Правительство Российской Федерации

Государственное образовательное бюджетное учреждение

высшего профессионального образования

«Национальный исследовательский университет –

Высшая школа экономики»

Факультет **Гуманитарных наук**

Кафедра **Фундаментальной и компьютерной лингвистики**

**КУРСОВАЯ РАБОТА**

На тему: Формат KWIC для параллельных корпусов:   
пакет search\_kwic для Python.

Студентка группы **№ 152**

Терехина Мария Дмитриевна

Научный руководитель

Старший преподаватель факультета

гуманитарных наук Мороз Г.А.

Москва 2018

# Оглавление

[Оглавление 2](#_gjdgxs)

[1.](#_30j0zll) Введение 3

[2.](#_1fob9te) Разработка 4

[2.1. Встроенный переводчик 4](#_3znysh7)

[2.2. Автоматическое выравнивание текстов 6](#_2et92p0)

[2.1.1 Введение 6](#_tyjcwt)

[2.2.2 Модель IBM-M2 7](#_3dy6vkm)

[2.2.3. Использование программы Fast Align 13](#_1t3h5sf)

[2.3. Программа на основе моделей универсальных зависимостей 14](#_4d34og8)

[2.3.1. Введение 14](#_2s8eyo1)

[2.3.2. Алгоритм 15](#_3rdcrjn)

[3.](#_26in1rg) Литература 16

# Введение

KWIC (Key Word in Context) – это формат представления конкордансов (списка контекстов искомого слова), при котором ключевое слово располагается по центру, слева и справа от него – контекст, помещающийся в строку. Формат KWIC используется многими корпусами, однако ни один из известных нам параллельных корпусов не поддерживает данный формат в полной мере. Например, в параллельном подкорпусе Национального корпуса русского языка в формате KWIC отображаются только контексты на языке запроса[[1]](#footnote-1). Это связано с тем, что нахождение в параллельном тексте фрагмента, соответствующего тексту запроса (далее этот фрагмент текста будет именоваться *переводом*) – достаточно нетривиальная задача.

В 2016 году в Школе Лингвистики Факультета гуманитарных наук НИУ ВШЭ стартовал проект lingcorpora, цель которого – создание пакета для языков программирования Python и R, содержащего API для корпусов текстов. На данный момент пакет для Python включает корпуса следующий языков: \_, для R: \_. Пакет находится в открытом доступе. Для Python его можно скачать через pip, для R пакет находится на стадии разработки.

Поскольку lingcorpora включает API для параллельных корпусов, появилась необходимость в формате KWIC и для них, для унификации выдачи. Предварительный поиск литературы показал, что публикаций или проектов на эту тему нет. В связи с этим разработка алгоритма приведения текста к формату KWIC производилась с нуля. Подробно этапы разработки алгоритма будут описаны в разделе 2.

Поскольку квикизация параллельного текста – не обязательное условие работы параллельных корпусов в lingcorpora, было решено вынести этот функционал в отдельный пакет search\_kwic, также находящийся в открытом доступе с возможностью установки через pip.

# Разработка

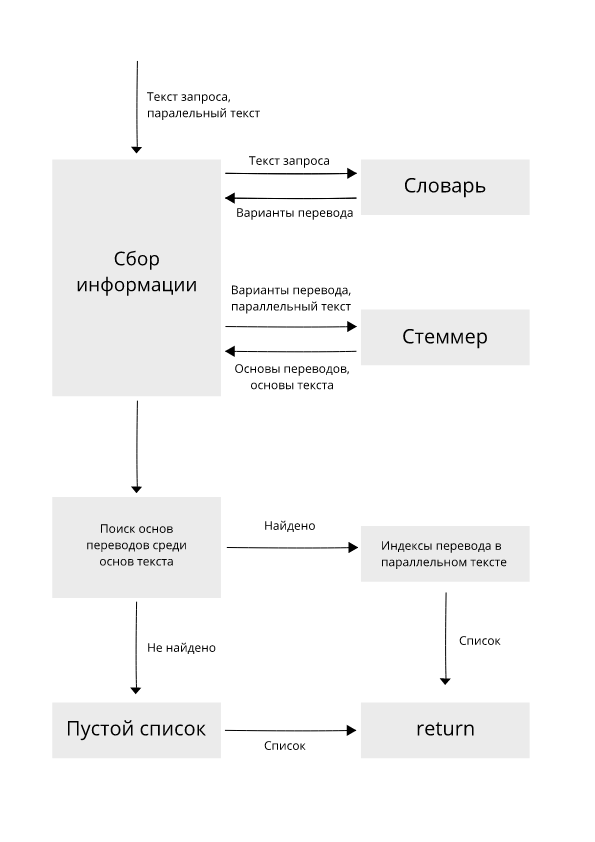
Алгоритм представления текста в формате KWIC подразумевает нахождение в параллельном тексте перевода запроса. Архитектура lingcorpora требует также нахождения индексов перевода (т.е. индексов начала и конца перевода в строке параллельного текста), что не составляет труда после того, как сам перевод найден.

В ходе разработки алгоритма было рассмотрено три принципиальных подхода к решению этой задачи: встроенный переводчик, автоматическое выравнивание текстов, программа на основе моделей универсальных зависимостей. Их устройство и причины отказа от первых двух из них будут описаны в этом разделе.

2.1. Встроенный переводчик

Данный алгоритм предельно прост и устроен следующим образом:

1. Программа получает на вход слово, перевод которого нужно найти в параллельном тексте и сам параллельный текст.
2. Происходит обращение к онлайн-словарю, в качестве ответа возвращается список возможных переводов данного слова.
3. Переводы передаются в стеммер, возвращаются основы переводов. Весь параллельный текст также передается в стеммер.
4. В списке основ параллельного текста производится поиск основ переводов.
5. Если основа какого-либо из возможных переводов найдена среди основ параллельного текста, слово, чья основа совпала с основой одного из возможных переводов, объявляется переводом, возвращаются его индексы (индексы начала и конца) в параллельном тексте. Если не найдена, возвращается пустой список (см. Рисунок 1).

Рисунок 1. Алгоритм работы программы со встроенным переводчиком

В качестве онлайн-словаря был выбран сервис Яндекс.Словари (<https://tech.yandex.ru/dictionary/>). Для этого было несколько причин, в том числе бесплатный доступ к API (с ограничением в 10 000 обращений в сутки) и простота использования. Сервис работает с get- и post-запросами, возвращает автоматически сгенерированную словарную статью в формате JSON, JSONP или XML (более подробно см. <https://tech.yandex.ru/dictionary/doc/dg/reference/lookup-docpage/>). Дальше страница обрабатывается xml-парсером, достаются все варианты перевода.

В качестве стеммера использовался nltk.stem.snowball (Bird, Loper 2004). Это алгоритм, который следуя ряду правил последовательно отсекает от основы аффиксы, принимая во внимание особенности языка. В данном пакете есть стеммеры для 15 языков, включая русский, однако нет стеммера, например, для польского, что вынуждает для некоторых языков искать другие алгоритмы выделения основы.

Отказаться от этого метода было решено по нескольким причинам: во-первых, в такой реализации алгоритм имел очень много внешних зависимостей. Пакет nltk часто вызывает проблемы при установке, поэтому иметь такую зависимость кажется нецелесообразным. Во-вторых, в nltk.stem.snowball сравнительно небольшое количество языков. Это значит, что при добавлении каждого нового языка приходилось бы дополнительно искать алгоритмы стемминга для этих языков. Очевидно, для очень многих языков такого функционала просто не существует. В-третьих, сервис Яндекс.Словари ограничивает число запросов в сутки 10 тысячами, что при интенсивном использовании достаточно малое количество, которое может быть исчерпано за несколько десятков запросов. И наконец, всем эти сложностям можно было бы найти решение, если бы алгоритм эффективно работал. Однако алгоритм давал непустой ответ лишь примерно в 25% случаев[[2]](#footnote-2).

2.2. Автоматическое выравнивание текстов

2.1.1. Введение

На втором этапе разработки алгоритма была рассмотрена возможность применения программы Fast Align (Dryer et al. 2013) для автоматического выравнивания текстов. Это алгоритм статистического машинного перевода, основанный на модели «шумного канала» или Noisy Channel model (Brill, Moore 2000). Модели такого типа применяются в следующей ситуации: нужно было написать какое-то слово *w*, состоящее из *k* букв. Однако получилось слово *n* из *l* букв (текст как бы прошел через «шумный канал» и исказился). Модель шумного канала будет решать задачу предсказания слова *w* по получившемуся слову *n*. Математически модель записывается следующим образом:

Т.е. предсказанное слово – это слово, вероятность которого быть исходным словом при отображении на письме n максимальна.

Применяя теорему Байеса получаем:

Последнее преобразование справедливо, поскольку знаменатель не зависит от w.

2.2.2. Модель IBM-M2

Наша модель утроена абсолютно аналогичным образом. Предположим, мы хотим перевести предложение с французского на английский. В таком случае мы будем искать решение следующего уравнения:

.

.

Плюс такой модели в том, что мы можем использовать множитель для увеличения вклада грамматики языка, на который осуществляется перевод. Проблема в том, что определить напрямую невозможно. Для решения этой задачи в модель вводятся дополнительные переменные выравнивания: , которые могут принимать значения , где m – количество слов в предложении (оригинале), – количество слов в предложении (переводе). Такая модель, которая включает дополнительные переменные выравнивания называется IBM model 2 (IBM-M2) (Collins 2011).

**Определение**. IBM-M2 model состоит из конечного множества английский слов *E*, множества французских слов , переменных *M* и *L* определяющих максимальную длину французских и английских предложений соответственно (Collins 2011: 6). Модель описывается следующими параметрами:

* для каждого . Параметр может быть интерпретирован как условная вероятность перевести французское слово как английское слово
* для каждого Параметр может быть интерпретирован как вероятность переменной выравнивания принять значение , учитывая длины и французского и английского предложений.

**Пример**. Чтобы пояснить все вышесказанное, рассмотрим два предложения, где

В таком случае переменные выравнивания задают следующее выравнивание:

*Le ⇒ the   
Programme ⇒ program   
 a ⇒ has   
 ete ⇒ been   
 mis ⇒ implemented   
 en ⇒ implemented   
application ⇒ implemented*

**Предположение о независимости.** Для работы IBM-M2 необходимо сделать два очень сильных предположения о независимости:

* зависит только от переменных *L* и *M* (и не зависит от слов в английском предложении и других переменных выравнивания).
* Значение зависит только от то есть только от английского слова, которому слово сопоставлено.

**Применение IBM-M2**. Рассматриваемая модель может быть применена на корпусе текстов, где нет приписанных переменных выравнивания в тренировочных данных.

Введем параметры (счетчики):

* – количество раз, когда слово *e* сопоставлено слову в тренировочных данных.
* – количество раз, когда слово e сопоставлено какому-либо слову в тренировочных данных.
* – количество раз, когда нам встретилось английское предложение длины и параллельное ему французское предложение длины , где слово французское слово сопоставлено английскому слову .
* – количество раз, когда нам встретилось английское предложение длины и параллельное ему французское предложение длины m.

Тогда согласно методу максимального правдоподобия[[3]](#footnote-3) наши параметры и (см. выше) будут равны:

Таким образом, наша модель принимает вид:

, где

То есть для каждого французского слова на позиции мы находим параллельное ему английское слово, расположенное на позиции , которую мы определяем, максимизируя произведение вероятности такого выравнивания и вероятности такого перевода

**Применение модели.** Алгоритм рассматривает все возможные пары слов языков и в корпусе, которые могут быть сопоставлены друг другу. Т.е. все возможные кортежи вида где — индекс параллельных друг другу фрагментов (~предложений) корпуса, — индекс слова в оригинальном тексте, — индекс слова в переводе.

Для каждой такой пары слов, если они сопоставлены друг другу, мы имеем . В таком случае мы инкрементируем счетчики . Если слова не сопоставлены, т.е. , счётчики не инкрементируются.

Рассмотрим сразу случай, когда в тренировочном корпусе нет переменных выравнивания , т.е. мы не можем напрямую вычислить все счетчики *c*, чтобы дать оценку параметрам *t* и *q*. В таком случае наш алгоритм будет работать итеративно, т.е. будет ЕМ-алгоритмом[[4]](#footnote-4). Для этого нам нужно:

* Изначально дать некоторую оценку счетчикам *.* Самый простой способ – инициализировать их рандомными числами.
* На каждой итерации алгоритма вычислять на основе наших данных и текущих значениях параметров *t*  и *q*. Затем пересчитывать параметры *t* и *q* на основе полученных счетчиков.

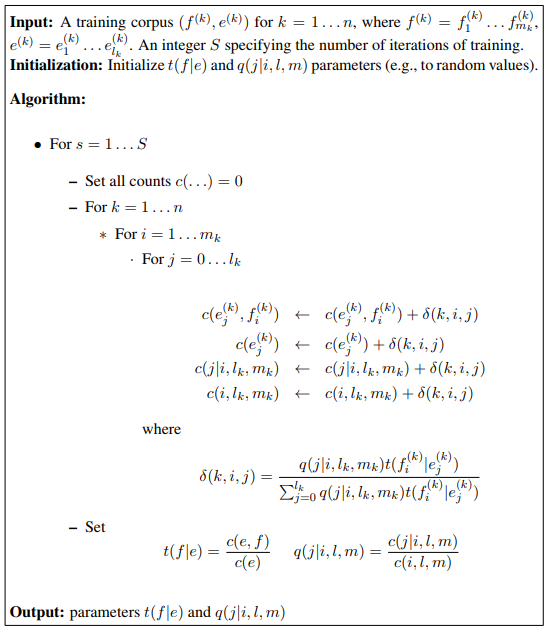
Если бы в тренировочных данных были переменные выравнивания, мы бы просто увеличивали значение счетчиков на 1, когда слова сопоставлены друг другу и не увеличивали бы в обратном случае:

В нашем же случае, поскольку переменных выравнивая нет, будем определять как

Это выражение по сути представляет собой вероятность того, что слово на позиции *j* будет сопоставлено слово на позиции *i*, учитывая параметры *p* и *q.* Для иллюстрации обратимся к примеру, рассмотренному выше:

Тогда значение будет текущей оценкой вероятности того, что слово сопоставлено слову и будет рассчитано как

Таким образом, оценка принимает во внимание и вероятность таких переменных выравнивания, и вероятность для слова *mis* быть переводом слова *implemented*. В знаменателе же находится сумма произведений этих вероятностей для всех возможных выравниваний в этом предложений (Collins 2011: 14). Полное описание алгоритма представлено на Рисунке 2.

Рисунок 2. Алгоритм работы IBM-M2 с ограниченными данными (Collins 2011:13)

2.2.3. Использование программы Fast Align

Как уже было сказано выше, нами была рассмотрена возможность использования программы Fast Align ((Dryer et al. 2013), являющаяся реализацией описанной выше модели IBM-M2, для нахождения перевода текста запроса. Однако для корректной работы программы нужен обучающий корпус объема, достаточного для оценки параметров алгоритма. В наших условиях обеспечить это невозможно, в силу, во-первых, отсутствия доступных для скачивания корпусов для некоторых из языков lingcorpora, во-вторых, Fast Align не предусматривает возможности сохранения обученной модели. То есть алгоритм пришлось бы обучать заново при каждом новом запуске программы. Если же использовать непредобученный алгоритм, добиться высокой точности работы невозможно, поскольку, как было описано выше, алгоритм инициализируется рандомными значениями параметров.

2.3. Программа на основе моделей универсальных зависимостей

2.3.1. Введение

Для итоговой версии программы было решено использовать модели универсальных синтаксических зависимостей (Universal Dependencies models) (Nivre et al. 2016). Universal Dependencies (Universal Dependencies 2014) – это проект, разрабатывающий универсальную кросс-лингвистическую аннотацию для синтаксической разметки с целью облегчения разработки многоязычного парсера, обучения на разных языках и типологически универсального синтаксического анализа.

Схема аннотаций основана на Stanford dependencies (de Marneffe et al., 2006, 2008, 2014), универсальных частеречных тегах Google (Lin et al., 2012) и морфосинтаксических тегах Interset interlingua (Zeman, 2008). Главная цель проекта – разработать универсальный перечень категорий, необходимый для аннотации сходных конструкций в разных языках, дополненный при необходимости лингвоспецифичными единицами. На данный момент разработано более 70 UD-моделей, больше 10 находятся в разработке. С полным списком моделей можно ознакомиться на сайте <http://universaldependencies.org/#language->.

Для Python и некоторых других языков программирования UD-модели доступны через UDPipe (Straka, Straková 2017) – проект чешского Institute of Formal and Applied Linguistics. UDPipe – это инструмент для автоматической обработки естественного языка, осуществляющий обработку текста на следующих уровнях:

* разделение на предложения
* токенизация
* приписывание частеречных тегов
* лемматизация
* синтаксическая разметка (в терминах UD-моделей)

Токенизация и разделение на предложения происходит при помощи однослойной двунаправленной рекуррентной нейронной сети GRU (Gated recurrent units), которая для каждого символа предсказывает, является ли он последним в токене, последим в предложении или не является последним ни там, ни там. Пробелы в токенах в конкретной модели допускаются, если в тренировочных данных был токен с пробелом (Straka, Straková 2017).

Частеречные теги генерируются тройками (UPOS, XPOS, FEATS), где UPOS – универсальный частеречный тег, XPOS – лингвоспецифичный частеречный тег, FEATS – список морфологических особенностей из унивесального инвенторя, на основе которых был сделан выбор в пользу выбранного частеречного тега. Частеречная разметка производится на основании четырех последних символов токена. Лемматизация происходит по автоматически сгенерированным на основе обучающей выборке правилам (Straka, Straková 2017).

Пакет ufal.udpipe доступен на сайте PyPI, возможна установка через pip. В репозитории проекта на сайте GitHub есть файл-оболочка для Python[[5]](#footnote-5), который был включен в состав пакета search\_kwic.

2.3.2. Алгоритм работы serch\_kwic

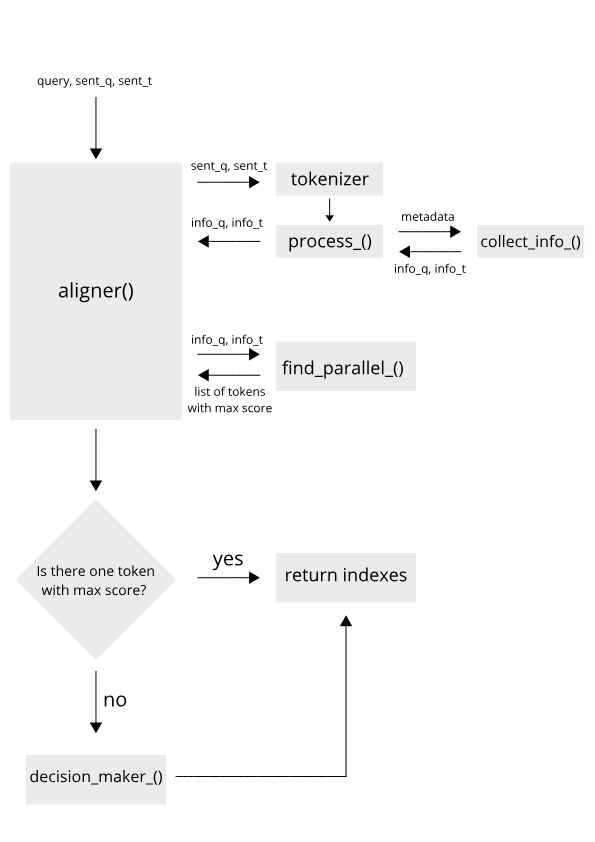
Пакет написан в парадигме объектно-ориентированного программирования. Файл search.py содержит класс *Aligner*, производящий поиск перевода. Класс инициализируется двумя аргументами: *queryLanguage* – язык текста оригинала, *targetLanguage* – язык параллельного текста. Языки указываются трехбуквенными сокращениями в формате ISO 639-3[[6]](#footnote-6). Далее происходит загрузка моделей: если модели для данных языков уже скачаны, они записываются в переменные *model\_ql* и *model\_tl*, если нет – происходит автоматическая загрузка из репозитория пакета на GitHub (<https://github.com/maria-terekhina/search_kwic>). Если моделей для данных языков нет, поднимается исключение . Все модели в формате zip-папки можно скачать, перейдя по ссылке (<https://lindat.mff.cuni.cz/repository/xmlui/handle/11234/1-2364>). Однако для автоматической загрузки такой формат неудобен, поэтому все модели были выгружены на GitHub в репозиторий пакета (Search KWIC 2018). Модели опубликованы под лицензией  [CC BY-NC-SA](http://creativecommons.org/licenses/by-nc-sa/4.0/) (Commons C.), допускающей свободное копирование и распространение материалов.

Запрос на нахождение параллельного слова выполняется через функцию *align()*. В функцию передаются следующие аргументы: *query* – текст запроса, *sent\_q* – предложение(я), содержащие текст запроса, *sent\_t* – предложение(я), содержащие параллельный текст. Если одна из моделей (или обе) не найдены, функция возвращает пустой список. Предложения токенизируется внутренними средствами моделей и передаются в функцию \_*process()*, где всем словам предложений приписываются частеречные и синтаксические теги, а также другая метаинформация[[7]](#footnote-7).

Полученные данные в виде строки передаются в функцию \_*collect\_info()*, где происходит парсинг этой строки. Порядковый номер каждого токена становится ключом в словаре. По каждому токену собирается информация о его значении (самом слове/знаке), частеречном и синтаксическом тегах, порядковом номере родителя и его синтаксическом теге, синтаксических тегах потомков (токенов, для которого данный токен является родителем), абсолютной позиции в предложении. Данная информация в виде словаря возвращается в функцию *align().*

Далее происходит вызов функции *\_ find\_parallel()* с параметрами query – текст запроса, *info\_q* и *info\_t* – словари с полученной метаинформацией обо всех токенов оригинального и параллельного текстов. В этой функции сначала среди ключей *info\_q* находится текст запроса, достается его метаинформация. Затем все токены из *info\_t* сравниваются с метаинформацией текста запроса по следующим параметрам: частеречный тег, синтаксический тег, синтаксический тег родителя, синтаксические теги потомков. За каждое совпадение токену из *info\_t* присваивается один балл (за теги потомков присваивается число баллов равное числу совпавших тегов). Вся информация передается обратно в *align().*

Если максимальный балл был присвоен только одному токену, функция *align()* возвращает список, содержащий индексы начала и конца этого слова в параллельном тексте. Если токенов с одинаковым максимальным баллом оказалось несколько, вызывается функция \_*decision\_maker()*, которая принимает решение, какой именно токен считать переводом: выбирается слово, номер абсолютной позиции которого в параллельном тексте меньше всего отличается от соответствующего номера позиции запроса в оригинальном тексте. Если таких слов оказалось два (одно расположено справа, а другое слева на одинаковом расстоянии), берется слово слева[[8]](#footnote-8).

Рисунок 3. Алгоритм работы класса Aligner

# Заключение

3.1. Структура пакета search\_kwic

Пакет search\_kwic состоит из двух файлов: search.py и model.py. Про второй из них, model.py, было уже написано выше, это файл файл-оболочка для работы с UD-моделями в Python. Файл search.py – основной файл пакета, содержащий алгоритм поиска перевода текста запроса (класс Aligner, принцип работы которого был описан в предыдущем разделе). Задача пакета – находить перевод слова и возвращать его индексы (начала и конца) в параллельном тексте.

Среди зависимостей, требующих дополнительной установки (не включенных в интерпретатор изначально) пакет search\_kwic имеет только пакет ufal.udpipe, свободно устанавливаемый через pip. Данная зависимость устанавливается автоматически при установке search\_kwic.

3.2. Оценка работы пакета

Оценка качества работы алгоритма происходила на примере трех языковых моделей: русском, английском и итальянском (поскольку автор работы владеет только этими языками). Были взяты 4 слова русского языка: *очки, третий, пошел, прекрасный* (слова разных частей речи). С каждым словом был сделан запрос в параллельный подкорпус НКРЯ в английский и итальянский подкорпуса при помощи пакета lingcorpora. Для каждого запроса были взяты первые 50 результатов. Итого было получено 400 пар параллельных фрагментов текста, 200 русско-итальянских, 200 русско-английских. Далее все пары были переданы в пакет search\_kwic. Для всех пар было замерено время работы алгоритма.

3.2.1. Время

Медианное время работы алгоритма (обработки одной пары оригинал-перевод), подсчитанное на описанных выше 400 парах – 0.38c (min = 0.09c, max = 3.84с, std = 0.44c).

3.2.2. Точность

Оценка точности производилась вручную. Если автор считал, что алгоритм правильно нашел перевод, паре присваивалось значение 1, если перевод был найден неверно – 0. Если автор считал, что найти правильный перевод в данном конкретном случае было невозможно, паре присваивалось значение 1. Найти перевод было невозможно, например, если предложения были неверно сопоставлены друг другу в корпусе и параллельный фрагмент не является переводом оригинала. Второй случай, при котором перевода фактически нет – если искомое слово входит в состав фразеологизма, непереводимого дословно[[9]](#footnote-9) или искомое слово просто опущено в параллельном тексте.

Далее была найдена сумма выставленного параметра по всем парам для каждого языка и поделена на количество пар. Для итальянского данное значение составило 0.67, для английского – 0.72.

# Литература

Bird, Loper 2004 — Bird S., Loper E. NLTK: the natural language toolkit //Proceedings of the ACL 2004 on Interactive poster and demonstration sessions. – Association for Computational Linguistics, 2004. – С. 31.

Brill, Moore 2000 — Brill E., Moore R. C. An improved error model for noisy channel spelling correction //Proceedings of the 38th Annual Meeting on Association for Computational Linguistics. – Association for Computational Linguistics, 2000. – С. 286-293.

Collins 2011 — Collins M. Statistical machine translation: IBM models 1 and 2 //Columbia Columbia Univ. – 2011.

Commons C. — Commons C. Attribution-NonCommercial-ShareAlike 4.0 International (CC BY-NC-SA 4.0).

De Marneffe et al. 2006 — De Marneffe M. C., MacCartney B., Manning C. D. Generating typed dependency parses from phrase structure parses //Proceedings of LREC. – 2006. – Т. 6. – №. 2006. – С. 449-454.

De Marneffe et al. 2008 — De Marneffe M. C., Manning C. D. Stanford typed dependencies manual. – Technical report, Stanford University, 2008. – С. 338-345.

De Marneffe et al. 2014 — De Marneffe, M. C., Dozat, T., Silveira, N., Haverinen, K., Ginter, F., Nivre, J., Manning, C. D. Universal Stanford dependencies: A cross-linguistic typology //LREC. – 2014. – Т. 14. – С. 4585-4592.

Dryer et al. 2013 — Dyer C., Chahuneau V., Smith N. A. A simple, fast, and effective reparameterization of ibm model 2. – Association for Computational Linguistics, 2013.

Lin et al. 2012 — Lin Y., Michel J. B., Aiden E. L., Orwant J., Brockman W., Petrov S. Syntactic annotations for the google books ngram corpus //Proceedings of the ACL 2012 system demonstrations. – Association for Computational Linguistics, 2012. – С. 169-174.

Nivre et al. 2016 — Nivre J., De Marneffe M. C., Ginter F., Goldberg Y., Hajič J., Manning C. D., McDonald R., Petrov S., Pyysalo S., Silveira N., Tsarfaty R., Zeman D. Universal Dependencies v1: A Multilingual Treebank Collection //LREC. – 2016.

Search KWIC 2018 — Search KWIC GitHub page, 2018. URL: <https://github.com/maria-terekhina/search_kwic> (Дата обращения: 16.05.2018).

Straka, Straková 2017 — Straka M., Straková J. Tokenizing, pos tagging, lemmatizing and parsing ud 2.0 with udpipe //Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies. – 2017. – С. 88-99.

Universal Dependencies 2014 — Universal Dependencies introduction// Filip Ginter GitHub page, 2014. URL: <https://fginter.github.io/docs/introduction.html> (Дата обращения: 16.05.2018).

Zeman 2008 — Zeman D. Reusable Tagset Conversion Using Tagset Drivers //LREC. – 2008. – Т. 2008. – С. 28-30.

1. Пример запроса в английский параллельный подкорпус НКРЯ, выдача в формате KWIC:  
   [http://search1.ruscorpora.ru/search.xml?mycorp=(lang%3A%22eng%22%20%7C%20lang\_trans%3A%22eng%22)&mysent=&mysize=24681277&mysentsize=1608376&dpp=&spp=&spd=&text=lexform&mode=para&sort=gr\_tagging&env=alpha&req=linguistics&out=kwic](http://search1.ruscorpora.ru/search.xml?mycorp=(lang%3A%22eng%22%20|%20lang_trans%3A%22eng%22)&mysent=&mysize=24681277&mysentsize=1608376&dpp=&spp=&spd=&text=lexform&mode=para&sort=gr_tagging&env=alpha&req=linguistics&out=kwic) [↑](#footnote-ref-1)
2. В момент разработки алгоритма не стояло задачи дать точную оценку эффективности его работы, поскольку было очевидно, что он не будет использоваться в качестве финальной версии. [↑](#footnote-ref-2)
3. Метод максимального правдоподобия (maximum likelihood method) [↑](#footnote-ref-3)
4. ЕМ-алгоритм [↑](#footnote-ref-4)
5. <https://github.com/ufal/udpipe/blob/master/bindings/python/examples/udpipe_model.py> [↑](#footnote-ref-5)
6. Со списком кодов можно ознакомиться по ссылке: <https://en.wikipedia.org/wiki/List_of_ISO_639-3_codes> [↑](#footnote-ref-6)
7. Список частеречных тегов доступен по ссылке: <https://goo.gl/GKEiQf>. Синтаксических — по ссылке: <https://goo.gl/qFetm8> [↑](#footnote-ref-7)
8. В данном случае нет принципиальной разницы, какое именно слово выбрать. Слово слева берется из-за особенности работы функции *min()*, которая при нахождении двух одинаковых минимальных значений возвращает первое из них, то есть в нашем случае вернет разницу в расстоянии от левого слова до текста запроса. Нужно понимать, что таких случаев, когда два равноудаленных слова имеют одинаковый балл, ничтожно мало. [↑](#footnote-ref-8)
9. Такое часто наблюдалось в запросах со словом *прекрасный*: слово входит в состав фразеологизма *в один прекрасный день*, чаще всего переводимого на английский как *one day*. Как видно, в таком случае слову *прекрасный* ничего не соответствует. [↑](#footnote-ref-9)