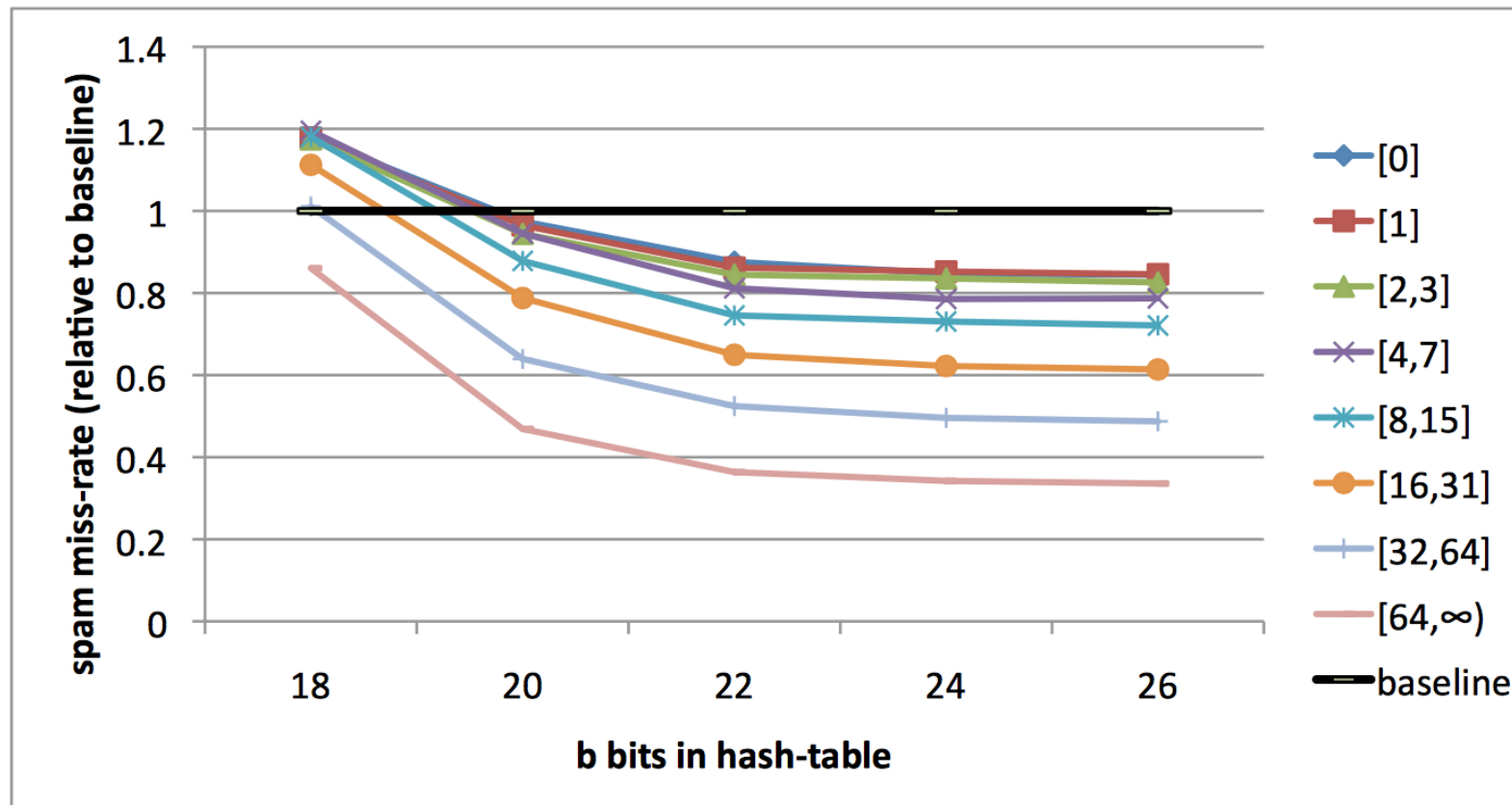
Хэширование в детекции спама

- Хэшированные **16 трлн признаков** дают существенный прирост качества
- Хэширование перестает влиять на качество **не персональной** модели



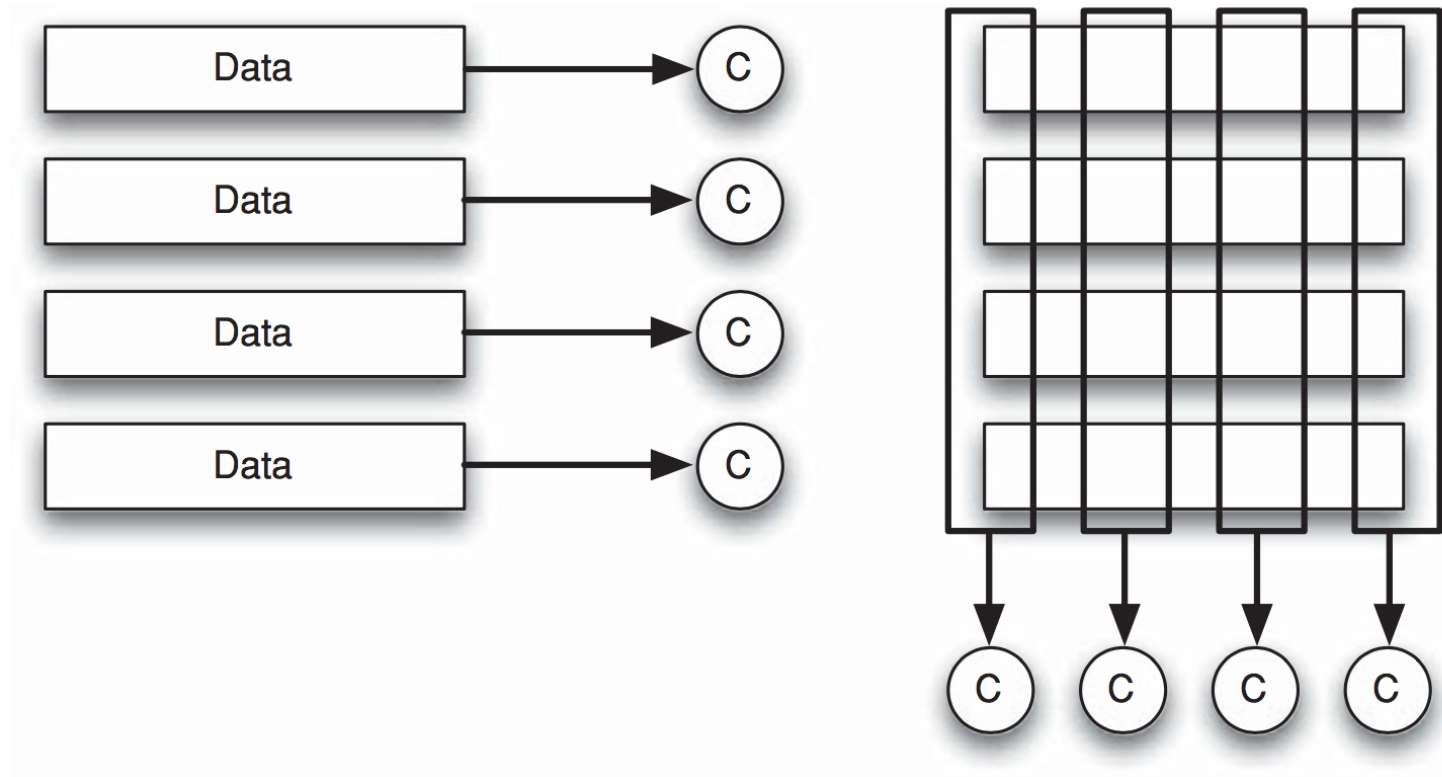
Хэширование в детекции спама

- Хорошо работает даже на пользователях, которых не было в обучении!
- Гипотеза: все «локальные» зависимости были учтены новыми признаками, «глобальные» зависимости стали более универсальными



Количество
писем
пользователя
в обучении

Как можно распараллелить работу VW



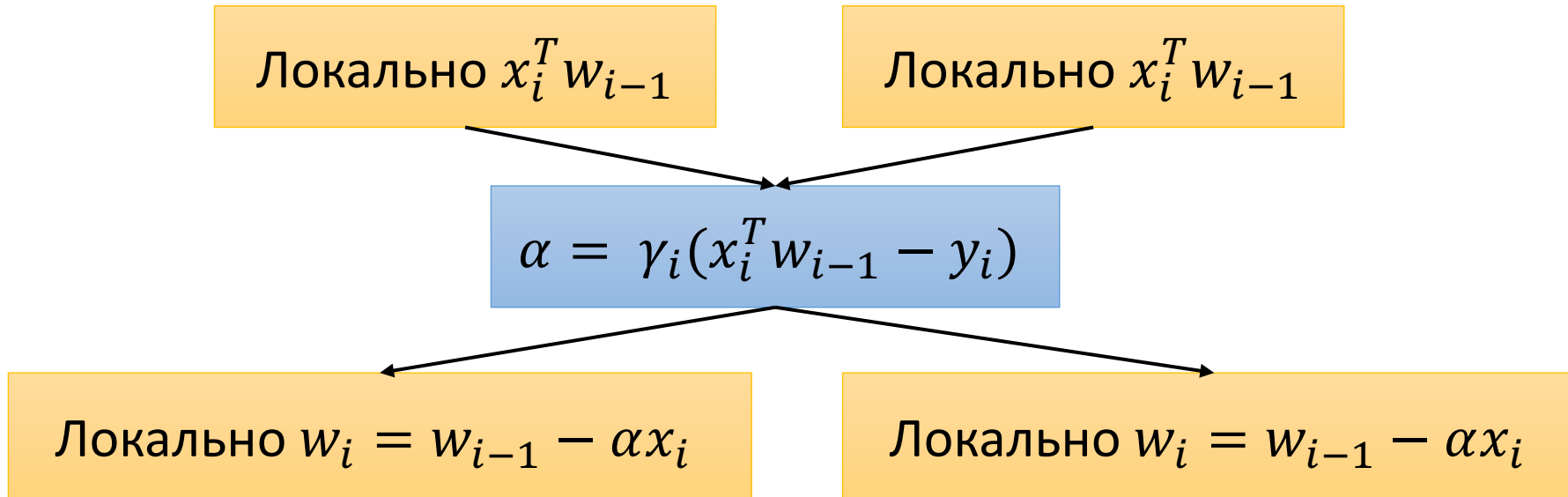
Делим объекты

Делим признаки

Делим признаки на многоядерной машине

- Нужно сделать шаг:

$$w_i = w_{i-1} - \gamma_i x_i (x_i^T w_{i-1} - y_i)$$

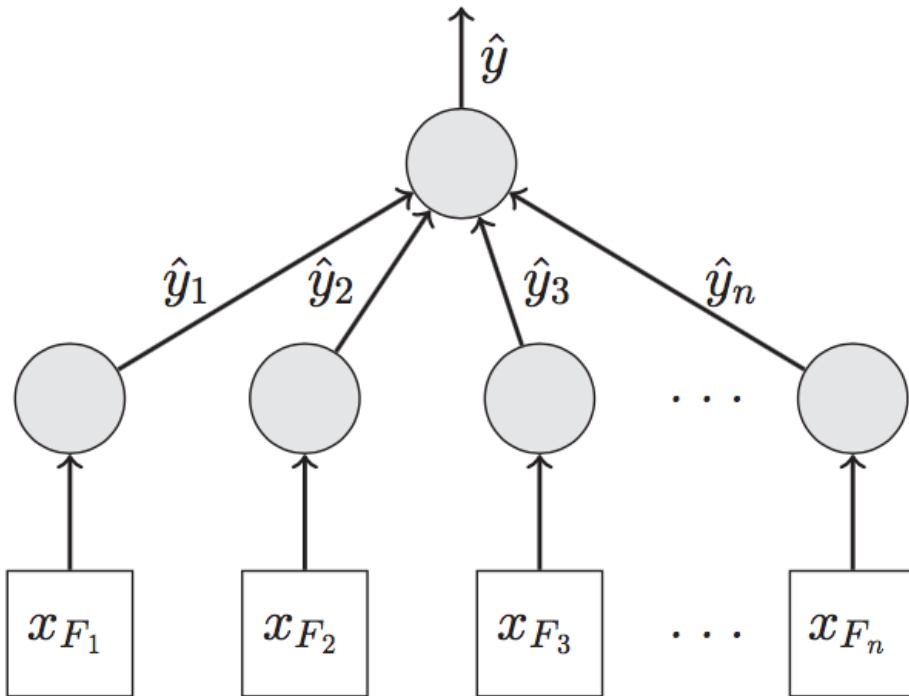


- Результат аналогичен однопоточному варианту
- На 4 ядрах дает ускорение в 3 раза, дальше слабо масштабируется

Делим признаки на много машин

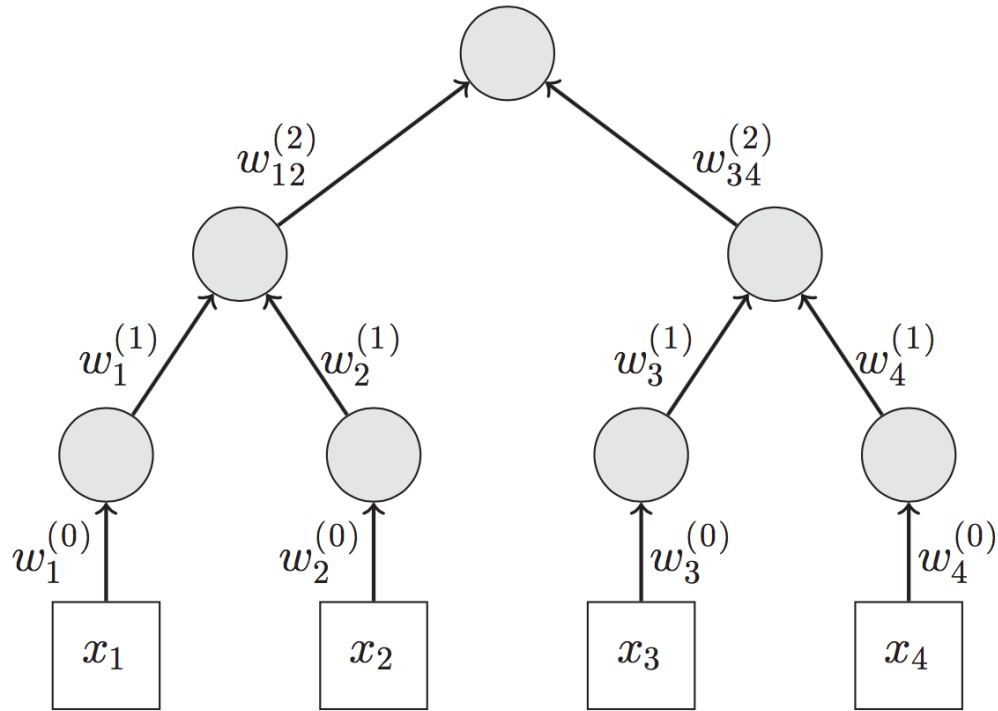
- Нужно сделать шаг:

$$w_i = w_{i-1} - \gamma_i x_i (x_i^T w_{i-1} - y_i)$$



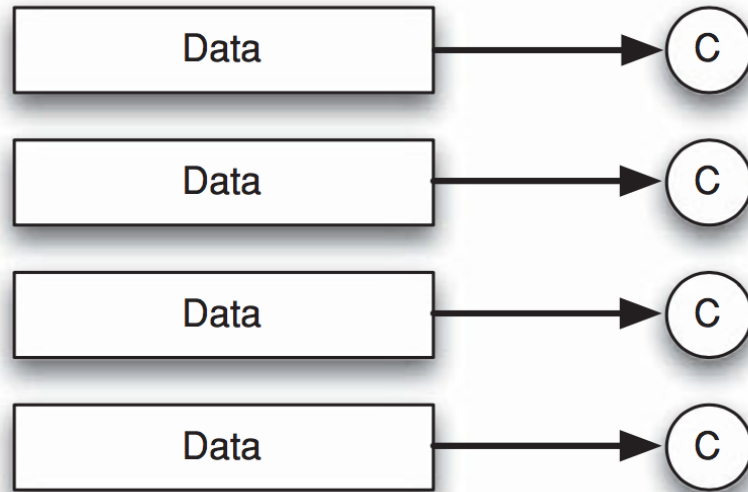
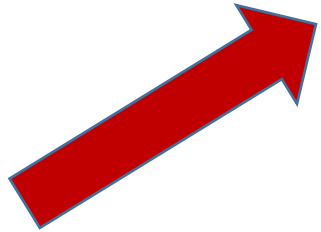
- На каждой машине учится модель, которая использует только свою часть признаков.
- Предсказания пересылаются на мастер машину, которая учит линейную модель, итоговая модель – линейная.
- Отличается от back-propagation, нет пересылки градиентов, модели учатся независимо.

Делим признаки на много машин – go deeper!

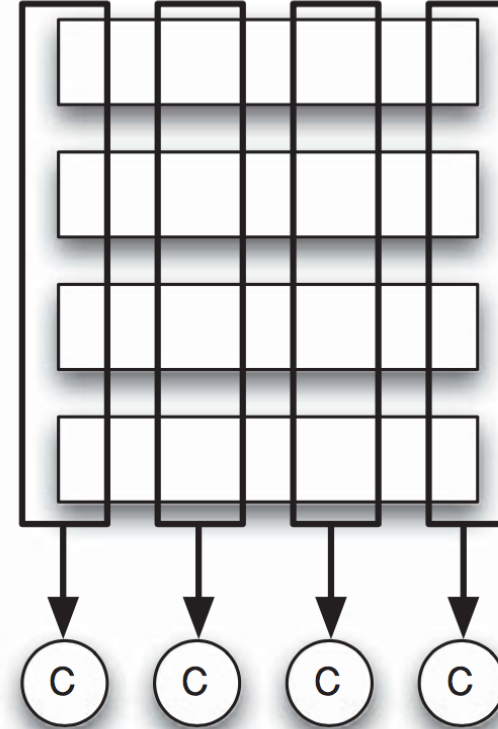


- По выразительной способности между Наивным Байесом и линейными моделями.
- Чем больше слоев, тем более выразительная модель, но больше задержка предсказания. Нужно искать баланс глубины и скорости.
- Не масштабируется линейно, время обмена больше времени вычисления.
- В экспериментах может проигрывать линейной модели по качеству.

Надо делить объекты на много машин!



Делим объекты



Делим признаки

Поможет Hadoop

- **Hadoop** – проект Apache для распределенных вычислений
- **Hadoop YARN** – планировщик задач и система для управления ресурсами кластера
- **Hadoop MapReduce** – система для вычислений в парадигме Map-Reduce
- **HDFS** – распределенная файловая система Hadoop

Особенности HDFS:

- Файлы хранятся **блоками** на разных машинах
- Каждый **блок дублируется** на нескольких разных машинах (replication factor)
- Информация о соответствии **путь файла → его блоки** хранится на специальной машине **Name Node** в RAM

Парадигма Map-Reduce на примере Word Count

Шаг Map:

$(K1, V1) \rightarrow \text{List}(K2, V2)$

$(\text{\#строки}, \text{"Deer Bear"}) \rightarrow [(\text{"Deer"}, 1), (\text{"Bear"}, 1)]$

Шаг Shuffle (или Sort):

Shuffle делит данные по $\text{hash}(\text{key}) \% N$ на N частей

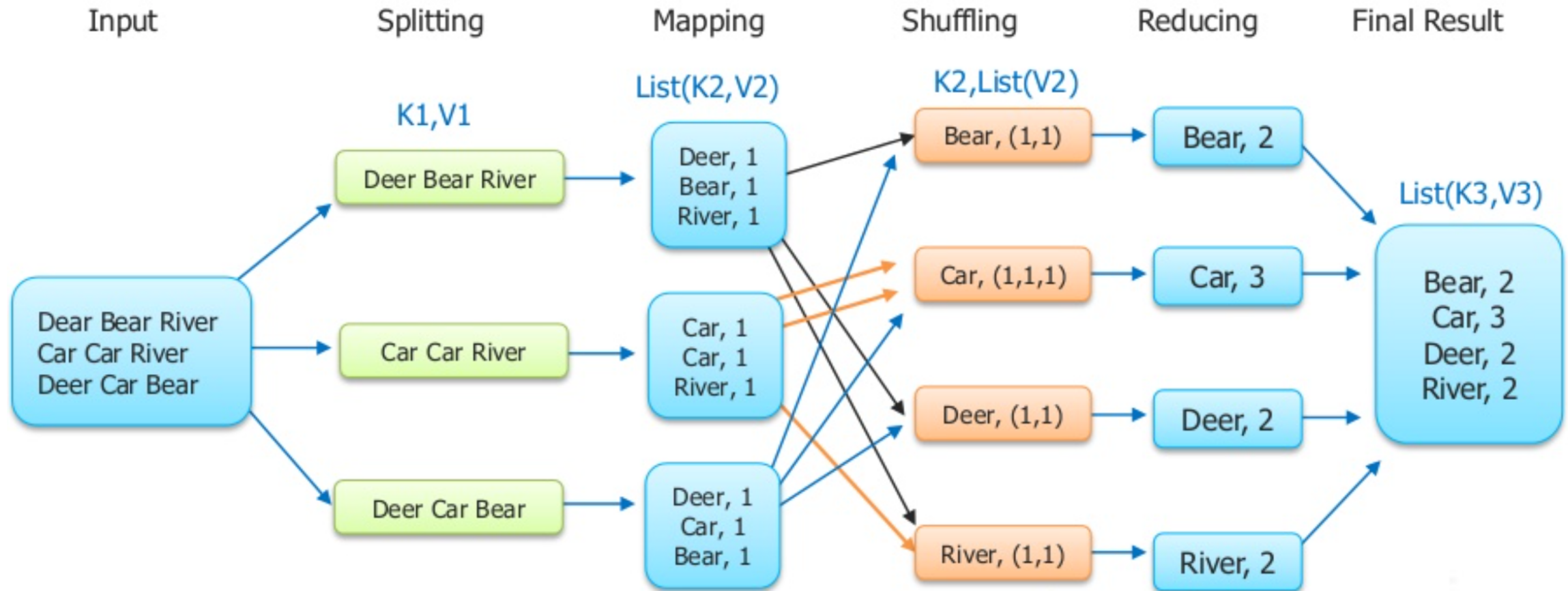
Sort сортирует данные по **key** и делит на N частей (по границам **key**)

Шаг Reduce:

$(K1, (V1, V2, \dots)) \rightarrow \text{List}(K3, V3)$

$(\text{"Bear"}, (1, 1)) \rightarrow [(\text{"Bear"}, 2)]$

Парадигма Map-Reduce на примере Word Count



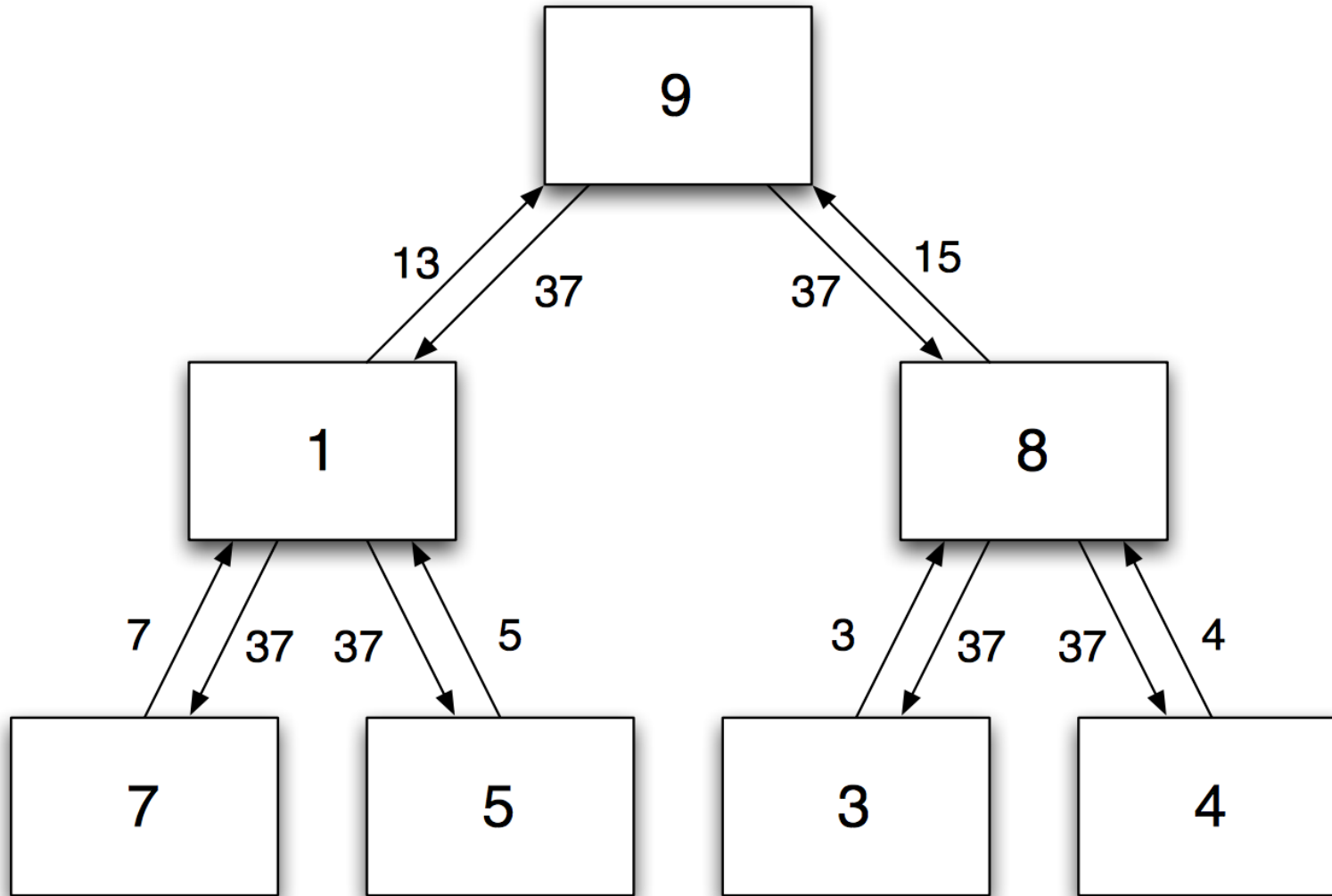
Файл в HDFS

Разбиение блоков
для Map-задач

Пересылка по сети
по $\text{hash}(\text{key}) \% N$,
 N – кол-во Reduce-задач

Файл в HDFS

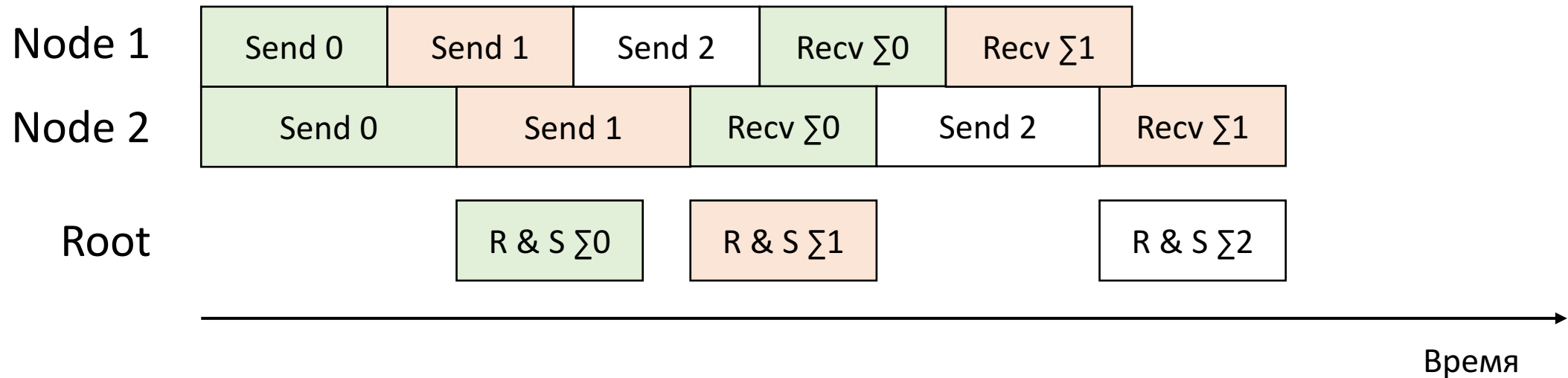
AllReduce в VW



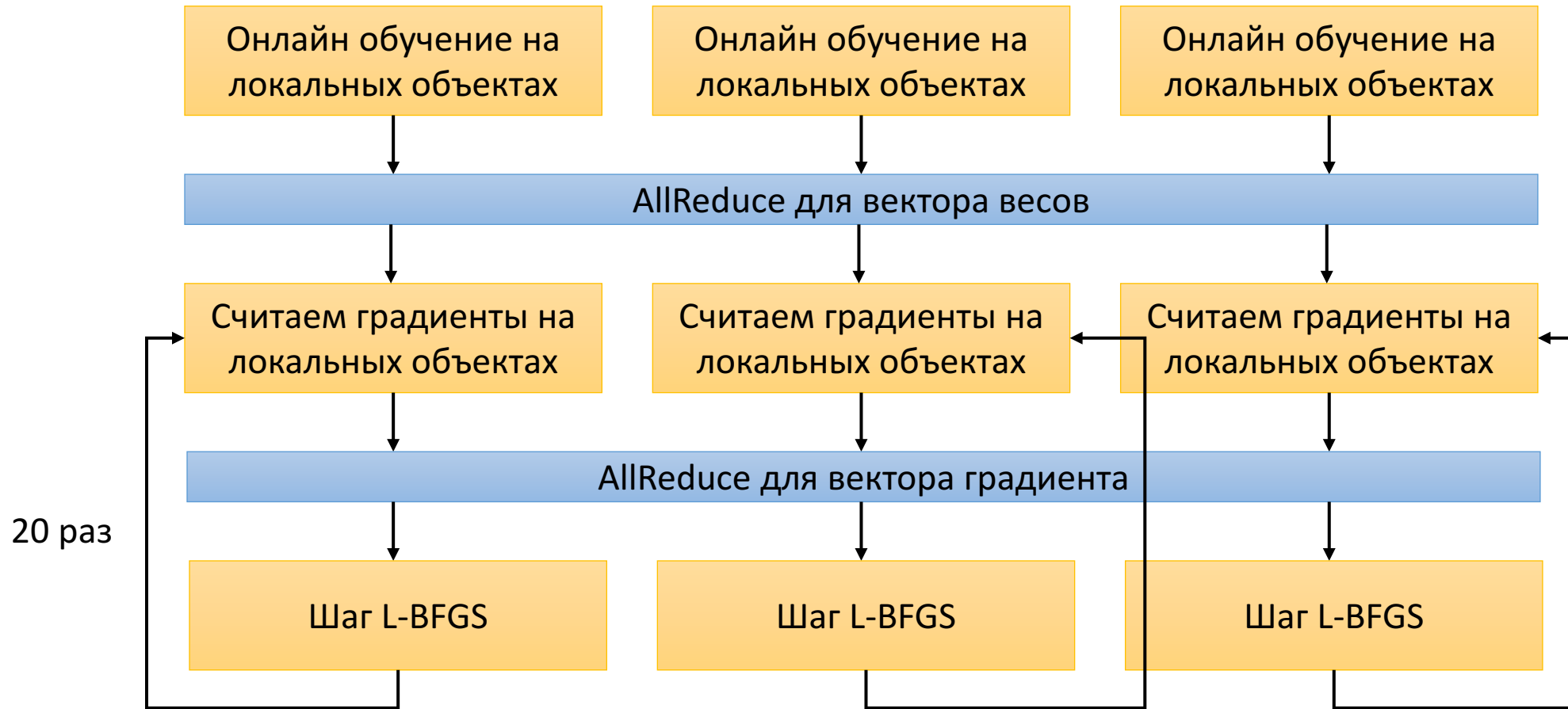
- Каждая машина считает вектор (градиент или веса) по своим объектам
- На каждой машине хочется получить сумму этих векторов от всех машин

Конвейеризация в AllReduce

- Не надо передавать весь вектор целиком и ждать, когда мастер машина посчитает сумму.
- Будем отправлять вектор поэлементно (или пачками) и по ходу получать обновленные значения.



Гибридная схема работы распределенного VW



Онлайн обучение на локальных объектах

- Map-задачи в Hadoop
- SGD algorithm using adaptive gradient update:

Require: Invariance update function s
(see Karampatziaakis and Langford, 2011)

$\mathbf{w} = \mathbf{0}, \mathbf{G} = \mathbf{I}$

for all (\mathbf{x}, y) in training set **do**

$\mathbf{g} \leftarrow \nabla_{\mathbf{w}} \ell(\mathbf{w}^\top \mathbf{x}; y)$

$\mathbf{w} \leftarrow \mathbf{w} - s(\mathbf{w}, \mathbf{x}, y) \mathbf{G}^{-1/2} \mathbf{g}$

$G_{jj} \leftarrow G_{jj} + g_j^2$ for all $j = 1, \dots, d$

end for

AllReduce для вектора весов

- Пусть у нас m машин, тогда средний вектор весов будем считать так:

$$\bar{\mathbf{w}} = \left(\sum_{k=1}^m \mathbf{G}^k \right)^{-1} \left(\sum_{k=1}^m \mathbf{G}^k \mathbf{w}^k \right)$$

- G^k — диагональные матрицы, диагональ такого же размера как w^k
- Две операции AllReduce. Работает быстрее агрегации при помощи MapReduce:

	Full size	10% sample
MapReduce	1690	1322
AllReduce	670	59

Градиентный спуск

Gradient descent update:

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \nabla f(\mathbf{w}_t)$$

gradient

$$\mathbf{g}_t = \nabla f(\mathbf{w}_t)$$

Equivalently:

- approximate

$$f(\mathbf{w}) \approx f(\mathbf{w}_t) + \mathbf{g}_t^T (\mathbf{w}_t - \mathbf{w}) + \frac{1}{2\eta} \|\mathbf{w}_t - \mathbf{w}\|^2$$

- optimize approximation:

$$\mathbf{w}_{t+1} = \underset{\mathbf{w}}{\operatorname{argmin}} \left(f(\mathbf{w}_t) + \mathbf{g}_t^T (\mathbf{w}_t - \mathbf{w}) + \frac{1}{2\eta} \|\mathbf{w}_t - \mathbf{w}\|^2 \right)$$

Can we replace quadratic term by a tighter approximation?

Метод Ньютона

Hessian

$$\mathbf{H}_t = \nabla^2 f(\mathbf{w}_t)$$

Better approximation

$$f(\mathbf{w}) \approx f(\mathbf{w}_t) + \mathbf{g}_t^\top (\mathbf{w}_t - \mathbf{w}) + \frac{1}{2} (\mathbf{w}_t - \mathbf{w})^\top \mathbf{H}_t (\mathbf{w}_t - \mathbf{w})$$

Update:

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \mathbf{H}_t^{-1} \mathbf{g}_t$$

Problem: Hessian can be too big (matrix of size $d \times d$)

L-BFGS

Instead of the Newton update

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \mathbf{H}_t^{-1} \mathbf{g}_t$$

Perform a *quasi-Newton* update:

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta_t \mathbf{K}_t \mathbf{g}_t$$

where: \mathbf{K}_t is a low-rank approximation of \mathbf{H}_t^{-1}
 η_t is obtained by line search

- rank m specified by user (default $m=15$)
- instead of storage d^2 , only storage $2dm$ required
(update of \mathbf{K}_t also has running time $O(dm)$ per iteration)

L-BFGS B VW

--bfgs

turn on LBFGS optimization

--l2 0.0

L2 regularization coefficient

--mem 15

rank of the inverse Hessian approximation

--termination 0.001

termination threshold for the
relative loss decrease

VW выводит ошибку модели

- Стандартный путь: обучение на train, качество на test
- При одном проходе по данным достаточно **progressive validation**
 - Для каждого нового объекта считается ошибка до обновления параметров
 - Выводится средняя ошибка по таким объектам
 - Доказано, что приближает ошибку на тесте
- При втором проходе по данным не будет иметь смысла
 - VW сам откладывает 10% данных в holdout, если указали больше одного прохода по данным (эпохи)
 - Можно отключить при помощи `--holdout_off`

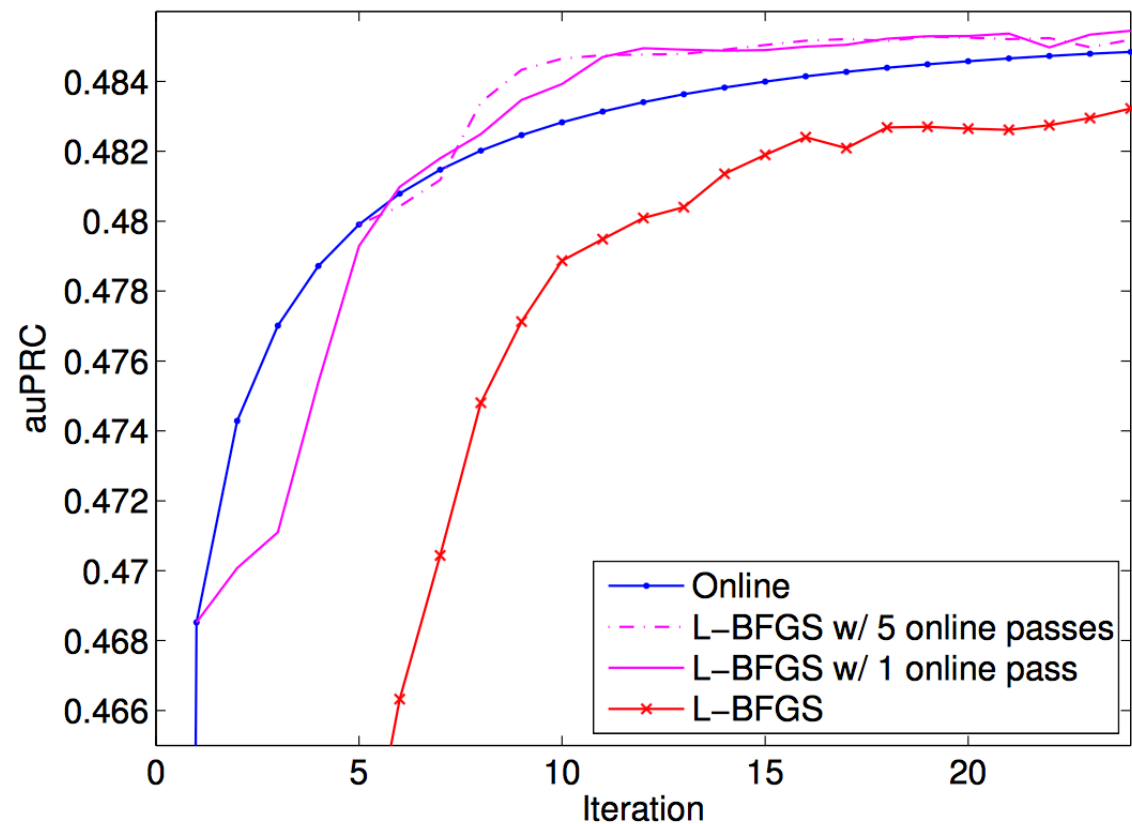
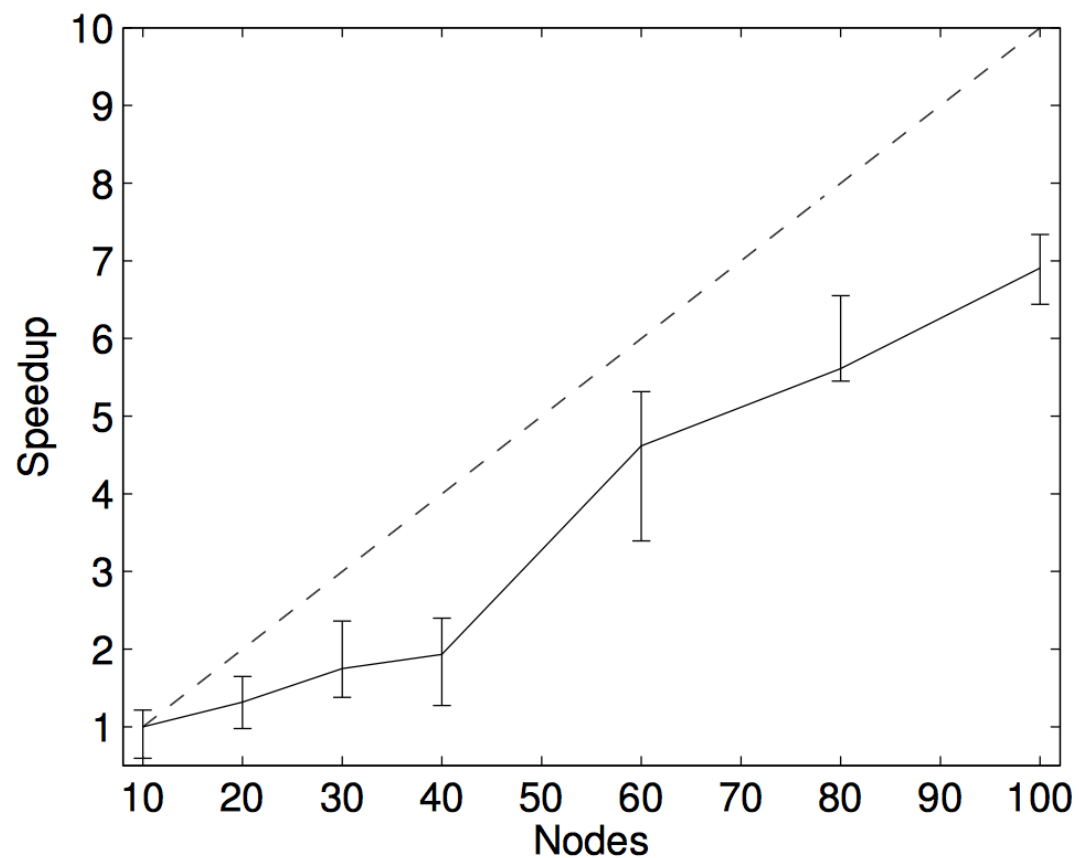
Задача предсказания кликов по рекламе

- **2.1T** разреженных признаков
- **17B** примеров
- **16M** параметров (24 hash bits)
- **1K** машин
- Оптимальный линейный классификатор за **70 минут**

	1%	10%	100%
auROC	0.8178	0.8301	0.8344
auPRC	0.4505	0.4753	0.4856
NLL	0.2654	0.2582	0.2554

Сэмплирование ухудшает качество –
нужно учиться на всех данных!

Задача предсказания кликов по рекламе



Ссылки

- A Reliable Effective Terascale Linear Learning System
<https://arxiv.org/pdf/1110.4198.pdf>
- <http://cilvr.cs.nyu.edu/diglib/lsm/lecture01-online-linear.pdf>
- https://github.com/JohnLangford/vowpal_wabbit/wiki/Tutorial
- <http://www.zinkov.com/posts/2013-08-13-vowpal-tutorial/>
- <http://mlwave.com/predicting-click-through-rates-with-online-machine-learning/>
- <http://aria42.com/blog/2014/12/understanding-lbfgs>