С.В. Знаменский

Обучаемый алгоритм автоматического исправления ошибочной кодировки русского текста[1]

(Электронный препринт: `http://rustex.botik.ru/znamensk/recode/preprint.ps`)

Аннотация: описывается и обосновывается подход, позволяющий при увеличении объема данных всего в 2-3 раза одновременно в сотни раз повысить эффективность распознавания кодировки, что позволит в десятки раз расширить список испытываемых кодировок без снижения надежности и ощутимого увеличения среднего времени распознавания.

§1. Постановка задачи

Традиционно существующие различные кодировки русского текста в различных операционных системах создают немалые трудности при обмене информацией по сети.

Неоднократно предпринимавшиеся государством и крупнейшими фирмами попытки стандартизации кодировок не привели и вряд ли приведут к единой для всех кодировке русского текста.

Вопрос казался уже практически решенным касательно электронной почты, как только используемая кодировка стала явно указываться в MIME-заголовках сообщений. Действительно немного полегчало, но совсем немного.

Во-первых, оказалось, что далеко не каждый пользователь в состоянии самостоятельно правильно настроить свою почтовую программу.

Во-вторых, даже квалифицированным пользователям ошибки в распространенном программном обеспечении зачастую не дают возможности так настроить почтовые программы, чтобы они правильно указывали кодировки не только в обычных письмах, но и в письмах, содержащих цитированный из полученной почты или пересылаемый текст. Стало уже привычным и что новые версии почтовых программ преподносят новые и новые сюрпризы. Да кое-где еще остались бездумные автоматические перекодировщики, которые перекодируют файл не проверяя, в какой он кодировке, после чего расшифровка сообщения требует от пользователя уже весьма ощутимых усилий, хотя многим из них удается писать простые программы для получившейся таким образом кодировки, позволяющие полностью без потерь автоматически раскодировать такие сообщения. Встречаются и случаи необратимых потерь информации, но они более редки и их учет выходит за рамки данной работы.

В-третьих, наконец, оказалось, что позиции русской буквы Ё ё в кодировке koi8-r нередко отличаются от стандартной, а с украинскими буквами дело обстоит и того хуже: живут по меньшей мере два (возможно недокументированные) варианта кодировки в которых украинские буквы, общие с русскими, расположены в соответствии с cp866, а остальные украинские буквы на совсем других (отличных от cp866 и между собой) позициях.

Появление 16-битной кодовой таблицы UNICODE не привело, вопреки ожиданиям, к стремительному исчезновению 8-битных кодовых таблиц несмотря на то, что проблемы многообразия кодировок не являются специфичными для российских пользователей. Похоже, что возрастание вдвое объема, необходимого для хранения информации и другие проблемы распределения памяти, связанные с резким (256-кратным) увеличением числа символов в кодовой таблице, отсутствие в UNICODE значительной части букв некоторых живых языков и наличие большого числа 8-битных текстов, программ и длгоритмов для их обработки если и позволят снизить остроту проблемы кодировок с повсеместным внедрением UNICODE, не переводя ее в новую размерность, то уж навряд ли дадут решить ее полностью и окончательно в обозримой перспективе.

[1] Работа выполнена при поддержке РФФИ, грант 98-07-90179

В итоге значительную часть сообщений по-прежнему приходится перекодировать или расшифровывать вручную. На WWW без особого труда можно найти разные раскодировщики почтовых сообщений – от примитивных до более сложных, автоматически выявляющих правильную кодировку и даже вплоть до дорогостоящего многомегабайтного монстра, помогающего восстановить даже казалось бы безвозвратно потерянную информацию в письме, где все русские буквы представлены десятком латинских. При этом используется встроенный словарь русских слов, правильный выбор которых осуществляется при участии человека.

Еще хуже, чем с e-mail, обстоит дело с общением в режиме ON-LINE, и, в частности, с обработкой сервером запросов WWW-броузера. Здесь текст от клиента может пойти совсем не в той кодировке, в какой запрашивались документы с сервера и проблема автоматического определения кодировки и перекодировки проходящего текста становится еще более актуальной. Проблема усугубляется и тем, что сервер не вправе приостановить свою интенсивную работу в ожидании вмешательства человека, и ограниченностью ресурсов оперативной памяти, используемой одновременно порой для обработки сотен запросов, и немногословностью получаемых от клиента запросов.

Целью работы является описание алгоритма, который:

(1) без вмешательства человека принимает безошибочное решение о необходимости и типе перекодировки текстового файла;
(2) эффективен даже для файлов, содержащих всего одно или несколько русских слов (например, результат заполнения анкеты на WWW);
(3) работает быстро и использует небольшую долю оперативной памяти.

Поскольку в этом (как и в практически любом) техническом вопросе наивно было бы претендовать на окончательное решение, то не менее, а может быть и более важным, чем само построение и описание эффективного алгоритма, является описание и пояснение причин, по которым одни методы и подходы были признаны недостаточно эффективными и заменены другими.

§2. В каком смысле можно обеспечить безошибочность перекодировки и как это сделать

Вопреки достаточно распространенным ожиданиям, задача автоматического определения правильной кодировки не может быть алгоритмически разрешена для всех файлов, содержащих русский текст: даже если разрешить только две наиболее распространенные кодировки KOI-8 и CP-1251, то файл с единственным русским словом "БЕЙ" в KOI-8 будет интерпретироваться как файл в кодировке CP-1251 с единственным русским словом "век". Есть и еще пары: "сел"-"УЕМ", "вел"-"ЧЕМ". Это означает, что автоматическое исправление кодировки вполне может не произойти, если файл содержит менее четырех русских букв. В реальной жизни столь короткие текстовые файлы не встречаются даже в разумных запросах к поисковым системам INTERNET.

Даже для сколь угодно длинного текста ситуация не облегчается, если допустить произвольные заранее не регламентированные кодировки. Для доказательства этого тезиса легко сочинить сколь угодно длинный текст, в котором нет относительно редких заглавных русских букв "Э" и "Ю" кроме один раз встретившегося женского имени "Юля" или "Эля". Если бы указанный алгоритм существовал, он должен был бы угадать, какое имя из этих двух упомянуто в тексте. Вероятность правильного угадывания в данном случае равна 1/2 и никакой дополнительной информации-подсказки нет. Ясно, что в описанной ситуации он не может быть безошибочным. Более того, Таким образом, корректная постановка задачи автоматического определения требует фиксации конечного числа реально встречающихся в интеренете кодировок, включая ошибочные. К таким базовым кодировкам кириллических букв следует в первую очередь отнести кодировки iso8859-5 и наиболее распространенные вариации koi8, cp1251, maccyrillic и cp866/альтернативной кодировки, кодировку, получающуюся из cp1251 инверсией седьмого бита и кодировки, получающиеся из вышеперечисленных неправильной перекодировкой, как, например, текст в koi8 может неправильно настроенной программой быть принят за текст в cp1251 и перекодирован из cp1251 в koi8 или иную кодировку. Базовый список таких ошибочных кодировок предполагается подготовить заранее и предусмотреть возможность пополнения его новыми уродцами. Для этой цели удобно использовать конфигурационный файл который можно дополнять, не перекомпелируя саму программу.

Очевидно, что чем богаче набор включенных в конфигурационный файл возможных кодировок проходящего текста, тем больше опасность ошибки в распознавании кодировки. Поскольку для 64 русских букв имеется $128 * 127 * 126 * ... * 65 = 128!/65! > 10^{1}00$ возможных вариантов расположения в верхней части кодовой таблицы, то принципиальная возможность однозначного определения одной из десятка или пусть даже тысячи распространенных возможных кодировок не кажется совершенно нереальной. Проблемы просто и не было бы, если бы кодировки были выбраны случайным образом. Это заведомо не так и вполне вероятно, например, что текст, набранный в одной из кодировок только заглавными буквами, может совпасть с тем же текстом, набранным в другой кодировке только маленькими. Что делать в этом случае корректирующей кодировку программе? Означает ли это, что программа обречена на некорректную работу, а идея на дискредитацию?

Безусловно, могут существовать ситуации, когда необходимость сохранить регистр (заглавные или строчные буквы) в проходящем файле важнее возможности исправить действительно ошибочную кодировку и без участия человека задача решаться не должна. Однако такие ситуации настолько редки, что пример привести я затрудняюсь. Обычно же либо регистр не критичен, либо одной из конфликтующих таким образом кодировок можно пренебречь. В соответствии с этим следует различать фатальную и терпимую неоднозначность трактовки проходящего текста и на всякий случай предостеречь от использования алгоритма в системах где любая неоднозначность нетерпима, в первую очередь в системах, где критическому анализу случайно могли бы подвергаться нетекстовые файлы или где есть реальные гарантии правильности кодировки.

В каких-то системах при неоднозначной трактовке проходящего файла может оказаться удобной выдача всех правдоподобных расшифровок. В большинстве случаев это неудобно и предлагается более конструктивный выход - придать каждой кодировке индекс распространенности, который должен учитываться в случае неоднозначной трактовки проходящего файла. При этом грубо говоря вероятность ошибочной трактовки окажется равной произведению (предположительно очень малой) вероятности того, что проходящий файл может быть фатально неоднозначно трактован, на (опять же очень малую) вероятность использования этой редкой кодировки. При правильной настройке системы малейший риск испортить правильную кодировку должен приводить к отказу от перекодировки и сохранению файла в исходном виде.

Итак, при правильной настройке применение оптимизированного алгоритма должно, по-видимому, позволить в десятки и возможно даже в сотни раз снизить риск увидеть абракадабру в полученном тексте. Это особенно важно в тех достаточно распространенных ситуациях, когда маловероятная возможность ошибочной перекодировки менее критична, чем значительно более вероятное получение нечитаемого сообщения: при выдаче e-mail и заполнении анкет поисковых систем на www. В остальных случаях возможность и целесообразность применения алгоритма должна тщательно анализироваться.

Из сказанного следует еще один, тоже не самый веселый, вывод: хотя и можно будет добавлять новые кодировки для учета корректирующей программой, но каждый раз при этом придется решать задачу минимизации риска ошибки с учетом возможности неоднозначной интерпретации проходящего текста. Задача эта очевидно алгоритмически неразрешима хотя бы уже потому, что не видится сколько-нибудь надежного способа четко определить, ни что именно может распространяться в виде файла на русском языке, ни какие именно кодировки распространены в сети, ни какова вероятность заполучить письмо в той или иной из них, тем более, что все это к тому же изменяется с появлением очередных версий коммуникационных программ, исправляющих старые ошибки и добавляющих новые. Единственный просматривающийся подход к этой проблеме состоит в сборе и полуавтоматизированном учете вновь поступающей информации о бродящих по сети текстах и кодировках.

Таким образом, рассматриваемая задача распалась на следующие подзадачи:

(1) Быстрый, не требующий очень большой памяти, гибкий и оптимизируемый в отношениях надежности алгоритм «is_it_text» определения того, насколько данный файл выглядит похоже на русский текст в заданной кодировке, возвращающий положительное значение «doubt_index» индекса сомнительности.

(2) Обучаемый по образцам встречающихся текстов модификатор настроек этого алгоритма.

(3) Расширяемый список описаний кодировок, реально встречающихся в сети по убыванию распространенности.

(4) Обучаемый по коллекции сомнительно определяемых текстов формирователь надежных приоритетов для определения кодировок в сомнительных случаях.

§3. Идеи алгоритма выделения текстовых файлов в правильной кодировке

Ключевым в рассматриваемой проблеме является алгоритм, позволяющий с высокой надежностью определить правильность указанной кодировки текстового файла.

Можно было бы, конечно, подключить программу проверки правописания и проверять соответствие проходящей информации нормам русского языка. У такого подхода есть два принципиальных недостатка: потребность в значительных ресурсах (словарь русских слов, правила сочетания падежей и т.п. - речь должна идти о мегабайтах информации, что неприемлемо) и, что не менее важно - это нетерпимость к любым отклонениям от стандарта русского языка - будь то нераспространенные фамилии и имена в анкетах для WWW, текст на украинском или болгарском языках, специальная терминология или другие новообразования.

Есть и в этих отношениях противоположный - простой, быстрый и некритичный к новообразованиям подход, основанный на сочетаниях заглавных и строчных букв. Рассмотрим для примера следующее письмо:

Я к Вам пишу: Чего же боле... Татьяна.

Если письмо пришло в кодировке KOI-8, а читается шрифтами в CP-1251, то на экране текст письма выглядит следующим образом:

с Л чБН РЙЫХ: юЕЗП ЦЕ ВПМЕ... фБФШСОБ.

Если же наоборот, письмо было в CP-1251, а читается в KOI-8, то оно выглядит примерно так:

ъ Й бЮЛ ОХЬС: вЕЦН ФЕ АНКЕ... рЮРЭЪМЮ.

Немедленно бросается в глаза существенное отличие правильного текста от неправильного: если в исходном тексте в некоторых словах первая буква заглавная, остальные строчные, то в ошибочно прочитанном - наоборот. Зачастую уже известно, в какой из перечисленных двух кодировок может оказаться текст, поскольку известно, в какой кодировке он читается и это может позволить уже на основе отслеживания сочетаний "заглавная-строчная" получить желаемый результат.

Если письмо из KOI-8 прочитано в iso1889-5, то картина еще более усложнится, но неправильность чередования заглавных и строчных букв проявится еще ярче:

ё Ы їСЭ аЩле: ВХЧЯ жХ ТЯЬХ... єСдибЮС.

К сожалению, этот подход слишком прост даже для того, чтобы надежно выбрать из двух кодировок правильную: достаточно набрать весь текст в одном регистре (только заглавными буквами) или другой текст только строчными буквами, чтобы сделать этот алгоритм полностью беспомощным. Не годится он и для исправления файлов, действительно испорченных в результате не столь уж редких обратных перекодировок (KOI-8 -¿ CP-1251 вместо CP-1251 -¿ KOI-8 или наоборот), при которых то же письмо выглядит с точки зрения чередования заглавных и строчных вполне пристойно:

У м Юво ткщи: Аеър ге чрне... Жвжыупв.

З и Ачк нуэя: Бежм те юмйе... Пчпщзлч.

Все это наводит на мысль о разумной середине между использованием лишь минимальных сведений о структуре русского текста (как чередование заглавных и срочных букв) и больших словарей.

Интересен в этом отношении широко распространившийся последнее время алгоритм автоматического определения кодировки проходящего файла, основанный на подсчете статистике частот содержащихся в нем русских букв и сравнении ее с посчитанной для "эталонного" текстового файла на русском языке. Этот алгоритм блестяще реализован, например, в популярной программе Е. Рошаля "FAR" управления файлами под Windows-95. Он абсолютно правильно определяет кодировку достаточно длинного русского текста (как правило одной строки букв в 30-60 уже вполне достаточно) и предусматривает реальную возможность дополнения распознавателя кодировки данными по новым языкам и кодировкам.

Относительной частотой вхождения буквы в русский текст или просто частотой буквы называется отношение количества вхождений данной буквы к общему количеству символов с кодами больше 128 в файле. Мне доподлинно не известно, какой именно алгоритм учета относительных частот вхождения букв в русский текст использует Е.Рошаль. Самый естественный на первый взгляд способ - это считать частоты координатами вектора в пространстве, размерность которого равна числу букв. Такой вектор относительных частот однозначно сопоставляется каждому документу. Показателем правильности выбора кодировки служит расстояние между векторами, сосчитанными для испытуемого файла и для заранее заготовленного длинного текста на русском языке. Расстояние обычно считается как корень из суммы квадратов разностей соответствующих компонент, но существует бесчисленное множество иных подходов к определению расстояния. Для того, чтобы показать, чем плохо обычное евклидово расстояние в данном случае, приведу для примера векторы относительных частот, посчитанные для данного текста в момент написания этой фразы (из текста удалены примеры и весь текст переведен в регистр строчных букв, его объем 18kb) и для словаря Зализняка объемом 1.2 Mb:

Таблица количества букв и их частот (в скобках)

буква	в тексте	в словаре	$\frac{N_x/n}{N'_x/n'}$	$N_x/n - N'_x/n'$	$(N_x/n - N'_x/n')^2$
Ъ	7 (0.05%)	383 (0.04%)	0.8	-0.01	0.0001
Э	49 (0.33%)	1471 (0.15%)	0.45	-0.18	0.0324
Ц	57 (0.38%)	6283 (0.68%)	1.79	0.30	0.09
Ш	68 (0.45%)	6327 (0.68%)	1.51	0.23	0.0529
Щ	75 (0.50%)	3681 (0.39%)	0.78	-0.11	0.0121
Ф	75 (0.50%)	4438 (0.48%)	0.96	-0.02	0.0004
Ю	100 (0.67%)	2208 (0.23%)	0.34	-0.44	0.1936
Ж	134 (0.89%)	6445 (0.70%)	0.79	-0.19	0.0361
Х	177 (1.18%)	5603 (0.60%)	0.51	-0.58	0.3364
Г	181 (1.21%)	12079 (1.31%)	1.08	0.1	0.01
Й	226 (1.51%)	23193 (2.52%)	1.67	1.01	1.0201
Б	251 (1.67%)	14185 (1.54%)	0.92	-0.13	0.0169
З	253 (1.69%)	16917 (1.83%)	1.08	0.14	0.0196
Я	268 (1.79%)	18495 (2.00%)	1.12	0.21	0.0441
Ч	277 (1.85%)	13109 (1.42%)	0.77	-0.43	0.1849
Ь	278 (1.85%)	39478 (4.28%)	2.31	2.43	5.9049
Ы	288 (1.92%)	22378 (2.43%)	1.27	0.51	0.2601
У	328 (2.19%)	19571 (2.12%)	0.97	-0.07	0.0049
П	413 (2.75%)	30957 (3.36%)	1.22	0.61	0.3721
Д	430 (2.87%)	20738 (2.25%)	0.78	-0.62	0.3844
М	471 (3.14%)	18979 (2.06%)	0.66	-1.08	1.1664
Л	546 (3.64%)	35472 (3.85%)	1.06	0.21	0.0441
К	606 (4.04%)	36961 (4.01%)	0.99	-0.03	0.0009
В	668 (4.45%)	38616 (4.19%)	0.94	-0.26	0.0676
Р	812 (5.41%)	54909 (5.96%)	1.10	0.55	0.3025
С	828 (5.52%)	51572 (5.60%)	1.01	0.08	0.0064
Н	943 (6.28%)	54568 (5.92%)	0.94	-0.36	0.1296
Т	1009 (6.72%)	73112 (7.94%)	1.18	1.22	1.4884
А	1085 (7.23%)	78600 (8.54%)	1.18	1.31	1.7161
И	1170 (7.80%)	67435 (7.32%)	0.94	-0.48	0.2304
Е	1193 (7.95%)	60447 (6.56%)	0.83	-1.39	1.9321
О	1741 (11.60%)	81716 (8.87%)	0.76	2.73	7.4529

Сравните итоговый порядок букв по возрастанию распространенности
ЪЭЮЩФХЦШЖГЧБЗЯМУДЫЙПЛКВЬСНРЕИТАО - в словаре,
ЪЭЦШФЩЮЖХГЙБЗЯЧЬЫУПДМЛКВРСНТАИЕО - в тексте.

Обращает на себя внимание, что частоты букв с более низкими частотами отличаются на доли процента, а частоты часто употребительных букв могут отличаться на несколько процентов. Квадрат отклонения частоты только одной наиболее упоьребительной буквы "о" превысил сумму квадратов пятнадцати первых по редкости букв. Это означает, что евклидово расстояние на самом деле чувствительно только к наиболее распространенным буквам. С другой стороны, отношение частот буквы в файлах отличается от единицы на 20% для самой редкой буквы и на 24% для самой частой, то есть процент отклонения практически не кореллирует с частотой вхождения буквы.

В таких ситуациях предпочтительно использование логарифмической шкалы. Поскольку до 20% колебаний связано с тематикой и видом файла на русском языке и проявляется даже в файлах, размер которых более 1Mb, то значения частот нет сысла запоминать с относительной точностью более 10-15%. Близость при достаточно большом объеме статистики также правильнее оценивать не по абсолютной близости частот, а по близости к единице их отношений или, что эквивалентно, по близости в логарифмической сетке координат. Если забыть о малых значениях, то представляется разумным вместо количества N_x вхождений заданной буквы $x \in X$ в текст использовать в рассчетах значение, близкое к $\log_{1.1} N_x$. Однако наша цель состоит в построении алгоритма, работающего в том числе с короткими текстами, а в коротких текстах неизбежно отсутствуют нескольких букв алфавита, то есть несколько координат оказываются равными минус бесконечности и следовательно расстояние, посчитанное по предложенной формуле, всегда оказывается бесконечным.

Причина неудачи в том, что выражение $\log_{1.1} N_x$ лишь при $N_x > 0$ адекватно отражает логарифм математического ожидания количества вхождений буквы в текст заданной длины с заданной кодировкой. При $N_x = 0$ значение последнего отрицательно и интуитивно очевидно, что вычисленная по наблюдавшимся данным оценка для коротких текстов должна быть невелика по модулю (например, отсутствие буквы в слове не столь показательно, как два раза всретившаяся буква), а для очень длинных текстов - возможно иметь больший модуль (если для больших текстов считать только первые буквы слов, то слово, начинающееся на "ы" должно вызывать бошие подозрения в неправильности кодировки, чем изменение на 20% относительных частот нескольких других букв). Из каких же объективных соображений можно определить эту оценку?

Для определения указанной оценки необходимо разумное априорное ограничение на функцию распределения следующей случйной величины: логарифм математического ожидания частоты случайно (по равномерному распределению) выбранной буквы. Поскольку об этой величине ничего заранее не известно, то предположим, что она симметрична (более естественного предположения на ум не приходит). Пусть у нас не встретилось множество X^0 букв мощности N_x^0 и $N_x \geq 2N_x^0$. Тогда обозначим X^2 множество из N_x^0 наиболее часто встретившихся букв и X^1 - множество остальных букв. Тогда средние значения логарифмов математических ожиданий частот букв связаны очевидным равенством $S(X^1) = \frac{S(X^0)+S(X^2)}{2}$, откуда $S(X^0) = 2S(X^1) - S(X^2)$ получается естественной заменой для $\log_{1.1} 0$. При $N_x = 2N_x^0$ имеем $S(X^0) = -S(X^1)$. В случае $N_x < 2N_x^0$ это равенство получается из условия симметричности в качестве верхней оценки, но оценки снизу условие симметричности не дает, поэтому имеет прямой смысл заменить $-\infty$ на эту верхнюю оценку.

Итак мы получаем математическую модель, в которой каждый документ представляется точкой многомерного пространства, лежащей на пересечении единичного куба с гиперплоскостью, проходящей через вершины, лежащие на осях. Чтобы было проще вычислять близость документа к эталонной статистике, в знаменателе дроби для $\hat{f}_x$ будем писать вместо суммы корень из суммы квадратов. При этом каждый документ отобразится точкой в многомерном пространстве, лежащей уже не на плоском сечении положительного октанта (сумма частот равна единице!), а на части единичной сферы, содержащейся в этом октанте. Последнее позволяет заменить оцеку сверху расстояния между точками оценкой снизу скалярного произведения радиус-вкторов этих точек, вычислять которое значительно быстрее. Здесь для нашей задачи важен вывод о том, что естественно сначала применить функцию преобразования частот, а уже затем определять близость, вычисляя не что-то сложное, а просто линейную комбинацию с посчитанными коэффициентами. Это особенно важно потому, что вычисление близости занимает бо́льшую часть времени работы алгоритма. Константа k в определении функции преобразования частот $\tilde{f}(x)$ должна быть определена эмпирически.

Еще раз обратим внимание на то, что частоты букв могут заметно отличаться в различного рода текстах на одном и том же языке. Например, частота мягкого знака изменилась более чем в два раза. Здесь кроется ахиллесова пята идеи: объем информации, которую можно использовать из статистики частот букв, как бы тщательно она ни составлялась, весьма ограничен. Построить фразу, на которой алгоритм дал бы сбой было очень трудно, но в конце концов это удалось. Фраза в koi8 "Я, ШЕЙХ, ЦЕНЮ НЮХ!" содержит только буквы, обычно чаще употребляемые в cp1251 ("с, ыеки, геоа оаи"), и кодировка файла с единственной этой фразой опознается программой "FAR" версии 1.51 как cp1251.

Статистика частот букв в "FAR" содержит не более 255 байт информации о структуре текстовых файлов. Значительная часть этой информации относится к соответствию букв в данной кодировке и к другим символам. Если запоминать корень из частоты буквы, то память, нужная для запоминания ее с разумной точностью, не превышает 1 байт, в целом получаем 32 байта языковой информации.

Учитывая разнообразие текстовых файлов (и особенно коротких запросов по www), для решения рассмотренной задачи следовало бы использховать намного больше информации о том, чем собственно русскоязычный текстовый файл отличается от остальных файлов. Здесь слова "русскоязычный файл" понимаются в следующем смысле: файл может включать кроме русских слов ASCII знаки и слова из латинских букв, русские буквы "р", "с" и некоторые другие могут подменяться близкими по начертанию латинскими, файл может содержать неологизмы и редко используемые слова и сокращения. Можно пойти дальше и включить сюда украинский, белорусский, болгарский и другие родственные языки, поскольку русский человек, не знакомый с этими языками, с первого взгляда отличит текст на любом из этих языков от текста в неправильной кодировке. Надежнее, однако, различать эти языки и держать отдельно информацию о каждом из них.

Для того, чтобы даже при многократном повышении надежности по сравнению с алгоритмом Е. Рошаля, заложенным в FAR, одновременно увеличить в десятки раз количество допустимых кодировок, необходимо выделить и использовать хотя бы раз в сто больше информации о структуре языка, причем не просто словарей, а именно информации общего характера, не препятствующей использованию неологизмов.

Простой способ расширения на порядок объема используемой информации - это запоминание относительных частот букв, стоящих на заданной позиции в слове (например: первой, второй, третьей, последнеей, предпоследней, предпредпоследней, ...). Если бы, например, "FAR" знал, что буква "ы" никогда не бывает в русском слове первой, то учет этой информации вероятно помог бы ему правильно опознать фразу о шейхе.

Более радикальный способ расширения информации - это использование относительных частоты сочетаний пар букв. Если даже не учитывать различий между прописными и строчными буквами, то возможных сочетаний пар $33 * 33 = 1089$, что в 33 раза больше, чем самих букв. Алгоритм, действующий по этому признаку, уже реализован и хотя на каждую пару букв в нем задействовано всего 2 бита информации, он как правило выдает правильную кодировку уже по одному достаточно длинному (более 5 букв) слову. Однако, к сожалению, на фразе про Шейха и он дает сбой.

Реализованный алгоритм можно значительно усилить, если запоминать на каждое сочетание хотя бы полбайта информации и при этом различать сочетания в начале слова, в самом конце, за одну букву до конца, за две буквы до конца и в оставшихся позициях, причем существенная информация содержится не только в частотах смежных пар букв, но и в частотах пар букв через одну. В итоге при использовании статистики перечисленных сочетаний пар букв, получается 8*1089=8732 случайных величин, слабо связанных взаимными зависимостями. Это в более чем 200 раз превышает заложенный Е. Рошалем объем информации, что казалось бы должно бы было позволить при более чем десятикратном увеличении количества допустимых кодировок резко снизить требования к длине файла. На примере фразы о шейхе видно, что автоматически этого не происходит. В чем же причина? Почему, к примеру, слово "ЛЮБЛЮ" в KOI-8 получается менее русским, нежели "мавма" в CP1251?

Внимательное рассмотрение показывает, что дело не в ошибках рассчетов и не в недостаточности собираемой информации. Оказалось, что, как это ни странно, порочна именно идея определять близость к русскому тексту непосредственным сопоставлением частот сочетаний пар букв, встретившихся в тексте с типичными частотами сочетаний. На математическом языке это означает, что евклидовы метрики на пространстве матриц частот сочетаний букв и даже на проствнстве матриц корней из частот сочетаний букв не адекватны рассматриваемой задаче. Постараюсь это объяснить на примере слов "ЛЮБЛЮ"-"мавма".

В следующей таблице отражены частоты по статистике словаря, в ней $x, y \in X$ - буквы, N_x, N_y - количество их в словаре, N_{xy} - количество сочетаний XY в словаре, N_{xy}/N_x - вероятность сочетания при условии наличия первой буквы, $\frac{N_{xy}}{N_x * N_y}$ - отношение N_{xy} к аналогичной величине, но вычисленной для случайного файла той же длины и с тем же количеством букв X и Y.

xy	N_x	N_y	N_{xy}	$\frac{N_{xy}}{N_x}$	$\frac{N_{xy}}{(N_x N_y)}$	$N_y x$	$N_{xy}/N_y x$	$\frac{N_{xy}}{\sqrt{N_x N_y}}$
лю	35472	2208	89	0.00250	0.0000011322	24	3.708333	0.032
юб	2208	14185	46	0.02083	0.0000014685	19	2.421052	0.008
бл	14185	35472	91	0.00641	0.0000001807	27	3.370370	0.005
лю	35472	2208	89	0.00250	0.0000011322	24	3.708333	0.032
ма	18979	78600	208	0.01096	0.0000001394	146	1.424667	0.005
ав	78600	38616	174	0.00221	0.0000000572	333	0.522533	0.003
вм	38616	18979	30	0.00077	0.0000000406	13	2.307692	0.001
ма	18979	78600	208	0.01096	0.0000001394	146	1.424667	0.005

Из нее видно, что даже если просто заменить частоты сочетаний условными вероятностями второй буквы при условии на первую букву сочетания, то "ЛЮБЛЮ" предстает уж никак не менее типичным для русского языка словом чем "мавма"; При перестановке букв сочетания из слова "ЛЮБЛЮ" становятся намного менее русскими, а в слове "мавма" - ненамного, а сочетание "ва" даже в два раза более распространено в русском языке, чем "ав".

Теперь ясно, что в таблице частот сочетаний букв полезная информация содержится, но просто она оказалась погребенной под толстым слоем информации о частотах отдельных букв, которая как уже упоминалось не стоит и ста байт. Чтобы вытащить ее на свет божий, нужно придать сочетаниям редких букв больший приоритет чем сочетаниям частых. Еще правильнее было бы отделить информацию о частотах отдельных букв и уточнить веса для этой и для оставшейся информации. Вопрос о том, как же именно убрать из информации о частотах сочетаний пар букв информацию о частотах букв, является далеко непростой. Уж, казалось бы, чем плоха формула $S_{xy} = \frac{N_{xy}}{N_x N_y}$, если уж для равномерно распределенных по большому файлу размером N байт N_x букв $x \in X$ и N_y букв $y \in X$ именно эта формула дает правильный результат?

Рассмотрим пример файла, содержащего повторения двух слов: "ЭЭ" и "ММ", причем "ЭЭ" употребляется в два раза чаще, чем "ММ". Тогда $S(\text{Э}, \text{Э}) = N_{\text{Э}}\text{Э}/(N_{\text{Э}} * N_{\text{Э}})$ становится в два раза меньше, чем $S(\text{М}, \text{М}) = N_{\text{М}}\text{М}/(N_{\text{М}} * N_{\text{М}})$, то есть S_{xy} становится обратно пропорциональным $N(X, Y)$, что уже совсем не дело. Вообще, если матрица диагональна, то $S(\text{Э}, \text{Э}) = N_{\text{Э}}\text{Э}/(N_{\text{Э}})$ превращает ее в единичную матрицу. Эта формула может быть также записана в виде $S_{xy} = N_{xy}/((N_X * N_Y)^0.5)$.

Случаи равномерного распределения и диагональной матрицы являются противоположными крайними случаями: в одном из них частоты сочетаний распределены между парами совершенно равномерно, а в другом - крайне неравномерно. Нечто промежуточное хочется иметь для произвольных матриц с неотрицательными членами. Если бы можно было подобрать l_X и r_Y чтобы матрица S_{xy}, определенная формулой $S_{xy} = N_{xy}/(l_X * r_Y)$, была двоякостохастической, то есть такой, что сумма всех элементов любой строки равна единице и также сумма всех элементов любого столбца равна единице, то это бы уже как раз и означало, что информация о частотах букв уже не вошла в эту матрицу, а вошла только в l_X и r_Y. К сожалению, это не всегда возможно: если файл содержит повторения трех слов "АХ", "ХА", "АА" в равном количестве, то $S(X,X)=0$ и поэтому из двоякостохастичности $S(A,X)=1-S(X,X)=1$, $S(X,A)=1-S(X,X)=1$, $S(A,A)=1-S(A,X)=0$, что противоречит определению.

Таким образом, полностью отделить частоты сочетаний от частот букв невозможно. Можно попытаться сделать это ровно в той степени, в какой это все-таки возможно. В настоящий момент не видно лучшего варианта кроме как найти и использовать хотя бы одно решение $(l_x, r_x), x \in X$ экстремальной задачи

$$\sum_{x\in X}\left(\sum_{y\in X}\frac{N_{xy}}{l_x r_y}-1\right)^2+\sum_{y\in X}\left(\sum_{x\in X}\frac{N_{xy}}{l_x r_y}-1\right)^2 \Rightarrow \min$$

$$\left(\sum_{y\in X} N_{xy}\right)^2 \leqslant l_x^2 \leqslant \sum_{y\in X} N_{xy}, \qquad x\in X$$

$$\left(\sum_{x\in X} N_{xy}\right)^2 \leqslant r_y^2 \leqslant \sum_{x\in X} N_{xy}, \qquad y\in X.$$

Благо, нахождение решения этой задачи известными методами не составляет принципиальной трудности. В результате матрица с элементами

$$S_{xy}=\sqrt{\frac{N_{xy}}{l_x r_y \sum_{\xi,\eta\in X}\frac{N_{\xi,\eta}}{l_\xi r_\eta}}}$$

представит собой разумное выделение информации, не имеющей отношения к частотам отдельных букв.

Другой подход, работающий в этом конкретном случае - это анализ частот пар букв, находящихся на заданной позиции в слове. Например, в слове "мавма" совершенно нетипичным для русского языка являюися сочетания первой "м" с третьей "в" и сочетания предпредпоследней "в" с предпоследней "м".

Возникает вопрос, как объединять информацию, основанную на анализе частот сочетаний первых двух букв с информацией, основанной на анализе частот самих букв в этих позициях, анализе последних букв и т.д. Точный ответ на этот вопрос зависит от специфики языка и распространенных кодовых таблиц. Идея состоит в том, что раз уж в пространстве возможных файлов есть множество R русских текстов в правильной кодировке и семейство эквивалентных ему множеств W_c русских текстов в фиксированной неправильной кодировке, $c \in \{$неправильные кодировки$\}$, и при этом есть различные признаки (чистые и смешанные) правильности кодировки, каждый из которых позволяет определить не слишком большое содержащее R множество, то пересечение этих множеств должно выделить искомую достаточно тесную окрестность множества R.

Таким образом, мы приходим к следующим выводам:

(1) Эффективное определение правильности кодировки может быть выполнено на основе анализа соответствия относительных частот вхождений букв и относительных частот двухбуквенных сочетаний в проходящем файле и в заранее обработанном образце.
(2) Ранее практически реализованные на этой основе Е. Рошалем и автором алгоритмы могут неправильно оценить кодировку фразы, состоящей из относительно редких русских букв. Для исключения этих ошибок следует использовать информацию о сочетаниях букв на заданных позициях и отделить ее от резко доминирующей над ней информации о частотах отдельных букв.

(3) Алгоритм может быть практически реализован на сравнении значения линейной комбинации (с положительными коэффициентами) предварительно преобразованных статистик вхождений букв и сочетаний с максимально возможным из ее значений для файла заданной длины. Имеет смысл проводить независимый подсчет для каждой группы статистик и контроллировать правильность кодировки независимо по всем статистикам.
(4) Оптимизирующая коррекция выбора преобразований статистик, уточнение различных накапливаемых статистик условного образцового большого текста, пересчет (с учетом добавления языков и кодировок) зависящих от длины файла и основанных на анализе коротких текстах предельно допустимых значений различных типов статистик, а также накопление образцов текстов, особенно трудно поддающихся распознаванию кодировки, – все это должно стать частью обучения алгоритма.

§4. Файлы сопровождения, их структура, работа и технология обучения алгоритма.

4.1. Общая структура системы. Решение поставленной здачи осуществляется в чередовании или комбинации двух режимов работы:

(1) 1) режим распознавания кодировки проходящего файла, в котором используется быстрый и экономный по времени и памяти алгоритм и
(2) 2) режим обучения новым кодировкам, языкам и их новым проявлениям, в ходе которого указанный алгоритм уточняется и модифицируется.

Если первый режим предполагает минимальное использование машинных ресурсов, то второй, наоборот, предполагает значительную статическую память о накапливаемом опыте, значительную оперативную память для переосмысления алгоритма с учетом добавляемой информации и ощутимые затраты времени на оптимизацию алгоритма.

Представляется вполне естественной реализация решения в виде следующей системы файлов:

(1) небольшой исполняемый файл encorrec или encorrec.com (от слов Encoding Correcting Recognizer), исправляющий кодировку, работая в режиме фильтра и требуя минимального размера оперативной памяти, считывая алгоритм распознавания кодировки из специального конфигурационного файла;
(2) редактируемый конфигурационный файл encorrec.cnf, начало которого содержит компактное описание алгоритма для исполняемого файла и считывается им при запуске либо команды для обучения, а далее содержится в читаемом виде вся предыстория обучения;
(3) исполняемый файл corrtuto, обрабатывающий команды и историю из конфигурационного файла и оптимизирующий алгоритм распознавания кодировки. При этом могут одновременно учитываться большой текстовый файл для уточнения информации о языке и текстовый файл с примерами строк, apriore трудно поддающихся определению кодировки. Последний может создаваться автоматически в ходе работы encorrec.

Чтобы четко охарактеризовать работу исполняемых файлов, проще всего, повидимому, подробно описать формат конфигурационного файла, сопровождая это описание необходимыми пояснениями. Как уже было упомянуто, конфигурационный файл состоит из двух частей. Первая часть содержит либо алгоритм работы correnc, запись которого использует только ASCII символы из первой половины таблицы, либо команды для corrtuto. Рассмотрим каждый из этих вариантов подробнее.

4.2. Описание алгоритма работы..
Алгоритм считывается из текстового файла. Символ # и последующая за ним часть строки не воспринимаются. Первая строка файла идентифицирует

тип файла и имеет вид

[File: EnCorRec.cnf]

Далее следуют разделы содержимого. Каждый раздел начинается с новой строки заголовком в квадратных скобках. Разделы с неизвестными текущей версии заголовками или в неподходящем месте файла игнорируются. В начале файла содержится информация о текстовых символах, затем перечисляются алгоритмы распознавания, затем перечисляются в порядке убывания распространенности кодировки и после заголовка [Notes] следует бесполезная для определения кодировки информация, важная для обучения новым языкам и трудным для распознавания текстам. Если обучение не планируется и место на диске дефицитно, то эта информация может быть отброшена.

[UpperCase]

В разделе по порядку перечисляются все N заглавных букв, информация о которых учитывается алгоритмом. Шестнадцатиричные коды этих букв в таблице UNICODE перечисляются через пробел, запятую или запятую с пробелом или концом строки. Перечисление завершается точкой с запятой. Последовательность присутствующих в любой кодовой таблице 32 русских букв в алфавитном порядке может быть заменена словом Russian32.

[LowerCase]

В разделе по порядку перечисляются все строчные буквы, информация о которых учитывается алгоритмом в том же виде и в том же порядке как заглавные.

[NoCase]

В том же формате перечислены буквы неопределенного регистра, такие как I в кавказских языках.

[TheSame]

Через пробел, запятую или запятую с пробелом перечиляются шестнадцатиричные коды букв в таблице UNICODE и других символов, имеющих тождественное очертание как, например, латинская буква "c" имеет очертание, сходное с русской буквой "с".

[Algorithm]

Этот высшего ранга заголовок объединяет ряд настроек и разделов с настройками алгоритма распознавания вплоть до следующего [Algorithm] или [CodeTables]. Часть настроек устанавливает переменные, которые могут быть изменены из командной строки:

PseudoGraphics=On/Off

Следует ли учитывать возможность использования в тексте псевдографики и спецсимводов редакторов? Для форм, представляемых по www, очевидно, Off, а для e-mail столь же очевидно On.

SeveralEncodings=On/Off

Следует ли предполагать возможность представления различных абзацев текста в разных кодировках?

OpenMime=On/Off

Следует ли вскрывать Mime-конверты с текстом, доставая его из base64, uuencode и Qouted-Printable? (для последующих версий)

Unicode=On/Off

Следует ли учитывать возможность представления текста в Unicode?

Остальные настройки из [Algorithm] описывают результаты обучения и распределены по следующим разделам:

[.X]

Не менее $N+2$ двузначных шестнадцатиричных чисел, характеризующих информацию о частотах первых букв слов и ее приоритет.

Аналогично, разделы [.?X], [.??X], [.???X], [X.], [X?.], [X??.], [X???.], [X????.] содержат не менее N^2+2 двузначных шестнадцатиричных чисел каждый, характеризующих информацию о частотах соответственно вторых, третьих, четвертых, последних, предпоследних, третьих, четвертых и пятых от конца букв слов и ее приоритет при применении алгоритма.

Аналогично, разделы [.XY], [.?XY], [.??XY], [XY.], [XY?.], [XY??.], [XY???.], [.X?Y], [.?X?Y], [X?Y.], [X?Y?.], [X?Y??.] содержат не менее N^2+2 двузначных шестнадцатиричных чисел каждый, характеризующих информацию о частотах сочетаний пар букв слов в соответствующих позициях и их приоритет при применении алгоритма.

[CodeTables]

Далее следуют разделы [cp866], [alt], [koi8-r] и т. д., содержащие описания кодировок.