

Продвинутый Spark



План на неделю



С нуля решим на Spark задачу классификации текстов



План на неделю



С нуля решим на Spark задачу классификации текстов



Реализуем модель «мешка слов»



План на неделю



С нуля решим на Spark задачу классификации текстов



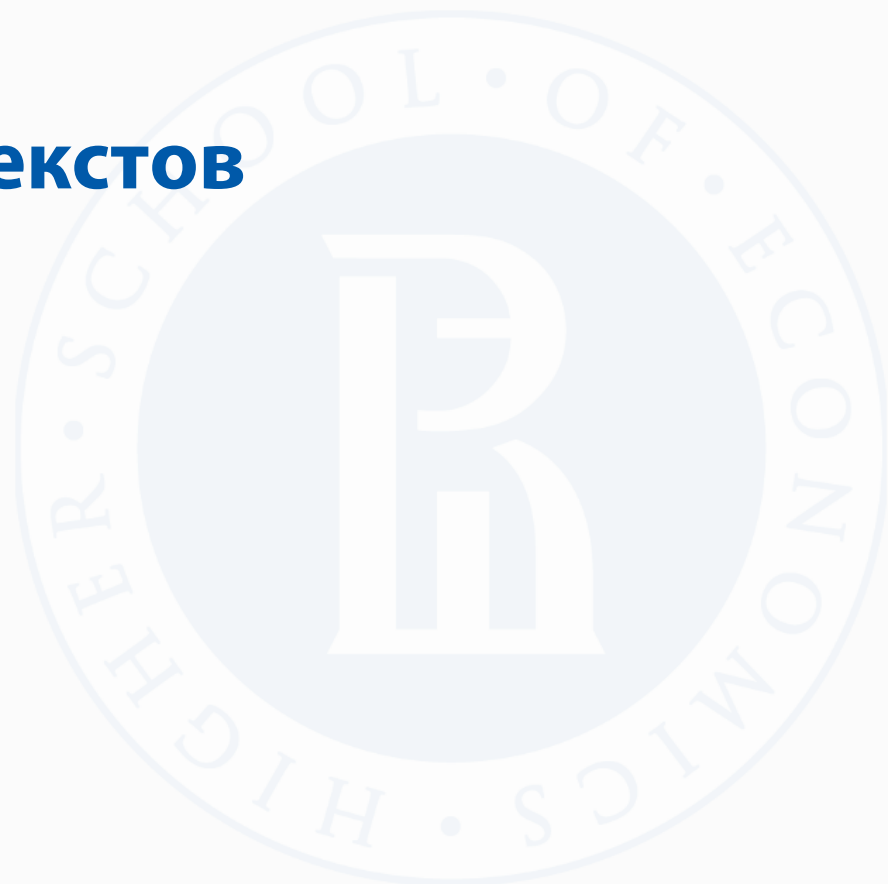
Реализуем модель «мешка слов»



Обучим вручную логистическую регрессию



Векторизация текстов



Векторизация текстов

➔ Модель «мешка слов»:

	good	movie	not	a	did	like
good movie	1	1	0	0	0	0
not a good movie	1	1	1	1	0	0
did not like	0	0	1	0	1	1



Векторизация текстов

➔ Модель «мешка слов»:

	good	movie	not	a	did	like
good movie	1	1	0	0	0	0
not a good movie	1	1	1	1	0	0
did not like	0	0	1	0	1	1

➔ Для векторизации нужен словарь: «слово» ➔ индекс

Векторизация текстов

➔ Модель «мешка слов»:

	good	movie	not	a	did	like
good movie	1	1	0	0	0	0
not a good movie	1	1	1	1	0	0
did not like	0	0	1	0	1	1

➔ Для векторизации нужен словарь: «слово» ➔ индекс

➔ Словарь нужен на каждой машине

Векторизация текстов

➔ Модель «мешка слов»:

	good	movie	not	a	did	like
good movie	1	1	0	0	0	0
not a good movie	1	1	1	1	0	0
did not like	0	0	1	0	1	1

➔ Для векторизации нужен словарь: «слово» ➔ индекс

➔ Словарь нужен на каждой машине

➔ Может не поместиться в RAM

Хэширование вместо словаря

- ➔ Считаем «слово» $\rightarrow \text{hash}(\text{«слово»})$
 - Возьмем большое число корзинок хэш-функции (2^{32})
 - Не занимает памяти и легко распараллеливается



Хэширование вместо словаря

- ➔ Считаем «слово» $\rightarrow \text{hash}(\text{«слово»})$
 - Возьмем большое число корзинок хэш-функции (2^{32})
 - Не занимает памяти и легко распараллеливается

- ➔ Пример хэш-функции (полиномиальная):

$$\text{hash}(s) = s[0] + s[1]p^1 + \dots + s[n]p^n$$

s — строка

p — фиксированное простое число

$s[i]$ — код символа

Хэширование вместо словаря

$\text{hash}(\text{good}) = 0$

$\text{hash}(\text{movie}) = 1$

$\text{hash}(\text{not}) = 2$

$\text{hash}(a) = 3$ ← КОЛЛИЗИЯ

$\text{hash}(did) = 3$ ←

$\text{hash}(\text{like}) = 4$

good movie
not a good movie
did not like



0	1	2	3	4
1	1	0	0	0
1	1	1	1	0
0	0	1	1	1

Хэширование вместо словаря

$\text{hash}(\text{good}) = 0$

$\text{hash}(\text{movie}) = 1$

$\text{hash}(\text{not}) = 2$

$\text{hash}(a) = 3$ ← КОЛЛИЗИЯ

$\text{hash}(did) = 3$ ←

$\text{hash}(\text{like}) = 4$

good movie
not a good movie
did not like



0	1	2	3	4
1	1	0	0	0
1	1	1	1	0
0	0	1	1	1



Чем больше корзинок хэш-функции, тем меньше шанс коллизии!

Хэширование вместо словаря

- Легко сделать из 32-битного хэша (просто считаем полином в `int32`) столько корзинок, сколько захотим:
- «слово» → $\text{hash}(\text{«слово»}) \% 2^{22}$
 - «слово» → $\text{hash}(\text{«слово»}) \% 2^{24}$



Хэширование вместо словаря

➔ Легко сделать из 32-битного хэша (просто считаем полином в `int32`) столько корзинок, сколько захотим:

- «слово» $\rightarrow \text{hash}(\text{«слово»}) \% 2^{22}$
- «слово» $\rightarrow \text{hash}(\text{«слово»}) \% 2^{24}$

➔ Ошибка падает как \log (бит в хэше):



Хэширование в детекции спама

- ➔ Датасет для обучения:
- 0.4 млн пользователей
 - 3.2 млн писем
 - 40 млн слов



Хэширование в детекции спама

➔ Датасет для обучения:

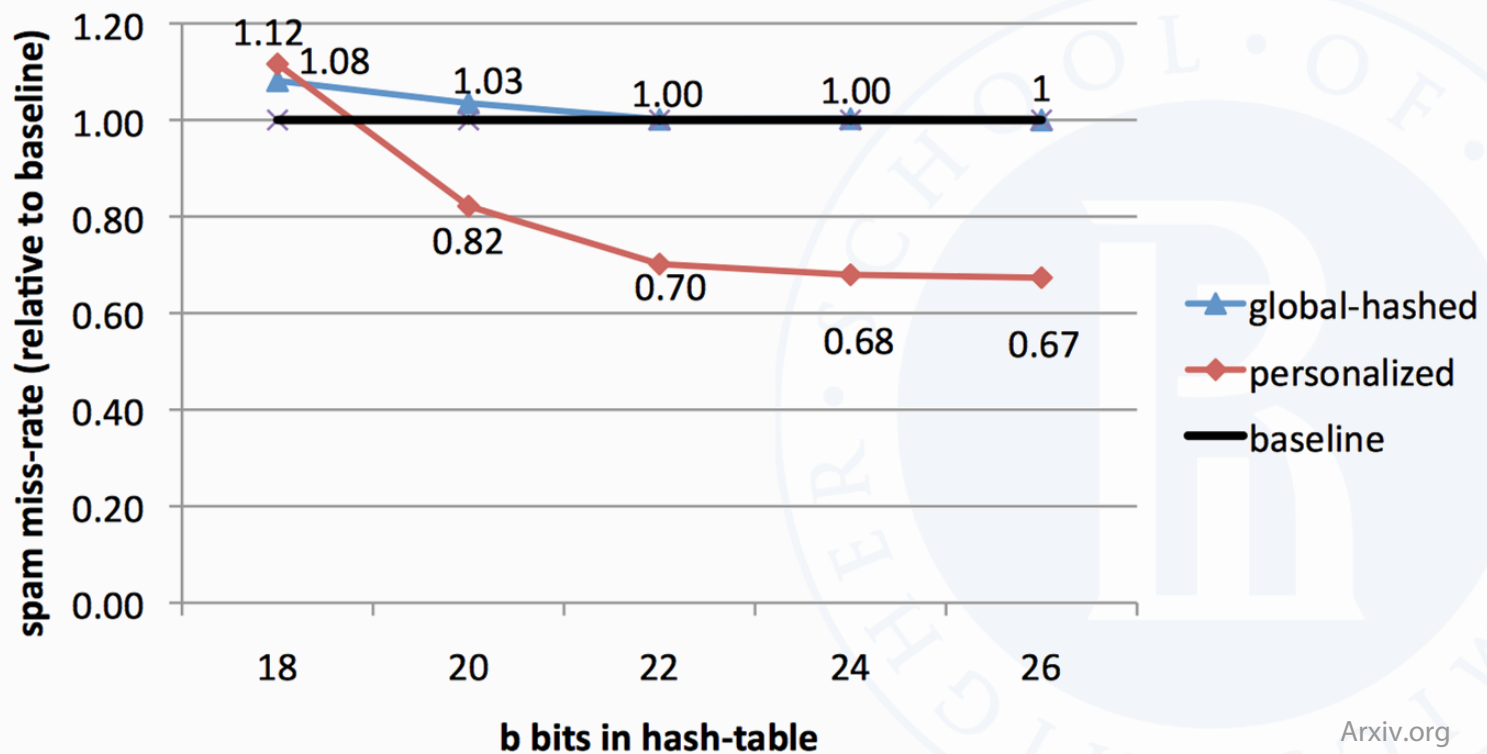
- 0.4 млн пользователей
- 3.2 млн писем
- 40 млн слов

➔ Добавим **персональные** слова в модель:

- для каждого слова добавим еще одно вида:
«пользователь1_слово»
- получим **16 трлн** новых признаков!

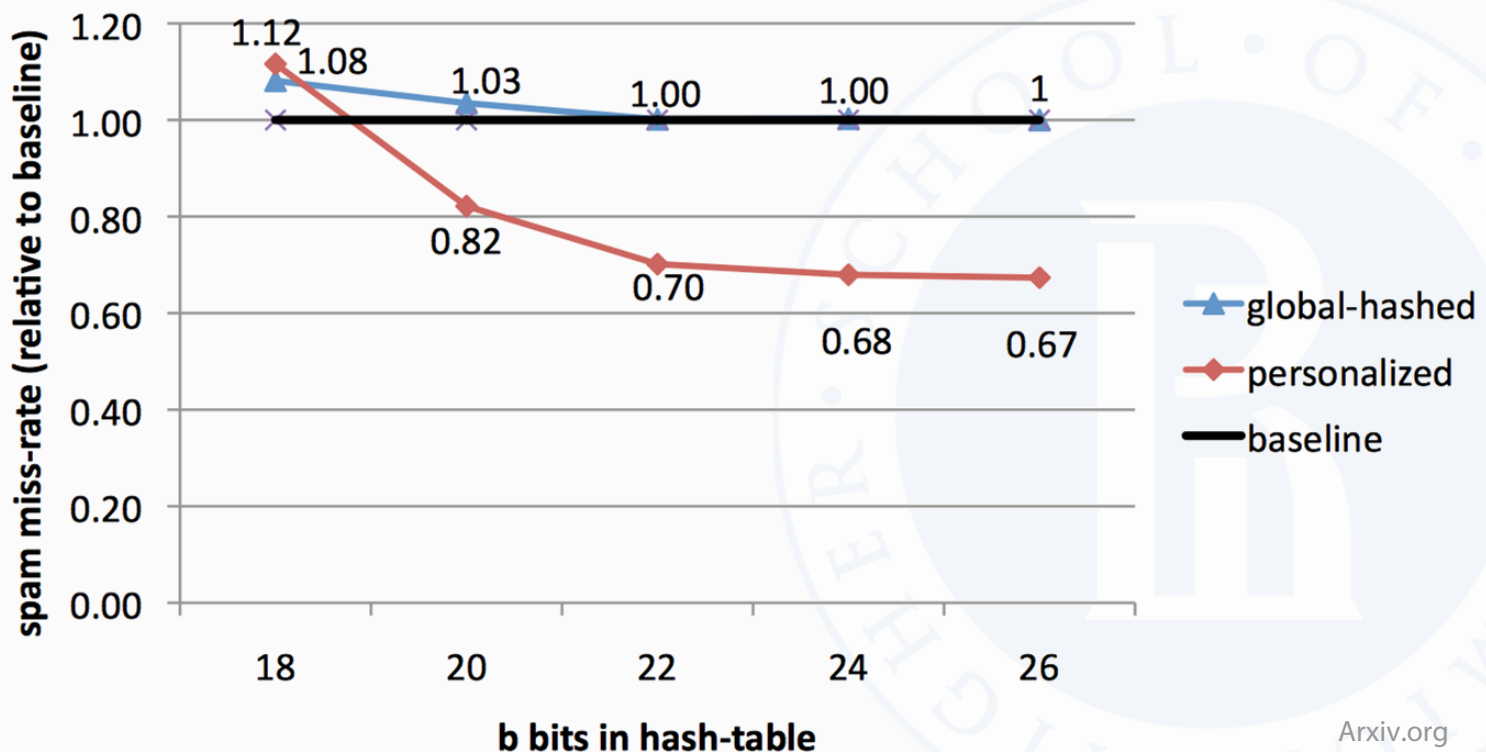
Хэширование в детекции спама

➔ Хэшированные **16 трлн признаков** дают существенный прирост качества



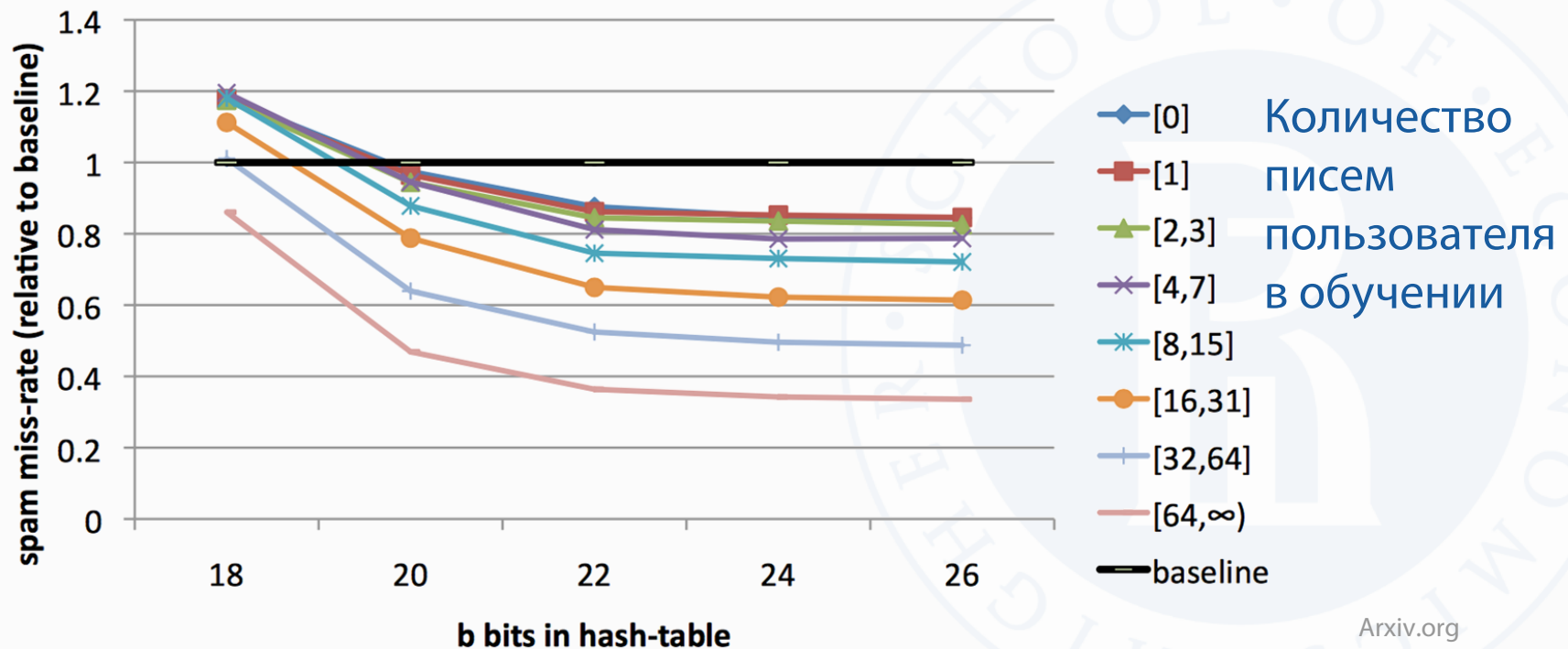
Хэширование в детекции спама

- Хэшированные **16 трлн признаков** дают существенный прирост качества
- Хэширование перестает влиять на качество **не персональной** модели уже на 22 битном хэше



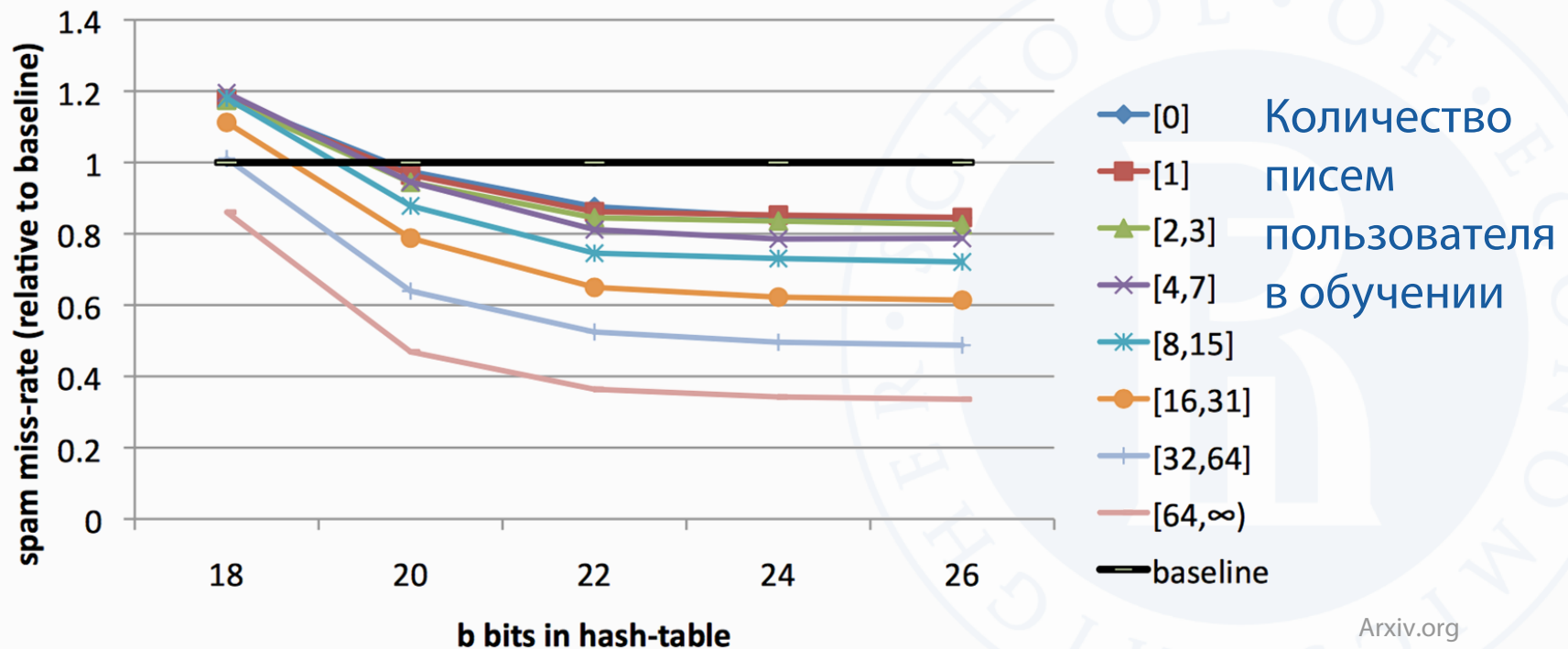
Хэширование в детекции спама

➔ Хорошо работает даже на пользователях, которых **не было** в обучении!



Хэширование в детекции спама

- ➔ Хорошо работает даже на пользователях, которых **не было** в обучении!
- ➔ Все **персональные** зависимости были учтены новыми признаками, а **глобальные** стали универсальнее



Резюме



Для векторизации текста нужно держать в памяти словарь: «слово» → индекс



Резюме



Для векторизации текста нужно держать в памяти словарь: «слово» → индекс



Можно без словаря: «слово» → $\text{hash}(\text{«слово»}) \% 2^{24}$



Резюме



Для векторизации текста нужно держать в памяти словарь: «слово» → индекс



Можно без словаря: «слово» → $\text{hash}(\text{«слово»}) \% 2^{24}$



Хэширование позволяет работать с огромным количеством признаков, оно реализовано в промышленном пакете `vowpal wabbit`



Резюме



Для векторизации текста нужно держать в памяти словарь: «слово» → индекс



Можно без словаря: «слово» → $\text{hash}(\text{«слово»}) \% 2^{24}$

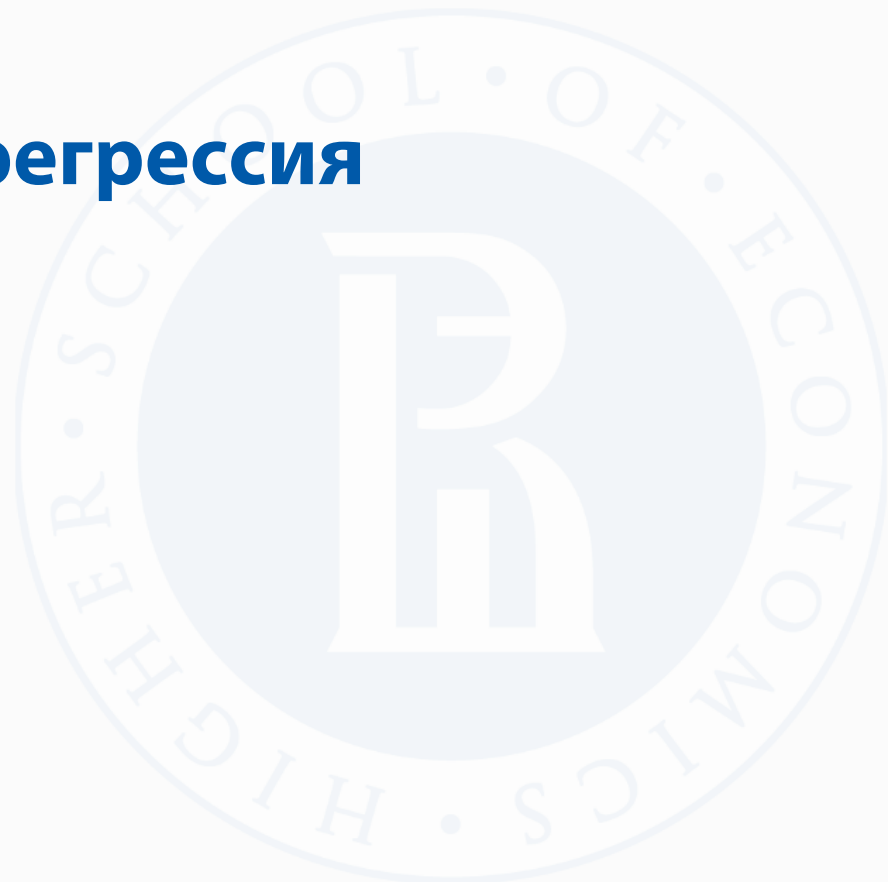


Хэширование позволяет работать с огромным количеством признаков, оно реализовано в промышленном пакете `vowpal wabbit`



Далее: поговорим о логистической регрессии

Логистическая регрессия



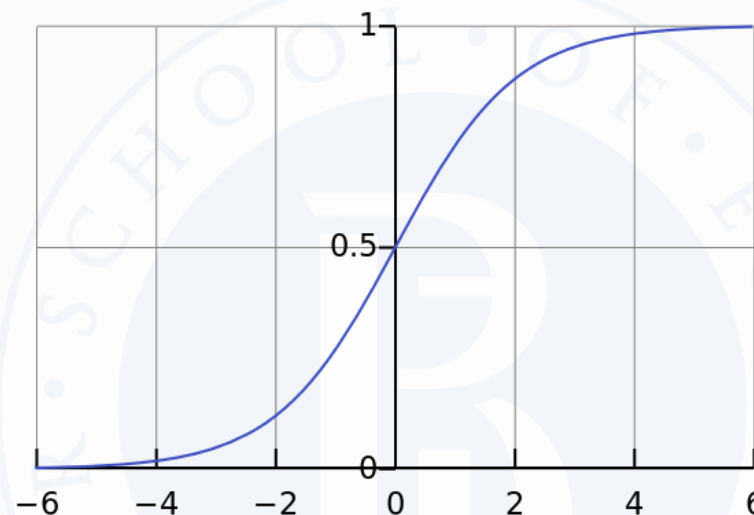
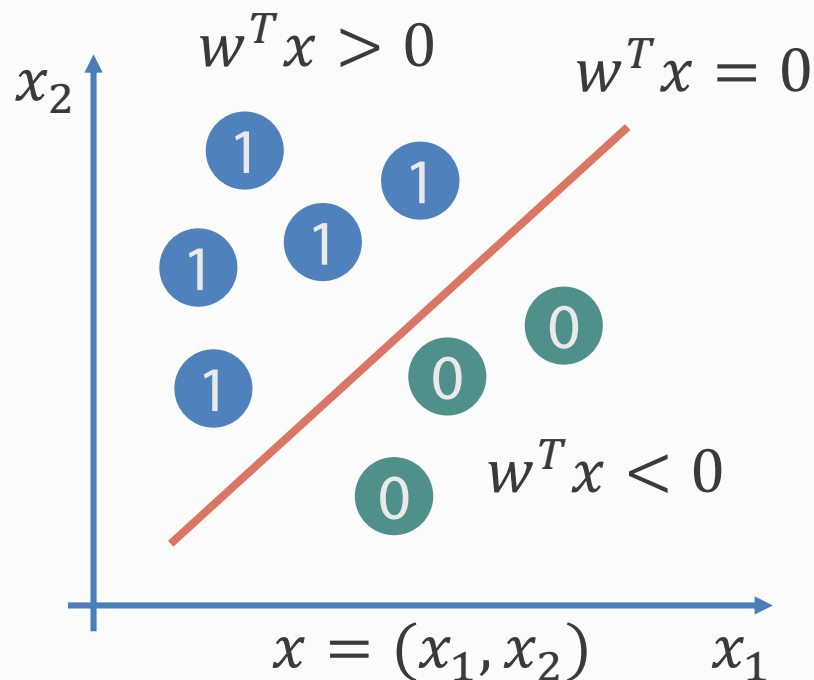
Логистическая регрессия

x^i — признаки i -го объекта

$y^i \in \{0, 1\}$ — класс i -го объекта

w — коэффициенты модели

$p^i = \sigma(w^T x^i)$ — вероятность класса 1 для i -го объекта



$$\sigma(d) = \frac{1}{1 + e^{-d}}$$

Wikipedia.org

Логистическая регрессия

x^i — признаки i -го объекта

$y^i \in \{0, 1\}$ — класс i -го объекта

w — коэффициенты модели

$p^i = \sigma(w^T x^i)$ — вероятность класса 1 для i -го объекта

➔ Минимизируем log-loss:

$$-\sum_i y^i \log(p^i) + (1 - y^i) \log(1 - p^i)$$

Логистическая регрессия

x^i — признаки i -го объекта

$y^i \in \{0, 1\}$ — класс i -го объекта

w — коэффициенты модели

$p^i = \sigma(w^T x^i)$ — вероятность класса 1 для i -го объекта

➔ Минимизируем log-loss:

$$-\sum_i y^i \log(p^i) + (1 - y^i) \log(1 - p^i)$$

➔ Шагая в итерациях в направлении антиградиента:

$$w_{new} = w - \alpha \sum_i x^i (p^i - y^i)$$

Логистическая регрессия

x^i — признаки i -го объекта

$y^i \in \{0, 1\}$ — класс i -го объекта

w — коэффициенты модели

$p^i = \sigma(w^T x^i)$ — вероятность класса 1 для i -го объекта

➔ Независимо считаем для каждого блока набора данных вклад в градиент:

$$\sum_i x^i (p^i - y^i)$$

Логистическая регрессия

x^i — признаки i -го объекта

$y^i \in \{0, 1\}$ — класс i -го объекта

w — коэффициенты модели

$p^i = \sigma(w^T x^i)$ — вероятность класса 1 для i -го объекта

➔ Независимо считаем для каждого блока набора данных вклад в градиент:

$$\sum_i x^i (p^i - y^i)$$

➔ Собираем на драйвере финальную сумму и делаем шаг:

$$w_{new} = w - \alpha \sum_i x^i (p^i - y^i)$$



Резюме



Для наглядности будем обучать градиентным спуском, который легко распараллелить






Резюме

-  Для наглядности будем обучать градиентным спуском, который легко распараллелить
-  Промышленные пакеты, такие как vowpal wabbit, делают L-BFGS шаг для более быстрой сходимости



Резюме

-  Для наглядности будем обучать градиентным спуском, который легко распараллелить
-  Промышленные пакеты, такие как vowpal wabbit, делают L-BFGS шаг для более быстрой сходимости
-  Мы готовы к решению задачи в Spark!

