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

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 (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. TODO: сказать, почему забиваем на фуригану и т.д.

Рассмотрим эти символы подробнее:

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

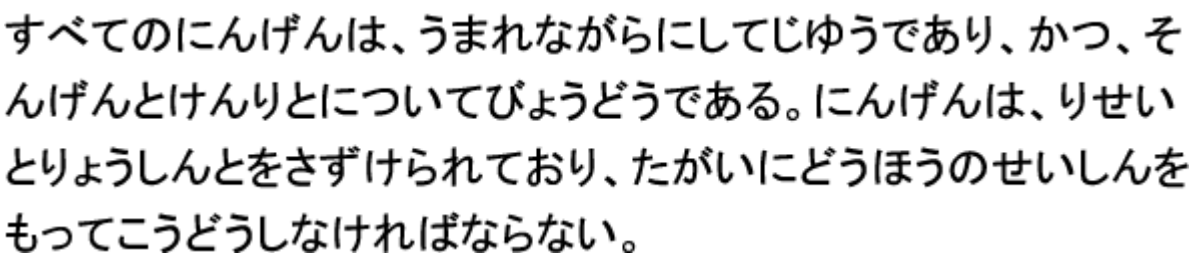


Рис. 1: hirag_sample

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

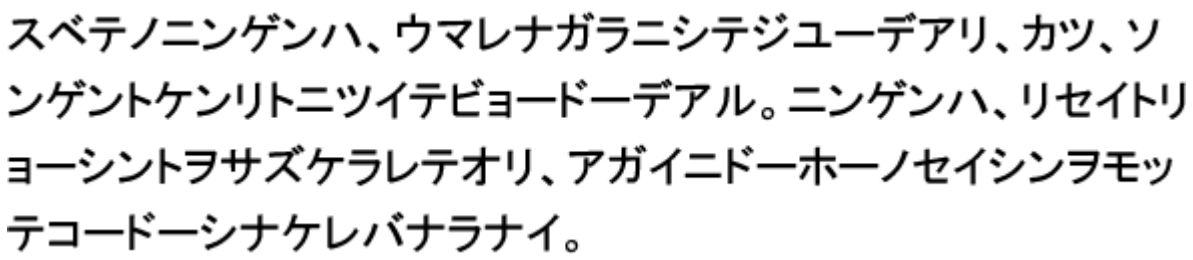


Рис. 2: katak_sample

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

Рис. 3: dakut_sample

Рис. 4: dakut_sample

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

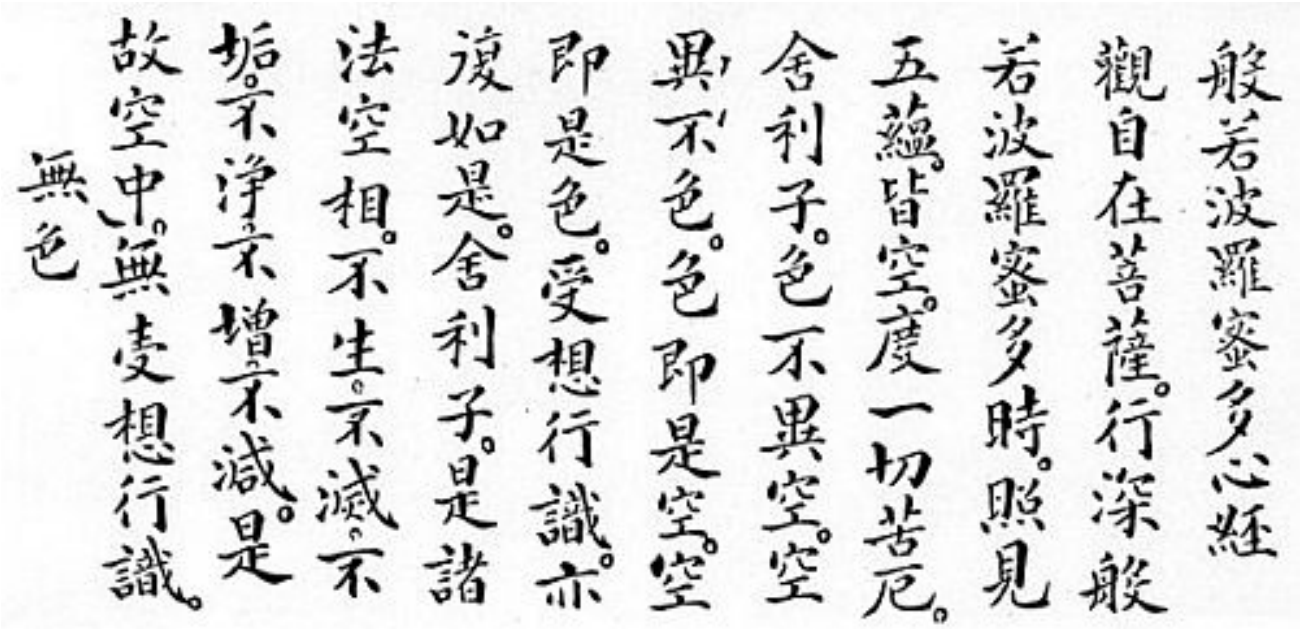


Рис. 5: kandji_sample

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?.

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

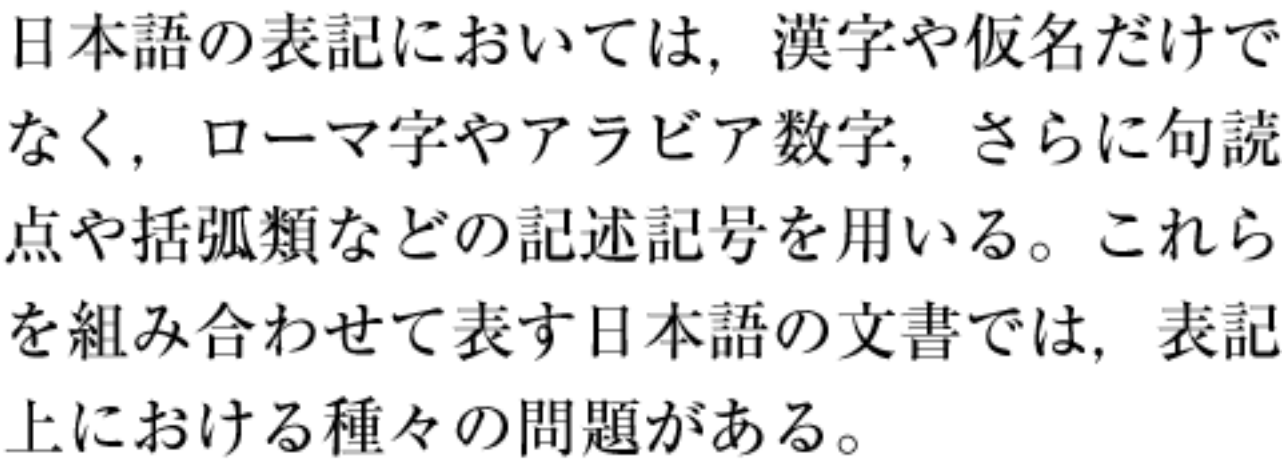


Рис. 6: japtext

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

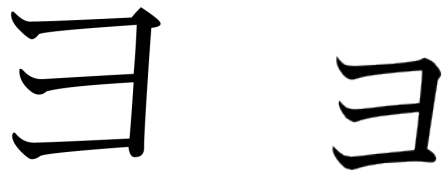
2Kana 2 похожие каны. Таких случаев достаточно мало, а методы их различения уже существуют (см., например, метод с использованием глубокого обучения и свёрточных нейронных сетей в работе [9])

TODO: и

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

и

BigSmall Существуют большие и маленькие каны, которые нужно различать.



и

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 TODO: а что делаем с омонимией?, 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 3), основанных на n-граммных моделях, и сравнить их по качеству.

2 Обзор источников

- Banerjee.
- Nagata - actual
- Nagata - old

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 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].

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

TODO: Рассказать про коэффициенты

3.4 TODO: Maybe Kneser-Ney

4 Описание эксперимента

Для исследования способов исправлять ошибки OCR в тексте необходимо либо иметь большой корпус размеченных данных с результатами распознавания, либо как-то выкручиваться.

Пришлось выкручиваться, эмулируя ошибки OCR самостоятельно.

TODO: Описание машины – в приложение

4.1 Корпус

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

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 5.

После вышеперечисленных операций корпус был готов к использованию, его размер составлял $\approx 1,5GB$. При этом размер алфавита в нашем корпусе составлял ≈ 7000 символов, что в 3 раза меньше размера таблиц Unicode.

Имея данные, готовые к использованию, было бы глупо не построить по ним несколько графиков.

4.2 TODO: Zipf

Если посмотреть на распределение частот отдельных символов, то оно выглядело так:

Рис. 7: 1gramstats

Видно, что распределение похоже на обратно экспоненциальное (кстати, это же утверждает закон Ципфа TODO: link). Проверим эту гипотезу, построив график обратного логарифма (TODO: формулы):

Рис. 8: zipf

Действительно, этот график с достаточной точностью ложится на прямую (TODO: погрешности?). Тем самым, в NLP закон Ципфа проверен ещё раз.

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: 1gramstats head

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.

На Рис. 10 схематично изображено распределение частот после применения бакета с $|\Sigma_{B_i}| = 200$.

Рис. 10: bucket

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* $Noise = \{(a_1, a_2), (b_1, b_2, b_3), (c_1, c_2), ...\}$ – множество 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 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 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	つづ
さざ	なに	ほぼぽ

HalfWidth Полуширинные/полноширинная катакана:

ﾇヌ	ｳウ	ﾔヤ
ｱア	ｴエ	ｭユ
ｲイ	ｵオ	ﾇヌ

ッツ	ㇰイ	ㇹオ
-ー	ㇴウ	ㇵカ
ㇷア	ㇶエ	ㇷ゚キ

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	オォ

Mix Комбинация предыдущих режимов.

かが	ュユ	イィ
きぎ	ョヨ	ウゥ
くぐ	ッツ	エェ
けげ	-ー	オォ
こご	ㇷア	
ㇵヤ	アァ	

TODO: Срез шума в Ципфе – график

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 (пер. с яп. Campaign has ended)
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。
HalfWidth	キャンペ-ンは終了致しました。
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 Baseline эксперимента

В качестве бейзлайна эксперимента была выбрана униграммная модель ($n = 1$).

Для разных шумов она показала следующие результаты, $|\Sigma_{B_i}| = 4800$:

Шум	Оценка модели
KaGa	0.70
HalfWidth	0.71
BigSmall	0.77
Mix	

TODO: достать и добавить результаты для остальных бакетов

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. Для подробных спецификаций использованного софта см. TODO: приложение.

5.1 Общее окружение: nltk, pygtrie

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.

Поскольку в ходе эксперимента считались статистики по 7-граммам для корпуса из TODO: статистики по строкам?, 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]). TODO: нормально ли репозиторий в cite?

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].

Graphviz Простая утилита и язык для визуализации графов, полезно для вглядывания в префиксные деревья.

5.3 Пример работы и статистики

TODO: Разобрать предложение и пройтись по этапам визуализации результата (до svg-картинки с траем) – если будет нужна красивая вода.

6 Результаты эксперимента

7 Анализ результатов

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 исследования 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.