Министерство образования и науки Российской Федерации

Государственное образовательное учреждение
высшего профессионального образования
Московский физико-технический институт
(государственный университет)

Факультет инноваций и высоких технологий
Кафедра компьютерной лингвистики

выпускная квалификационная работа
(бакалаврская работа)
по направлению 01.03.02 «Прикладная математика и информатика»

Японский. Буквенные n-грамы для распознавания

Студент Куликов А.В.

Научный руководитель Андрианов А.И.

Москва, 2017

Оглавление

Введение 4
1 Постановка задачи 6
1.1 Обзор японского языка 7
1.2 Путающиеся символы в японском 9
1.3 Формальная постановка задачи 10
2 Обзор источников 12
2.1 Методы, не использующие словное деление 12
2.2 Методы, использующие словное деление 12
3 Описание моделей оценивания текста 14
3.1 N-граммные модели с фиксированным n 14
3.2 Backoff-модель 15
3.3 Модель Катца (Katz) 15
4 Описание эксперимента 17
4.1 Корпус 17
4.2 Обработка корпуса 18
4.3 Генератор шума и режимы его работы 21
4.4 Статистика по шумовым символам 23
4.5 Ход эксперимента 26
4.6 Baseline эксперимента 27
5 Реализация модели 28
5.1 Общее окружение: nltk, pygtrie 28
5.2 Полезные утилиты 28
6 Результаты эксперимента 30
6.1 Результаты для различных моделей 30

6.2 Затраты ресурсов 31
7 Анализ результатов 32
8 Заключение 33
Список литературы 34
Приложение А. 36

Введение

История попыток распознать текст началась более века назад. В 1914 году Эмануэль Гольдберг разработал устройство, которой считывало символы и транслировало их в телеграфный код. Примерно в то же время ирландский химик Эдмунд Фурнье д'Альбе создал и запатентовал «оптофон» — прибор, умеющий переводить написанное в систему звуков, различающихся по высоте. Оптофон предназначался для того, чтобы слепые могли «читать».

В 1929 году Густав Таушек (Gustav Tauschek) разработал метод оптического распознавания текста. Машина Таушека представляла собой механическое устройство, которое использовало шрифтовые шаблоны и фотодетектор. Он запатентовал своё изобретение сначала в Германии, а позднее и в США, в 1935 году. Это и положило начало проблеме качественного оптического распознавания символов (Optical Character Recognition, OCR).

Коммерческое производство подобных маних было налажено уже в 1950-х, после войны. Использовавшие наработки военных, производители OCR-машин продвигались всё дальше, увеличивая применимость технологии и качество распознавания.

Постепенно появлялись как универсальные OCR-программы (ABBYY FineReader, Adobe Acrobat), так и специализированные для конкретной области (SmartScore для нотной записи, Persian Reader для фарси и т.д.). При этом точность в задаче распознавания напечатанных латинских символов достигла 99%-100% качества, в то время как корректное распознавание рукописного текста или текста, написанного в другом алфавите, до сих пор является темой множества исследований. Особняком стоит задача распознавания текста на восточных языках (китайский, японский, корейский, ...), из-за большого размера алфавита в этих языках.

Настоящая работа представляет собой сравнение некоторых методов машинного обучения для исправления ошибок распознавания текста в японском языке.

Спектр способов, которыми можно решать проблему автоматического

исправления ошибок, довольно широк, и включает в себя различные вариации n-граммных методов (n-gram models), использование нейросетей (Neural Networks, NN), скрытых моделей Маркова (Hidden Markov Models, HMM) и прочих методов машинного обучения. Более подробный обзор основных современных подходов можно найти в [1].

Среди возможных решений использование n-граммных моделей занимает особую нишу из-за относительной прозрачности и интуитивности принципов работы, и в то же время достаточно широких возможностей по настройке алгоритма.

Подход, предложенный в [2], использует n-граммные модели, а также различные алгоритмы сглаживания для исправления опечаток, опираясь на словное деление текста.

В работе [3] также даются эвристики для определения границ слов, использующие граф линейного деления (ГЛД). Эти границы слов затем используются в n-граммной модели в качестве вспомогательного контекста.

Более подробно эти и другие подходы разобраны в соответствующем разделе (разд. 2).

Данное исследование призвано рассмотреть некоторые из n-граммных моделей и сравнить их эффективность в задаче исправления опечаток в японском языке.

Актуальным приложением этой работы является система распознавания восточных языков в ABBYY FineReader.

1 Постановка задачи

Определение 1.0.1. *Оптическое распознавание символов (Optical Character Recognition, OCR)* – процесс считывания текста с физического носителя и его сохранения в цифровом формате. Текст состоит из *символов*.

Определение 1.0.2. *Ошибка OCR* – случай, когда очередной символ текста распознался неверно или не распознался. Ведёт к понижению качества распознавания.

Определение 1.0.3. *N-грамма* – последовательность из n элементов (слов, звуков, символов). Анализируя их частотности, можно строить модели для анализа и синтеза языка.

Определение 1.0.4. *N-граммная модель* – вероятностная модель языка, которая рассчитывает вероятность последнего элемента n-граммы, если известны все предыдущие.
При использовании n-граммных моделей предполагается, что появление каждого элемента зависит только от предыдущих элементов.

Цель работы – сравнить эффективность различных символьных n-граммных моделей в задаче исправления ошибок OCR в японском языке.

Из цели работы вытекают следующие **задачи**:

- Рассмотреть существующие подходы к n-граммному моделированию японского языка;
- Реализовать некоторые модели;
- Развернуть систему для тестирования и сравнения моделей.

Чтобы понять специфику цели работы, нужно учесть особенности японского языка.

Очевидно, что устройство японского языка на уровне конкретных символов сложнее, чем устройство языков латино-романской группы, в которых существует всего 25-40 символов, учитывая возможную диакритику.

1.1 Обзор японского языка

Письменный японский текст – это комбинация слогово-фонетических символов (кана) и иероглифов (кандзи). Слоговая азбука кана делится на катакану и хирагану, которые представляют собой разные графические формы одних и тех же слогов.

Рассмотрим эти символы подробнее:

- Хирагана (см. Рис. рис. 1). В основном используется для образования грамматических морфем.

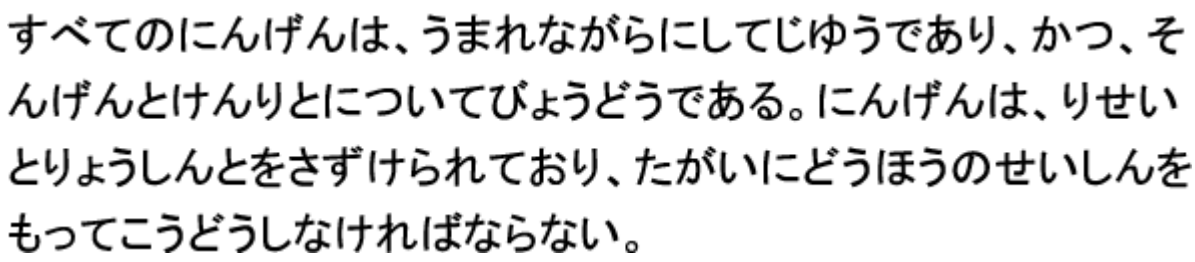


Рис. 1: hirag_sample

- Катакана (см. Рис. рис. 2). Используется для транскрибирования иностранных заимствованных слов.

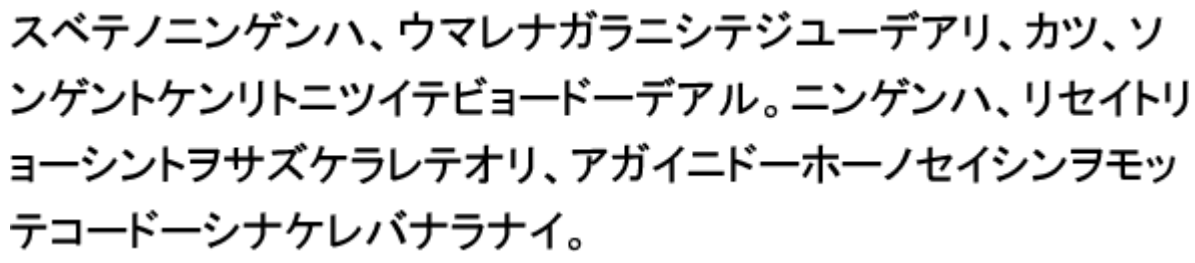


Рис. 2: katak_sample

- Также есть диакритические символы – дакутен, хандакутен (см. Рис. рис. 3 и рис. 4). Они могут применяться как к катакане, так и к хирагане, и определённым образом влияют на звучание слогов.

Рис. 4: dakut_sample

- Кандзи (см. Рис. рис. 5). Это символы, несущие семантическую нагрузку.

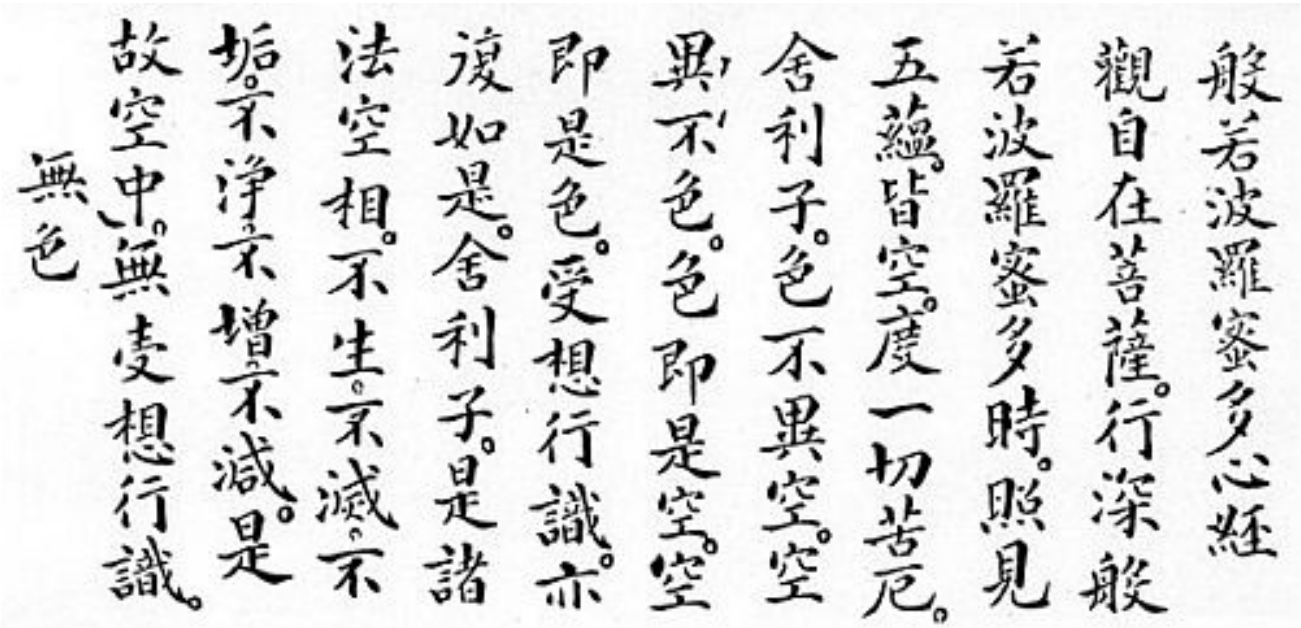


Рис. 5: kandji_sample

Кана различает 46 слогов, которые могут записываться как катаканой, так и хираганой. А вот иероглифов кандзи существует гораздо больше (2136 (т.н. jōyō kanji – "обычно используемые кандзи") достаточно для жизни, 6879 используется в кодировке JIS X 0208 (Japanese Industrial Standart, см. [10]), а стандарт Unicode определяет то ли 21000, то ли 75000, надо разобраться) TODO: цифры, мб зависит от стандарта Unicode?.

Кроме перечисленных символов, в японском тексте могут быть и другие: фуригана – маленькие знаки каны в качество фонетических подсказок, ромадзи – система транслитерации японских слов в латиницу и т.д. Однако, в данной работе эти разделы оставлены за кадром.

Японский текст записывается с помощью комбинаций кандзи, кан и пунктуации, при этом отсутствует пробельное деление предложений на слова (см. Рис. рис. 6).

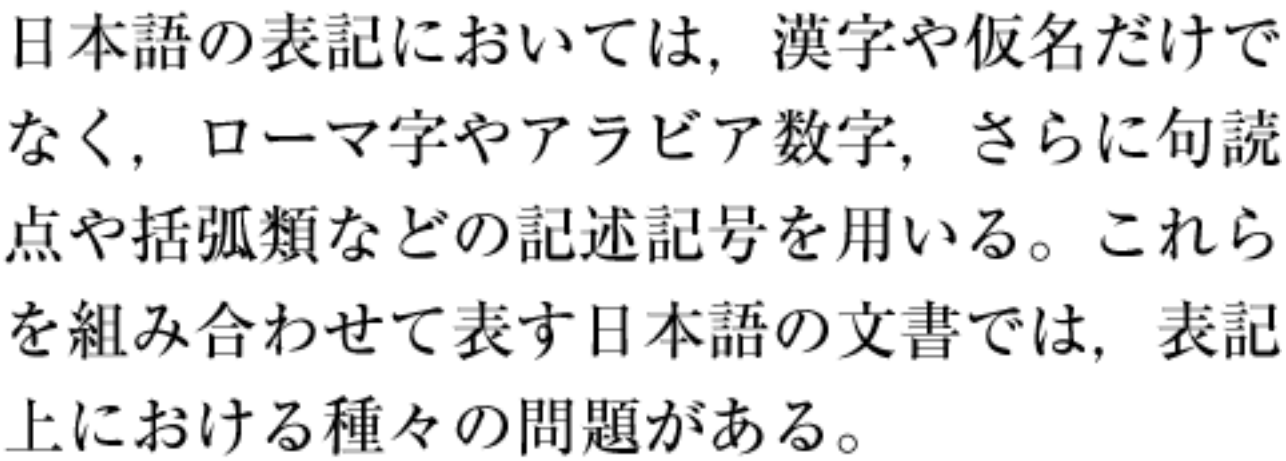


Рис. 6: japtext

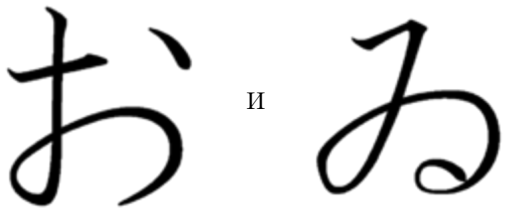
По сравнению с латино-романскими языками, где алфавит меньше в сотни раз, а деление текста на слова очевидно, задача корректного распознавания символов становится значительно сложнее. Это требует более изощрённых подходов для автоматического анализа распознанного текста и поиска ошибок в нём.

Рассмотрим несколько примеров символов, которые легко спутать.

1.2 Путающиеся символы в японском

При таком большом размере алфавита частые ошибки OCR можно разбить по классам. Вот некоторые из них:

2Kana 2 похожие каны. Таких случаев достаточно мало, а методы их различения уже существуют (см., например, метод с использованием глубокого обучения и свёрточных нейронных сетей в работе [9]).

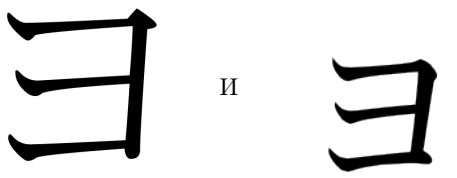


и

KaGa Кана может легко путаться с соответствующим её дакутен-символом.

и

BigSmall Существуют большие и маленькие каны, которые нужно различать.



и

В будущем мы будем рассматривать 3 случая ошибок: **KaGa**, **BigSmall** и **Mix** (смесь **KaGa** и **BigSmall**), более строгое определение которых будет дано в разд. 4.

1.3 Формальная постановка задачи

Определение 1.3.1. *Алфавит* $\Sigma = \{a, b, c, ..\}$ – множество символов в данном языке. В японском языке их около 80000, стандарт Unicode поддерживает примерно 21000.

Определение 1.3.2. *Текст* $Text \in \Sigma^+$ – последовательность символов из алфавита Σ положительной длины.

Определение 1.3.3. Текст делится на конечное множество *предложений* $S = \{S_1, S_2, S_3, ...\}$ знаками пунктуации и форматированием. $Text = S_1 S_2 S_3 ....$

Для каждого из предложения текста существует единственно верный вариант написания, а также некоторое (фиксированное) число неверных. Требуется ответить, какой из вариантов верен.

Определение 1.3.4. *Оценивающий алгоритм (estimator)* $\Theta : S \to \mathbb{R}^+$ – функция, возвращающая оценку правильности варианта S.

Среди k вариантов предложения выбирается наилучший: $S_{best} = \underset{S}{\operatorname{argmax}}\, \Theta(S)$, который и считается правильным.

Если S_{best} угадано верно, то на данном предложении алгоритм Θ отработал правильно.

Определение 1.3.5. *Качество алгоритма* $Q(\Theta) = \dfrac{\#\{\textit{угаданных предложений}\}}{\#\{\textit{всего предложений}\}}$.

Задача – реализовать ряд оценивающих алгоритмов (см. раздел разд. 3), основанных на n-граммных моделях, и сравнить их по качеству.

2 Обзор источников

Проблема качественного OCR в различных языках (в частности, в японском) стоит достаточно давно, и существует множество различных подходов к её решению. Большинство подходов основываются на применении различных методов машинного обучения, такие как n-граммные модели, нейросети, модели Маркова и т.д.

В силу особенностей японского языка (см. разд. 1.1), а именно отсутствия словного деления в текстах, существующие методы исправления ошибок OCR можно разделить на 2 класса: использующие информацию о словном делении и не использующие её.

2.1 Методы, не использующие словное деление

Идея подобных методов заключается в том, чтобы, не тратя ресурсы на определение границ слов в тексте, оперировать предложениями (границы которых выделяются достаточно легко) как единицами трансляции, и исправлять возможные ошибки OCR, не углубляясь в членение предложений.

TODO: примеры статей?

Стоит также заметить, что настоящая работа может быть отнесена к этому классу.

2.2 Методы, использующие словное деление

Информация о словном делении текста даёт удобный контекст для выделения признаков и настройки параметров при машинном обучении. Но эту информацию надо откуда-то получать, что приводит к делению алгоритмов на 2 этапа:

- Получение словного деления. Может производиться с помощью специализированных алгоритмов (например, модификаций алгоритма Витерби, как в [3], или марковских случайных полей (conditional random fields, CRFs), как в [**?**]), а также путём анализа конкретных языковых конструкций (например, bunsetsu boundaries, см. [**?**]).

Кроме того, словное деление может быть получено путём использования в работе предварительно размеченного корпуса, в разметке которого есть словное деление. В качестве примеров таких корпусов можно привести EDR Japanese Corpus (см. [**?**]), ATR Dialogue Database (см. [**?**]) и т.д.

- Применение словного деления как контекста для детектирования ошибок OCR. Достоверная информация о словном делении позволяет считать слова единицами трансляции, не теряя контекста предложения. Это даёт возможность получить больше признаков для машинного обучения. Подобный подход представлен в статье [2], где используются n-граммные модели с backoff и различными подходами к сглаживанию (Good-Turing, Witten-Bell).

Обзору основных актуальных методов исправления ошибок OCR в японском также целиком посвящена статья [1].

3 Описание моделей оценивания текста

Перед обучением моделей корпус разбивается на независимые и гомогенные части: обучающую и тестовую выборки. Обучающая выборка используется для обучения модели, тестовая – для проверки качества обучения и, собственно, оценки модели.

В работе рассматриваются следующие модели:

- n-граммные с фиксированным n, $n \in \{1, 2, 3\}$
- Backoff-модель, $n_{max} \in \{3, 5, 7\}$
- Модель Катца (Katz), $n_{max} \in \{3, 5, 7\}$

Также из-за большого размера алфавита необходимо использовать сглаживание (smoothing) для учёта символов и n-грамм, не встретившихся в обучающей выборке. Подробнее о механизме сглаживания – см. раздел разд. 4.

3.1 N-граммные модели с фиксированным n

Обучение модели Для данного n по обучающей выборке собираются статистики по всем n-граммам. Эти статистики затем нормализуются и сериализуются для дальнейшего использования.

$$C(x_{i-n+1}, ..., x_{i-1}, x_i) = \frac{N(x_{i-n+1}, ..., x_{i-1}, x_i)}{N(x_{i-n+1}, ..., x_{i-1})}$$

Применение модели В силу простоты модели оценка n-граммы из тестовой выборки берётся напрямую из собранных на предыдущем этапе статистик.

$$P(x_i|x_{i-n+1}, ..., x_{i-1}) = C(x_{i-n+1}, ..., x_{i-1}, x_i)$$

3.2 Backoff-модель

Обучение модели Этап обучения модели практически такой же, как и в случае простой n-граммной модели, с разницей в том, что здесь собираются статистики для всех $n \leq n_{max}$.

Применение модели Идея backoff-подхода состоит в том, что при нехватке данных для оценки какой-либо n-граммы $x_{i-(n-1)}...x_{i-1}x_i$ постепенно уменьшается n, что позволяет увеличить общность алгоритма и оценить n-грамму по частям, но более надёжно. За счёт этого уменьшается вероятность переобучения модели на конкретных данных. Подробнее можно узнать в [11].

$$P_n(x_i|x_{i-n+1},...,x_{i-1}) = \begin{cases} C(x_i|x_{i-n+1},...,x_{i-1}) & \text{if } C(x_i|x_{i-n+1},...,x_{i-1}) > k \\ P_{n-1}(x_i|x_{i-n+2},...,x_{i-1}) & \text{otherwise} \end{cases}$$

3.3 Модель Катца (Katz)

Обучение модели Этап обучения модели такой же, как и в случае backoff-модели.

Применение модели Модель Катца является улучшенной версией backoff-модели, в которой накладывается динамический дисконт (коэффициенты $d_{w_{i-n+1}...w_i}$ и $\alpha_{w_{i-n+1}...w_{i-1}}$) на оценку n-граммы в случае уменьшения n. Более подробно о модели Катца можно прочитать в [4].

$$P_n\left(w_i|w_{i-n+1}...w_{i-1}\right) = \begin{cases} d_{w_{i-n+1}...w_i}\dfrac{C(w_{i-n+1}...w_i)}{C(w_{i-n+1}...w_{i-1})} & \text{if } C(w_{i-n+1}...w_i) > k \\ \alpha_{w_{i-n+1}...w_{i-1}}P_{n-1}\left(w_i|w_{i-n+2}...w_{i-1}\right) & \text{otherwise} \end{cases}$$

При этом коэффициенты $d_{w_{i-n+1}...w_i}$ и $\alpha_{w_{i-n+1}...w_{i-1}}$ вычисляются следующим образом:

$d_{w_{i-n+1}...w_i}$ – размер дисконта, вызванного сглаживанием (см. разд. 4)

$$\alpha_{w_{i-n+1}...w_{i-1}} - \text{ размер дисконта за backoff } -$$
$$= \frac{\beta_{w_{i-n+1}...w_{i-1}}}{\Sigma_{\{w_i:C(w_{i-n+1}...w_{i-1})\leq k\}} P_{n-1}\left(w_i|w_{i-n+2}...w_{i-1}\right)},$$

где $\beta_{w_{i-n+1}...w_{i-1}}$ – остаточная плотность вероятности для $(n-1)$-грамм –

$$= 1 - \Sigma_{\{w_i:C(w_{i-n+1}...w_i)>k\}} \frac{C(w_{i-n+1}...w_i)}{C(w_{i-n+1}...w_{i-1})}$$

4 Описание эксперимента

Для исследования способов исправлять ошибки OCR в тексте необходимо либо иметь большой корпус размеченных данных с результатами распознавания, либо построить некоторую модель частотных ошибок OCR и эмулировать их самостоятельно. Для данной работы был выбран второй подход, поскольку это убирало необходимость интегрироваться с какой-либо OCR машиной (например, ABBYY FineReader). Подобная интеграция могла бы быть достаточно трудоёмкой задачей, не являясь при этом основной целью исследования.

4.1 Корпус

Для обучения и сравнения n-граммных моделей использовался корпус html-страниц с ряда японских сайтов TODO: Каких сайтов? общим размером $\approx 8,5GB$.

В корпусе было 163244 html-страницы со средним размером $\approx 52kB$.

Тексты из этого корпуса не были результатом OCR, поэтому в них не должно было быть ошибок, связанных с распознаванием. Эти тексты были признаны верными с точки зрения языка и подходящими для обучения моделей.

В рамках подготовки корпуса к эксперименту тексты были перемешаны, чтобы тематика текста не зависела от его исходного положения в корпусе, из текстов были удалены html-теги, сложное форматирование, небольшое число мусорных символов. Также корпус был единообразно переведён в кодировку Unicode. Подробнее об этих технических этапах – см. раздел разд. 5.

После вышеперечисленных операций корпус был готов к использованию, его размер составлял $\approx 1,7GB$. При этом размер алфавита в нашем корпусе составлял ≈ 7000 символов, что в 3 раза меньше размера таблиц Unicode.

Имея данные, готовые к использованию, проанализируем их.

4.2 Обработка корпуса

После обработки корпуса его можно было описать следующими цифрами:

Параметр	Значение
Размер (kB)	1 721 504
Символов	640 604 961
среди них уникальных	6 861
Предложений	79 497 345

TODO: таблицы и рисунки лейблы

Если посмотреть на распределение частот отдельных символов (рис. 7), то оно выглядит так:

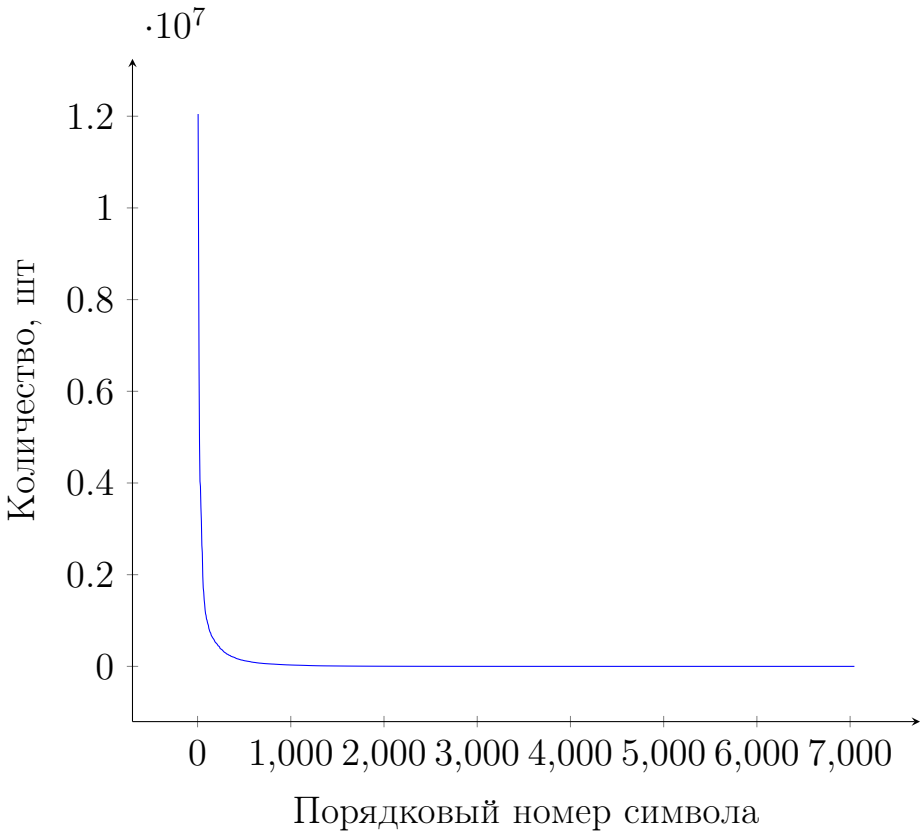


Рис. 7: Распределение частот униграмм

Видно, что распределение похоже на обратное экспоненциальное (кстати, это же утверждает закон Ципфа (Zipf, см. [11])). Проверим эту гипотезу, построив график обратного логарифма (рис. 8):

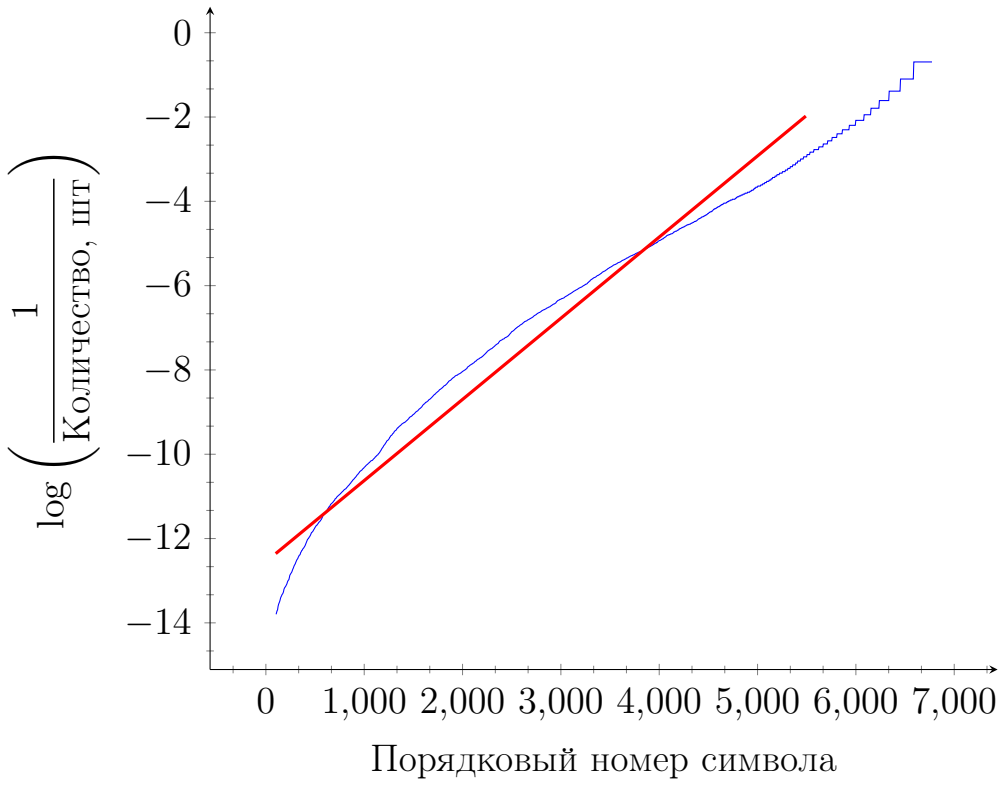


Рис. 8: Проверка закона Ципфа

Действительно, этот график с достаточной точностью ложится на прямую, выбиваясь из неё только в начале списка символов (там находятся самые частотные кандзи, а также практически вся хирагана/катакана, подробнее про распределения различных подмножеств символов см. разд. 4.4). Тем самым, эмпирический закон Ципфа был проверен на ещё одном корпусе.

Посмотрев на рис. 7, можно также заметить, что только очень малая часть символов появляется большое число раз. Посмотрим поближе на "голову" того же распределения (рис. 9):

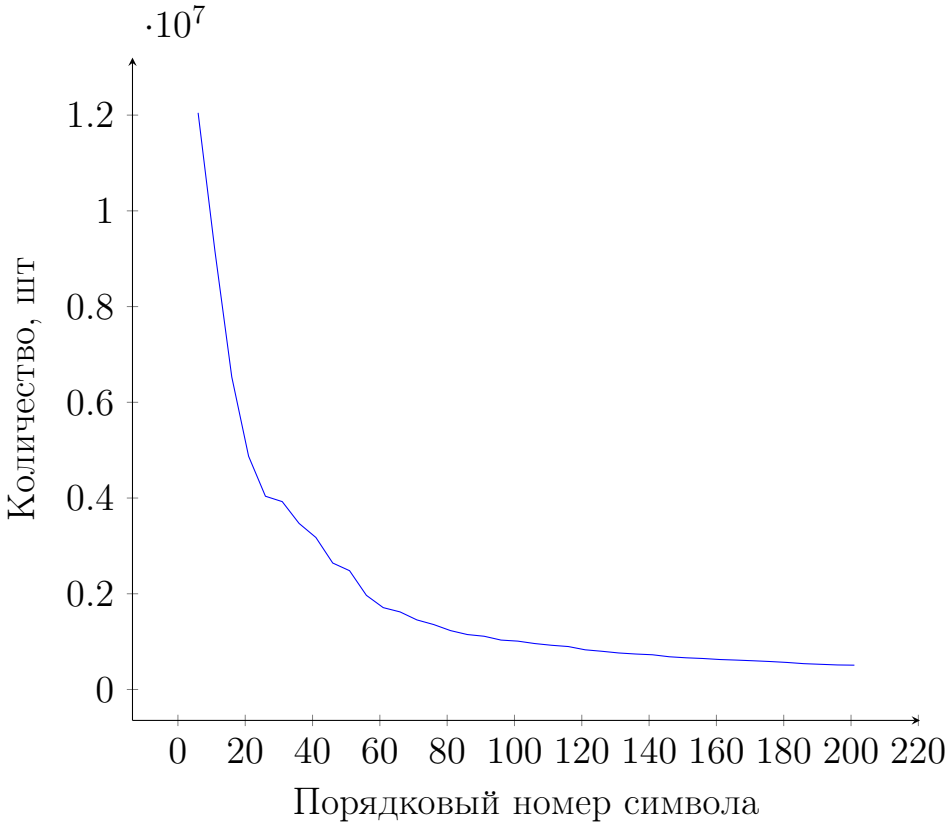


Рис. 9: Частоты униграмм – голова распределения

Действительно, лишь ≈ 200 символов встречаются достаточно часто.

Осталюся ещё примерно 6500 символов, которые входят в алфавит, но статистически мало отличаются от тех символов, что вовсе не встретились в нашем корпусе. Для оптимизации времени работы и занимаемой памяти эти символы можно представить более сжато.

Определение 4.2.1. *Корзина (бакет, bucket)* – множество символов, которые считаются статистически малозначимыми и заменяются на U+FFFD (Unicode Replacement Character).

Бакет B_i характеризуется числом $|\Sigma_{B_i}|$ – размером алфавита, который остаётся после сливания некоторого хвоста распределения в бакет. Было решено рассматривать бакеты с алфавитами размером $|\Sigma_{B_i}| = \{7000, 4800, 2600, 200\}$, поскольку примерно на эти размеры алфавитов приходятся изменения в характере убывания частот символов. При этом бакет $|\Sigma_{B_i}| = 7000$ представляет собой исходный корпус.

Основным бакетом впоследствии был выбран $B^* : |\Sigma_{B^*}| = 4800$, поскольку при нём, с одной стороны, реализуется **сглаживание** хвоста распределения, что косвенно даёт возможность учитывать символы, не встретившиеся в обучающей выборке. С другой стороны, размер алфавита остаётся достаточно большим, что будет важно при рассмотрении шумов (см. разд. 4.4)

На рис. 10 схематично изображено распределение частот после применения бакета с $|\Sigma_{B_i}| = 2600$.

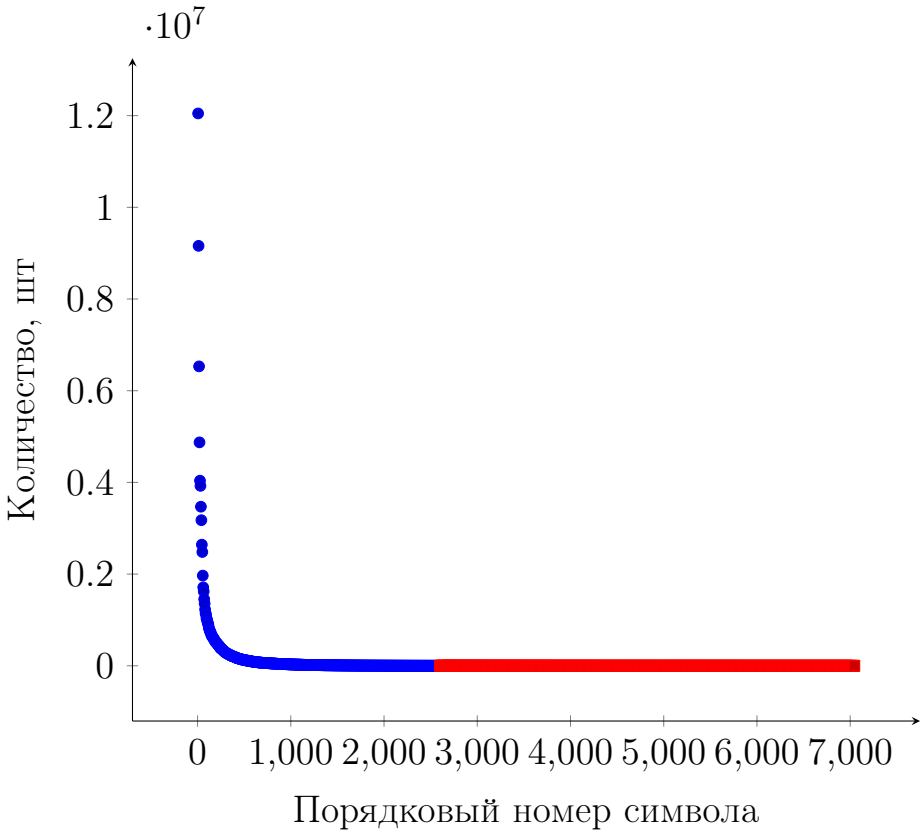


Рис. 10: Распределение частот униграмм: бакет с $|\Sigma_{B_i}| = 2600$

Поскольку нам были недоступны корпуса текстов, распознанные какой-либо OCR машиной, было принято решение эмулировать ошибки OCR самим. Это делалось при помощи генератора шума.

4.3 Генератор шума и режимы его работы

Определение 4.3.1. *Шум* $Noise = \{(a_1, a_2), (b_1, b_2, b_3), (c_1, c_2), ...\}$ – мно-

жество наборов символов алфавита Σ, которые легко спутать при распознавании. Конкретные шумы определяются эмпирически.

Для эмуляции ошибок OCR был разработан скрипт – генератор шума. Он параметризуется конкретным шумом и частотой его применения.

Определение 4.3.2. *Генератор шума* – настраиваемый скрипт, который принимает эталонное предложение S, находит в нём символы-представители наборов конкретного шума $x \in S \mid \exists \xi = \{\xi_1, \xi_2, ..., \xi_l\} \in Noise : x \in \xi$, и случайным образом меняет эти символы x на "шумовые"из соответствующего набора ξ.

С помощью шума $Noise$ случайным образом генерируются ошибки в предложениях текста $Text$. Таким образом происходит стохастическая эмуляция ошибок OCR.

Тестовая часть корпуса была разбита на предложения (см. формальную постановку задачи в разделе разд. 1), которые независимо друг от друга зашумлялись. Эти предложения после зашумления подавались на вход оценивающему алгоритму Θ, который выбирал лучший из предложенных вариантов.

Были определены следующие шумы, обоснование выбора см. в разделе разд. 1:

KaGa Наборы симвопов, соответствующие добавлению диакритики. Например,

かが	しじ	たな
きぎ	すず	だな
くぐ	ふぶぷ	んだ
けげ	そぞ	ちぢ
こご	ただ	ほぼぽ
さざ	なに	. . .

BigSmall Большие/маленькие написания букв:

あぁ	つっ	アァ
いぃ	やゃ	イィ
うぅ	ゆゅ	ウゥ
えぇ	よょ	エェ
おぉ	わゎ	. . .

Mix Комбинация предыдущих режимов.

かが	ふぶぷ	んだ
あぁ	そぞ	ちぢ
いぃ	ただ	ほぼぽ
うぅ	なに	. . .
えぇ	たな	
おぉ	だな	

Интересно понимать, как выглядит результат работы генератора шума. Предположим, на вход генератору было дано следующее предложение:

キャンペーンは終了致しました。

Тогда для различных шумов и режима "1 символ на предложение"получались такие результаты, которые затем фиксировались.

Шум	Текст
Эталон	キャンペーンは終了致しました。
KaGa	ギャンペーンば終了致しました。
BigSmall	キヤンペーンは終了致しました。
Mix	ギャンペーンは終了致しました。

4.4 Статистика по шумовым символам

Посмотрим на то, где в общем распределении символов лежат шумовые символы. На всех графиках этого раздела пунктиром показана граница основного бакета B^* с $|\Sigma_{B^*}| = 4800$.

KaGa Рассмотрим, как выглядят **KaGa** шумовые символы на фоне всех остальных в корпусе (шкала логарифмическая, рис. 11).

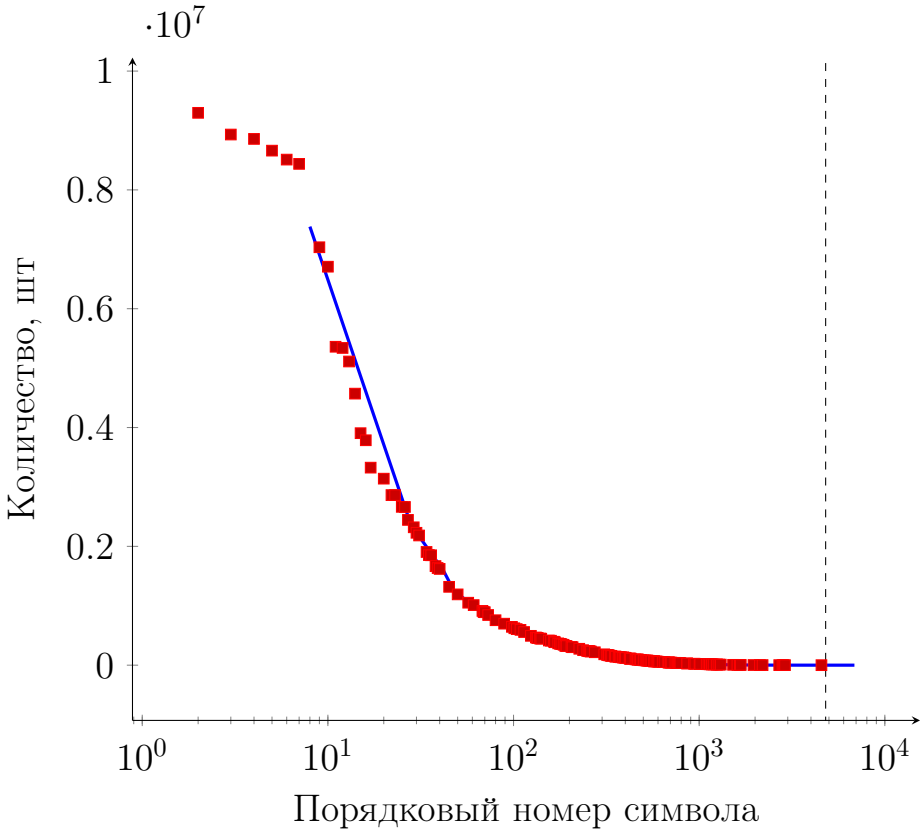


Рис. 11: Распределение частот шума **KaGa**

Видно, что шумовые символы лежат достаточно равномерно по кривой распределения, что может в будущем помешать оцениванию моделей для небольших размеров алфавита.

BigSmall Приведём аналогичный график для шума, состоящего из больших и маленьких кан (шкала логарифмическая, рис. 12).

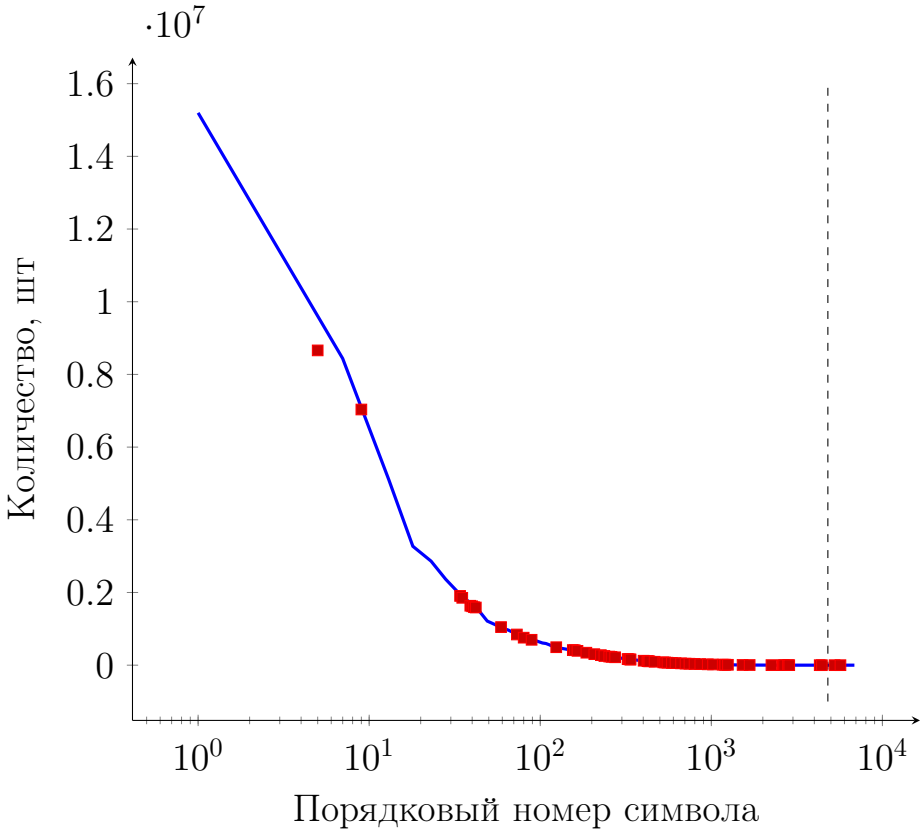


Рис. 12: Распределение частот шума **BigSmall**

В этом случае видно, что шумовые маленькие буквы в основном сконцентрированы в хвосте распределения. Поэтому для малых размеров алфавита работа с этим шумом бесполезна.

Mix Если смешать эти 2 шума (**KaGa** и **BigSmall**) и построить такой же график для смеси, то результат предсказуемо будет являть собой наложение двух предыдущих графиков (шкала логарифмическая, рис. 13)

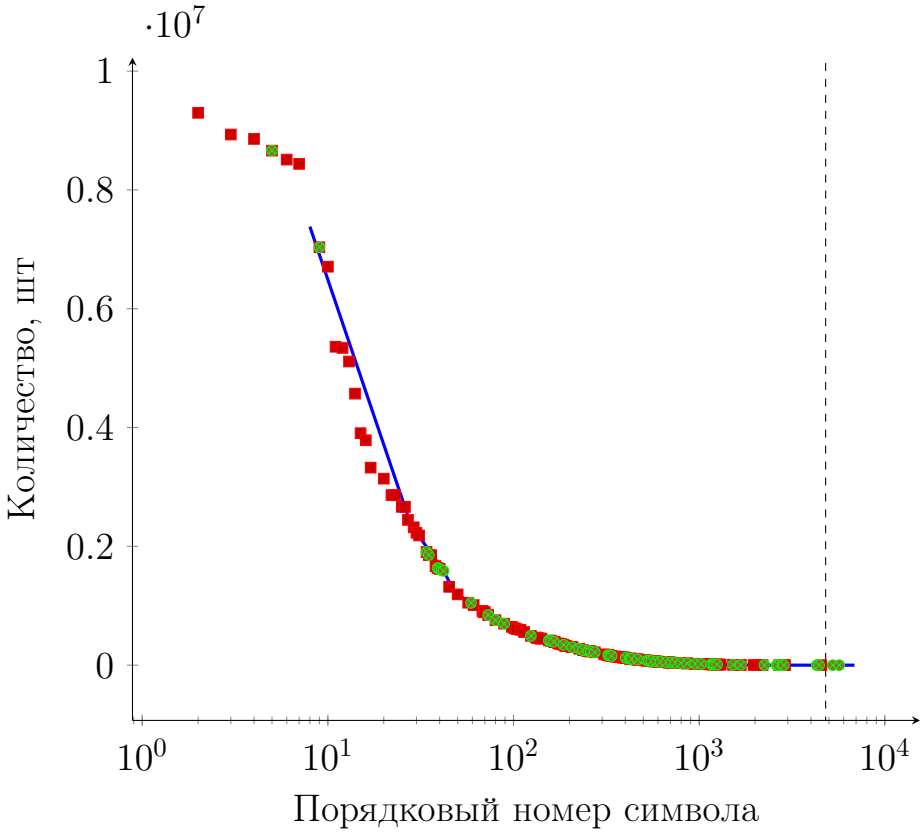


Рис. 13: Распределение частот шума **Mix**

4.5 Ход эксперимента

Глобально эксперимент состоял из следующих этапов:

1. Обучение модели:
   (a) Получение n-грамм для обучающей выборки;
   (b) Подсчёт статистики по n-граммам;
   (c) Сериализация статистики на диск;
2. Применение модели:
   (a) Десериализация обученной модели;
   (b) Получение n-грамм для тестовой выборки;
   (c) Оценивание моделью различных вариантов написания для предложения;

(d) Вычисление оценки модели.

Определение 4.5.1. *Уверенность (confidence)* – порог отношения оценок величины $\frac{\max(\xi_1, \xi_2)}{\min(\xi_1, \xi_2)} = Confidence$, при достижении которого оценки ξ_1, ξ_2 величины ξ считаются неразличимыми.

При этом на этапе оценивания вклада результата работы модели на конкретном предложении (п. 2d) существовало 2 альтернативы:

1. $\Theta(S_{good}) > \Theta(S_{noise}) \Rightarrow$ на этом предложении модель отработала хорошо, иначе – плохо.
2. $\frac{\max(\Theta(S_{good}), \Theta(S_{noise}))}{\min(\Theta(S_{good}), \Theta(S_{noise}))} < Confidence \Rightarrow$ на этом предложении модель отработала уверенно, классификация проводится, далее см. п. 1. Если же различие между оценками меньше порога $Confidence$, то результат – отказ от классификации этого случая.

4.6 Baseline эксперимента

В качестве бейзлайна эксперимента были выбраны результаты работы униграммной модели (n-граммная при $n = 1$) (результаты показаны для основного бакета, $|\Sigma_{B^*}| = 4800$). Оценка производилась без учёта уверенности.

Для разных шумов бейзлайн показал следующие результаты:

Шум	Оценка модели
KaGa	0.75
BigSmall	0.88
Mix	0.76

Гипотеза: шум **BigSmall** отличить проще, т.к. маленькие буквы в принципе встречаются реже, чем символы из **KaGa**. TODO: надо ли?

5 Реализация модели

Из-за обилия задач, связанных с обработкой текста в Unicode, а также по причине наличия удобных специализированных библиотек в качестве основного языка программирования был выбран Python 3. Впрочем, отдельные задачи, связанные с предобработкой корпуса, писались на Bash. Подробные спецификации использованного софта и аппаратные характеристики используемой машины см. в ?? A.

5.1 Общее окружение: nltk, pygtrie

Поскольку в ходе эксперимента считались статистики по 7-граммам для корпуса из $79 \cdot 10^6$ предложений, было необходимо хранить статистики эффективно. Для хранения строк подобного вида лучше всего подходит такая структура данных, как префиксное дерево (бор, trie).

Поскольку задачи писать эффективное и масштабируемое префиксное дерево не было, была выбрана его реализация, предоставленная Google в библиотеке pygtrie (см. [5], [6]).

Для получения n-грамм, а также различных вспомогательных задач был выбран популярный python-пакет nltk (Natural Language ToolKit, см. [7])

5.2 Полезные утилиты

Pickle Удобный модуль для сериализации/десериализации сложных Python-объектов.

BeautifulSoup.UnicodeDammit Спасительный модуль для работы с разнообразными кодировками в составе пакета BeautifulSoup. Особенно мощно работает в связке с библиотеками chardet и cchardet. Без него привести корпус к читабельному виду было бы невозможно.

В качестве подтверждения – статистика по различным кодировкам в исходном корпусе:

Кодировка	Количество файлов
utf-8	68789
iso-8859-2	46870
shift_jis	42015
euc-jp	2562
cp932	1575
ascii	544
windows-1253	436
iso-8859-7	256
windows-1252	78
iso-8859-5	9
gb2312	4
tis-620	4
ibm866	1
maccyrillic	1
Всего	163144

Документацию можно найти в [8].

6 Результаты эксперимента

Напомним условия эксперимента. Модели Ngram($n = 1$), Backoff($n = 5$) и Katz($n = 5$) запускались на тестовых выборках размером $\approx 300MB$, что соответствует $\approx 8 \cdot 10^6$ предложений, или же $\approx \frac{1}{10}$ части корпуса. При этом размер алфавита был равен $|\Sigma B^*| = 4800$, выборки были зашумлены 3 различными способами: **KaGa**, **BigSmall** и **Mix**.

6.1 Результаты для различных моделей

Первая часть эксперимента проводилась без оценивания уверенности ответов (см. разд. 4.5) и показала следующие результаты:

$M \backslash N$	**KaGa**	**BigSmall**	**Mix**
Ngram(1) (Baseline)	0.75	0.88	0.76
Backoff(5)	0.89	0.93	0.90
Katz(5)	0.96	0.96	0.96

TODO: сюда ли красивую таблицу со статистиками по канам?

После того, как в первой части эксперимента хорошие результаты показала модель Katz(5), для неё были проведены испытания по поиску оптимального уровня уверенности $Confidence, C$:

$C \backslash N$	**KaGa**	**BigSmall**	**Mix**
0.9	0.86	0.94	0.86
0.95	0.979	0.988	0.981
0.97	0.976	0.986	0.964
0.99	0.94	0.98	0.95

$C \backslash N$	**KaGa**	**BigSmall**	**Mix**
0.9	0.53	0.43	0.52
0.95	0.61	0.69	0.62
0.97	0.77	0.85	0.77
0.99	0.91	0.94	0.91

В итоге оптимальная конфигурация – модель Катца с $n = 5$ и уверенностью $= 0.97$ (Katz($n = 5, C = 0.97$)). Она даёт высокие результаты исправления ошибок ($0.98 - 0.99$), классифицируя достаточно большую часть

выборки ($77 - 85\%$). Результаты работы этой конфигурации в будут более детально проанализированы в разд. 7.

6.2 Затраты ресурсов

TODO: написать

7 Анализ результатов

В итоге оптимальной была признана модель Катца с глубиной бора $n = 5$, результаты которой ещё улучшились после добавления опции отказа от классификации при недостаточной уверенности.

7.1 Сравнение с другими методами

7.2 Анализ ошибок

8 Заключение

В работе проанализированы некоторые модели машинного обучения, основанные на символьных n-граммах, эти модели были сравнены в качестве решений для автоматического исправления ошибок OCR. Исследования были проведены для различных характеров ошибок OCR, которые имитировались в процессе эксперимента.

Также были проведены измерения производительности этих подходов с точек зрения времени и памяти. В силу особенностей японского языка затраты памяти на работу с обученной моделью впечатляют.

Возможными направлениями дальнейшего развития данной темы могли бы стать исследованияп с реальными OCR-ошибками, или привлечение знаний о грамматической структуре языка для выделения важных частных случаев.

Литература

[1] *Soumendu Das et al.* Survey of Pattern Recognition Approaches in Japanese Character Recognition //
(IJCSIT) International Journal of Computer Science and Information Technologies, Vol. 5(1), 2014. P. 93 - 99.

[2] *Nagata, Masaaki* Japanese OCR Error Correction using Character Shape Similarity and Statistical Language Model //
Proceedings of the 17th International Conference on Computational Linguistics - Volume 2, 1998. P. 922 - 928.

[3] *Nagata, Masaaki* Context-based Spelling Correction for Japanese OCR //
Proceedings of the 16th Conference on Computational Linguistics - Volume 2, 1996. P. 806 - 811.

[4] *S. Katz* Estimation of probabilities from sparse data for the language model component of a speech recognizer. //
IEEE Transactions on Acoustics, Speech, and Signal Processing, vol. 35, no. 3, 1987. P. 400 - 401.

[5] Pygtrie documentation [Электронный ресурс].
URL: http://pygtrie.readthedocs.io/en/latest/ – 2014

[6] Pygtrie github [Электронный ресурс].
URL: https://github.com/google/pygtrie – 2017

[7] Natural Language Toolkit [Электронный ресурс].
URL: http://www.nltk.org/ – 2017

[8] Beautiful Soup Documentation [Электронный ресурс].
URL: https://www.crummy.com/software/BeautifulSoup/bs4/doc/ – 2015

[9] *Tsai, Charlie* Recognizing Handwritten Japanese Characters Using Deep Convolutional Neural Networks //
URL: http://cs231n.stanford.edu/reports/2016/pdfs/262_Report.

[10] Update Registration 87 Japanese Graphic Character Set for Information Interchange [Электронный ресурс].
URL: https://web.archive.org/web/20150318013247/
http://kikaku.itscj.ipsj.or.jp/ISO-IR/168.pdf – 1992

[11] *Christopher D. Manning, Hinrich Schütze* Foundations of Statistical Natural Language Processing // MIT Press, 1999.

Приложение А.

А.1 Аппаратные характеристики платформы

Архитектура	x86-64
Частота процессора	3.30 ГГц
Количество процессоров	32
Оперативная память	192 Гб

А.2 Используемое программное обеспечение

Python	3.5.2 64-bit