Ответы на вопросы

Чуйкова Екатерина

377а

Московский физико-технический институт

18.01.2019

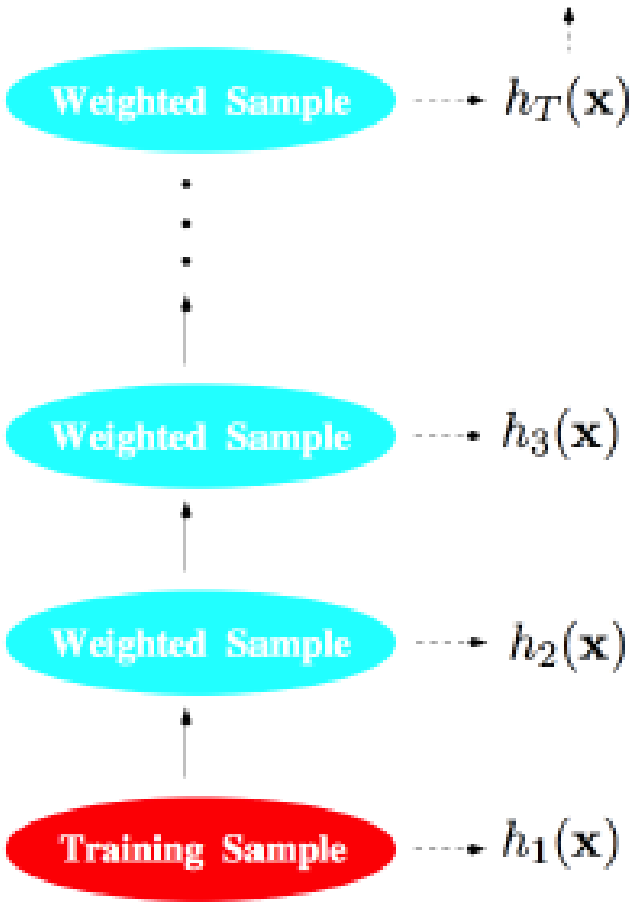
Билет 10
Методы построения композиций классификаторов. Бустинг и бэггинг.

Композиция классификаторов

При решении сложных задач часто оказывается, что ни один из алгоритмов не обеспечивает желаемого качества восстановления зависимости. В таких случаях имеет смысл строить композиции алгоритмов, в которых ошибки отдельных алгоритмов взаимно компенсируются.

Бустинг

$$f_T(\mathbf{x}) = \text{sign}\left[\sum_{t=1}^{T} \alpha_t h_t(\mathbf{x})\right]$$



Процедура последовательного построения композиции алгоритмов машинного обучения, когда каждый следующий алгоритм стремится компенсировать недостатки композиции всех предыдущих алгоритмов

Бустинг

- Жадный алгоритм.
- Можно использовать как метод фильтрации выбросов, т.к. объекты с наибольшими весами, скорее всего, являются шумовыми выбросам
- Хорошая обобщающая способность

Бэггинг

1. Из исходной обучающей выборки длины ℓ формируются различные обучающие подвыборки той же длины ℓ с помощью бутстрепа — случайного выбора с возвращениями. При этом некоторые объекты попадают в подвыборку по несколько раз, некоторые — ни разу.
2. Производится независимое обучения каждого элементарного классификатора
3. Производится классификация основной выборки на каждом из подпространств (независимо).
4. Принимается окончательное решение о принадлежности объекта одному из классов
   - Простое большинство
   - Взвешивание классификаторов

Бэггинг

- Ошибки базовых алгоритмов взаимно компенсируются при голосовании.
- Выбросы могут не попадать в некоторые обучающие подвыборки.
- Особенно эффективен на малых выборках, когда исключение даже небольшой доли обучающих объектов приводит к построению существенно различных базовых алгоритмов.

Билет 21
Выпуклая оболочка конечного множества точек на плоскости. Алгоритм Джарвиса. Алгоритм Грэхема.

Выпуклая оболочка

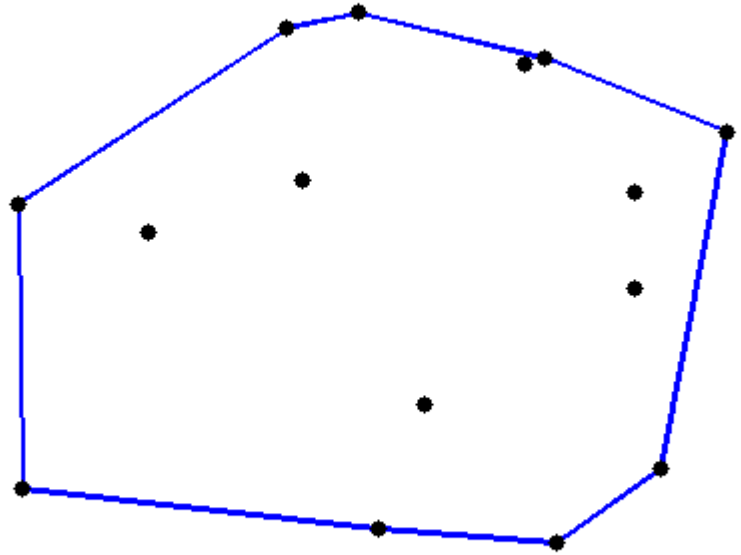
Пусть на плоскости заданы k различных точек $p_1, p_2, \ldots, p_k$.
Множество точек

$$p = \alpha_1 p_1 + \alpha_2 p_2 + \ldots + \alpha_k p_k,$$
$$(\alpha_i \in R,\ \alpha_i \geq 0,\ \alpha_1 + \alpha_2 + \ldots + \alpha_k = 1)$$

называется ***выпуклым множеством***, порожденным точками $p_1, p_2, \ldots, p_k$,
а точка p - называется ***выпуклой комбинацией*** точек $p_1, p_2, \ldots, p_k$.

Выпуклой оболочкой множества точек ***L*** на плоскости называется наименьшее выпуклое множество, содержащее ***L***.

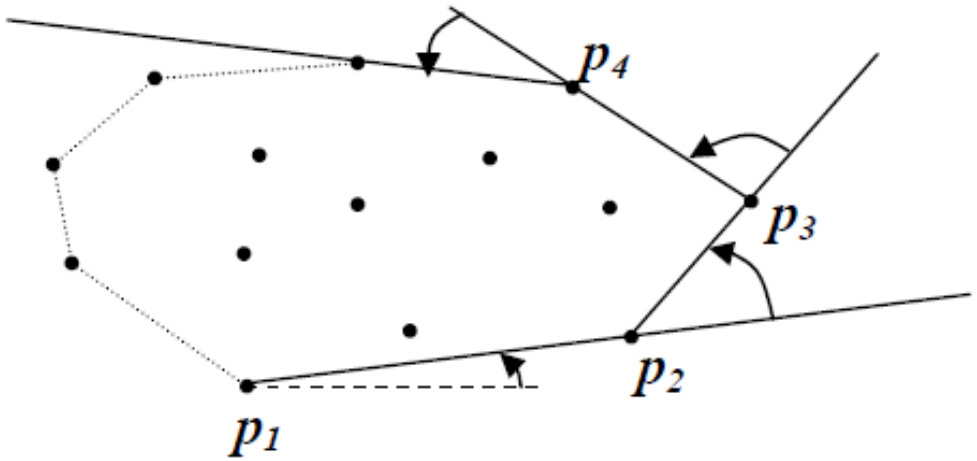
Задача построения выпуклой оболочки



Для множества ***L*** из ***n*** точек требуется построить выпуклую оболочку *conv(**L**)* в виде полного описания границы.

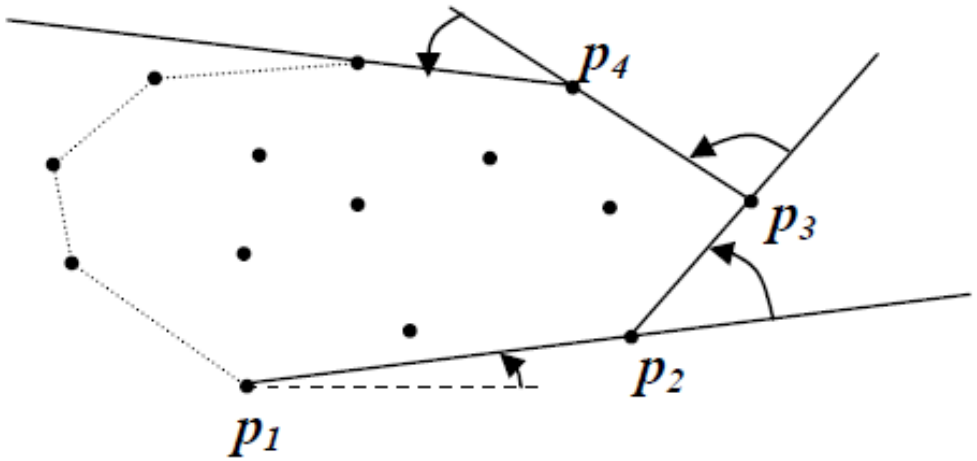
Описание границы :
упорядоченной подмножество точек из ***L***, называемых граничными точками.
Точки из ***L***, не являющиеся граничными, называются внутренними.

Метод Джарвиса (заворачивание подарка)



- Пусть найдена наименьшая в лексикографическом порядке точка p_1 заданного множества L. Эта точка заведомо граничная, т.е. является вершиной выпуклой оболочки.
- Пусть точка p_2 - это точка, имеющая наименьший положительный полярный угол относительно точки p_1 как начала координат.
- Следующая граничная точка p_3 выбирается таким образом, чтобы вектор p_2p_3 имел наименьший положительный угол относительно вектора p_1p_2.
- Аналогично ищутся остальные граничные точки.

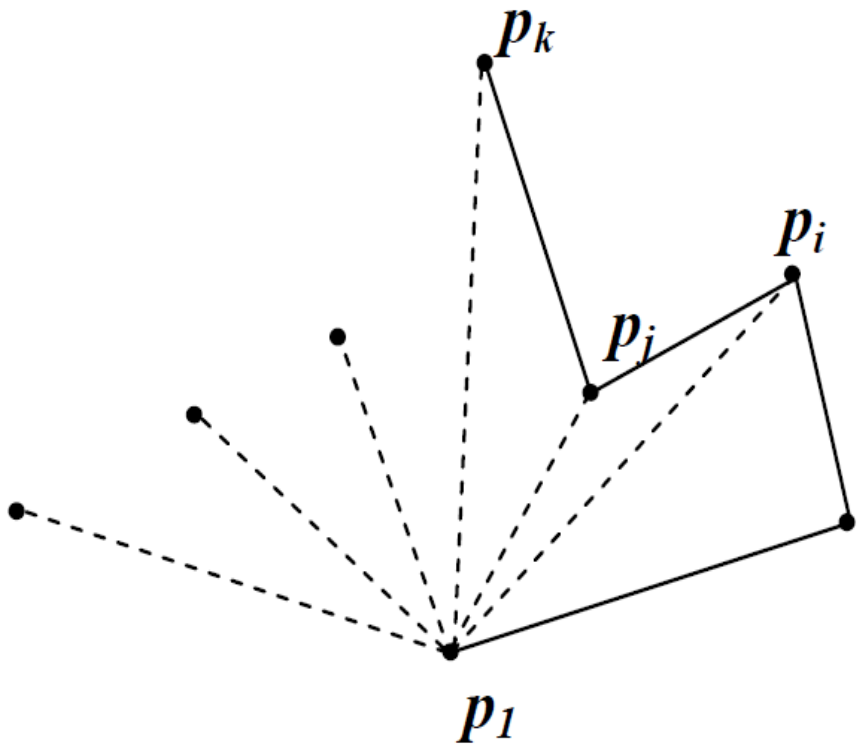
Метод Джарвиса (заворачивание подарка)



Алгоритм Джарвиса затрачивает на нахождение одной граничной точки линейное время.

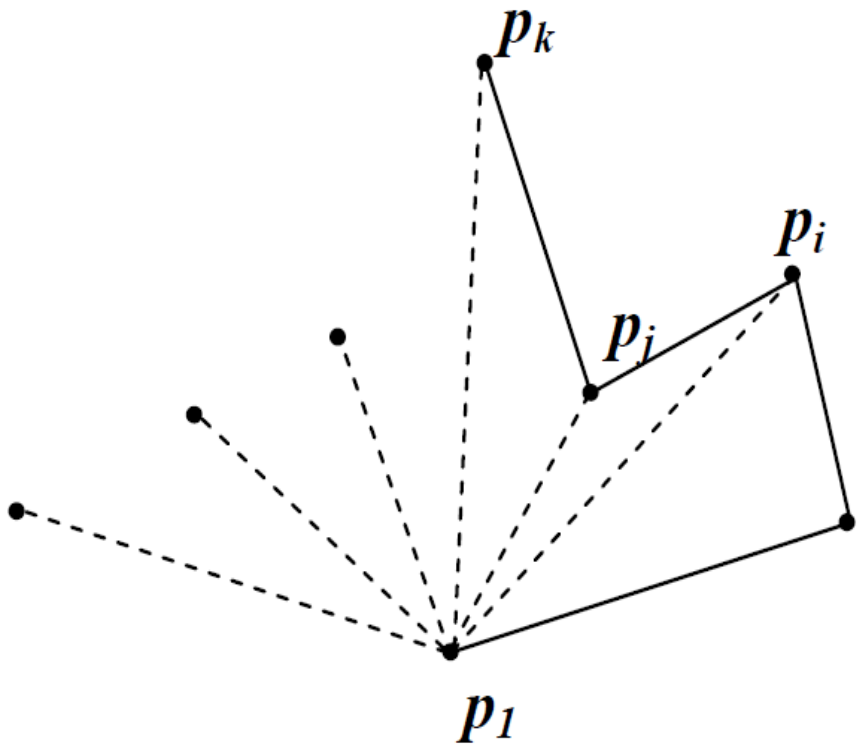
Общее время выполнения алгоритма в худшем случае ***O(nh),*** где ***h*** - число вершин выпуклой оболочки.

Метод Грэхема



- Пусть найдена наименьшая в лексикографическом порядке точка ***p1 (O(n))***.
- Упорядочим лексикографически все остальные точки в соответствии с полярными углами векторов p_1p_i и длинами этих векторов
  - 1. Тройка $p_ip_jp_k$ образует правый поворот. Удалить среднюю точку p_j тройки из списка вершин и проверить вновь образовавшуюся текущую тройку $p_{i-1}p_ip_k$.
  - 2. Тройка $p_ip_jp_k$ образует левый поворот. Продолжить просмотр, перейдя к проверке тройки $p_jp_kp_{k+1}$.

Метод Грэхема



- Поиск ***p1 - O(n)***.
- Время обхода составляет ***O(n)***.
- Сортировка массива точек занимает время ***O(n log n)***

Общее время работы алгоритма ***O(n log n)***.

Билет 35
Векторные представления предложений и текстов. Основные модели (TF-IDF, усреднение, нейронные модели). Механизм Self-attention.

TF-IDF

Вес некоторого слова пропорционален частоте употребления этого слова в документе и обратно пропорционален частоте употребления слова во всех документах коллекции.

$$\text{tf-idf}(t, d, D) = \text{tf}(t, d) \times \text{idf}(t, D)$$

TF-IDF

TF (*term frequency* — частота слова) - Оценивается важность слова в пределах одного документа

$$\mathrm{tf}(t,d) = \frac{n_t}{\sum_k n_k}$$

n_t – число вхожнений слова t в документ

$\sum_k n_k$ - общее число слов в данном документе

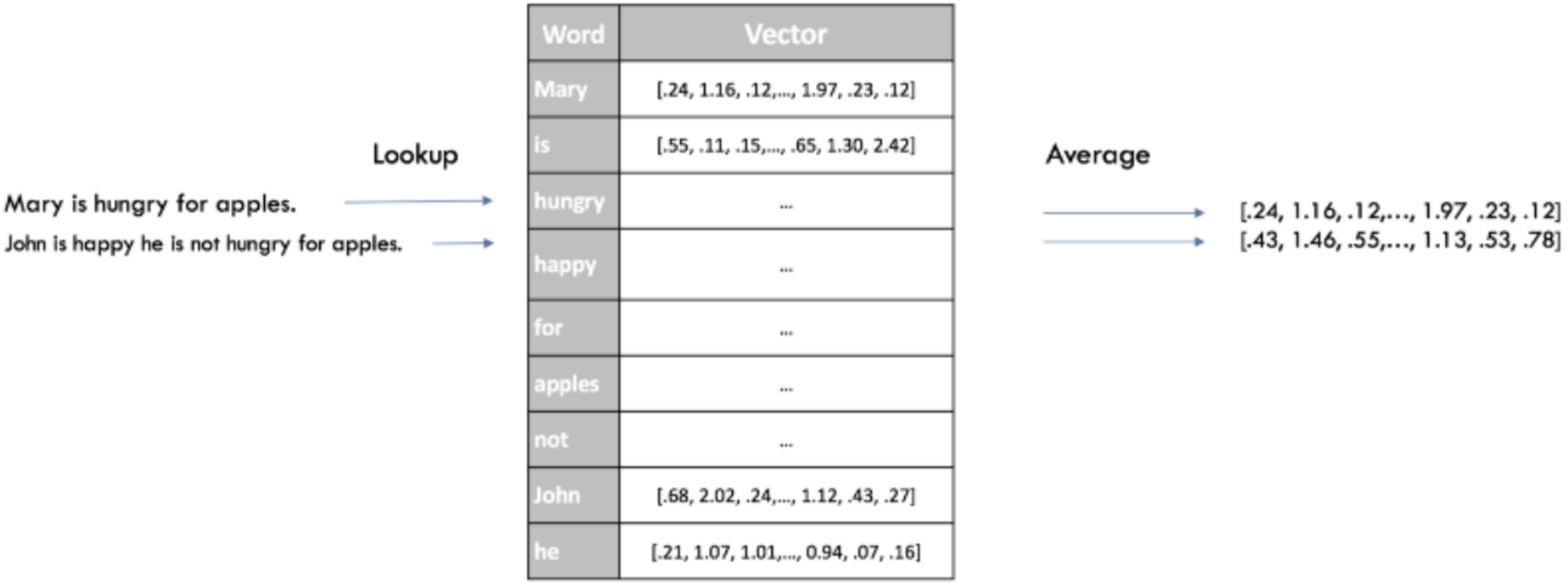
IDF (*inverse document frequency* — обратная частота документа) - уменьшает вес широкоупотребительных слов

$$\mathrm{idf}(t,D) = \log \frac{|D|}{|\{\, d_i \in D \mid t \in d_i \,\}|}$$

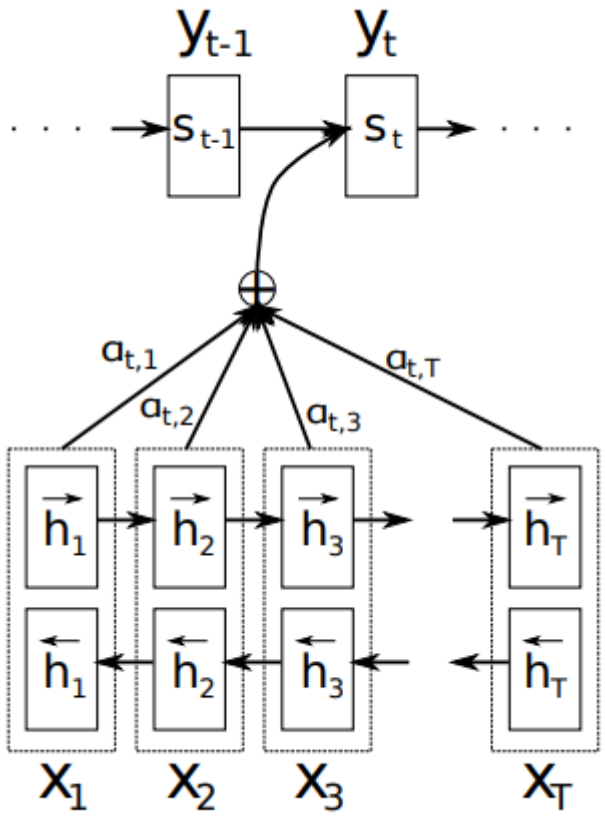
$|D|$ - число документов в коллекции

$|\{\, d_i \in D \mid t \in d_i \,\}|$ - число документов из коллекции D, в которых встречается t

Усреднение Word2Vec



Self-attention

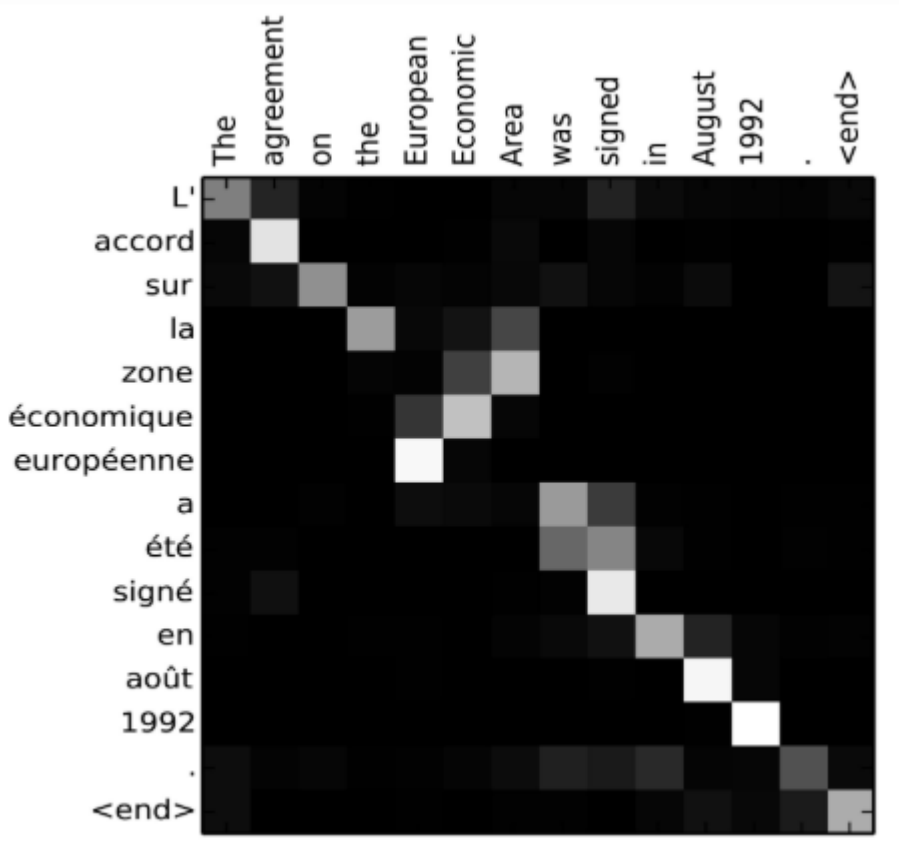


$$e_{ij} = a(s_{i-1}, h_j)$$

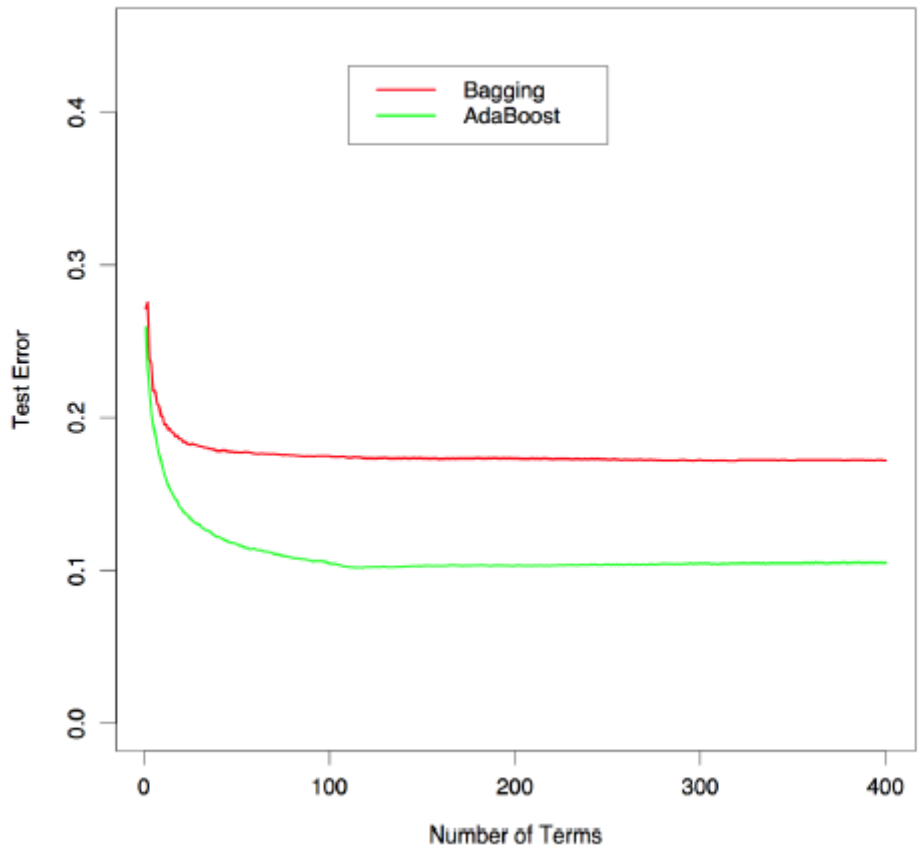
$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{ik})}$$

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j.$$

$$s_i = f(s_{i-1}, y_{i-1}, c_i)$$



Self-attention


Bagging
AdaBoost
Test Error
0.0
0.1
0.2
0.3
0.4
0
100
200
300
400
Number of Terms