**МИНОБРНАУКИ РОССИИ**

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«РОССИЙСКИЙ ГОСУДАРСТВЕННЫЙ ГУМАНИТАРНЫЙ УНИВЕРСИТЕТ»**

**(РГГУ)**

ИНСТИТУТ ЛИНГВИСТИКИ

ФУНДАМЕНТАЛЬНАЯ И ПРИКЛАДНАЯ ЛИНГВИСТИКА

УНЦ лингвистической типологии

Епишев Егор Дмитриевич

**Тексты поп и рэп исполнителей: количественный анализ и автоматическое определение авторства**

Выпускная квалификационная работа студента 4-го курса

очной формы обучения

Направление 45.03.03

|  |  |
| --- | --- |
| **Допущен к защите на ГЭК**  Заведующий кафедрой  доктор филологических наук, профессор  Вера Исааковна Подлесская  «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_ 20….. г. | Научный руководитель  кандидат филологических наук,  доцент УНЦ лингвистической типологии  Мария Борисовна Коношенко  «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_ 20….. г. |

Москва 2020

# ОГЛАВЛЕНИЕ

[ОГЛАВЛЕНИЕ 2](#_Toc43376948)

[Введение 4](#_Toc43376949)

[Глава 1. Подсчёт расстояний между корпусами 6](#_Toc43376950)

[1.1 Корпус 6](#_Toc43376951)

[1.1.1 Сбор корпуса 6](#_Toc43376952)

[1.1.2 Подготовка корпуса 7](#_Toc43376953)

[1.2. Параметры сравнения текстов и подсчет расстояний 9](#_Toc43376954)

[1.2.1 Леммы 9](#_Toc43376955)

[1.2.2 N-граммы 10](#_Toc43376956)

[1.2.3 Скипграммы 10](#_Toc43376957)

[1.2.4 Подсчет расстояний (леммы, n-граммы, скипграммы) 11](#_Toc43376958)

[1.2.5 Ключевые слова 13](#_Toc43376959)

[Глава 2. Анализ результатов 15](#_Toc43376960)

[2.1 Сравнение рэп исполнителей 15](#_Toc43376961)

[2.1.1 Леммы 15](#_Toc43376962)

[2.1.2 Ключевые слова 17](#_Toc43376963)

[2.1.3 N-граммы 18](#_Toc43376964)

[2.1.4 Скипграммы 20](#_Toc43376965)

[2.1.5 Общий результат 20](#_Toc43376966)

[2.2 Сравнение поп и рэп стиля 21](#_Toc43376967)

[2.2.1 Леммы 22](#_Toc43376968)

[2.2.2 Ключевые слова 22](#_Toc43376969)

[2.2.3 N-граммы 23](#_Toc43376970)

[2.2.4 Скипграммы 23](#_Toc43376971)

[2.3 Выводы 24](#_Toc43376972)

[Глава 3. Автоматическое определение авторства 25](#_Toc43376973)

[3.1 Параметры 25](#_Toc43376974)

[3.2 Алгоритмы определения автора 25](#_Toc43376975)

[3.2.1 Первый вариант 25](#_Toc43376976)

[3.2.2 Второй вариант 27](#_Toc43376977)

[3.3 Результаты 28](#_Toc43376978)

[3.3.1 Внутренняя проверка 28](#_Toc43376979)

[3.3.2 Внешняя проверка 30](#_Toc43376980)

[Заключение 33](#_Toc43376981)

[Список литературы 35](#_Toc43376982)

[Интернет ресурсы 36](#_Toc43376983)

[Приложение 38](#_Toc43376984)

[Программы 38](#_Toc43376985)

[Таблицы 58](#_Toc43376986)

# Введение

Цель данной работы – количественный анализ текстов поп и рэп исполнителей на двух уровнях – полное сравнение текстов стилей и сравнение текстов каждого исполнителя отдельно, а также описание и анализ алгоритма автоматического определения авторства, который комбинирует в себе несколько методов решения данной задачи

Задача автоматического определения расстояния между текстами с последующей кластеризацией является одной из самых актуальных в компьютерной лингвистике. В частности, эта тема рассматривалась в работе [Балуева 2019] на материале поэзии. Главной целю было с помощью автоматических количественны методов определить схожесть текстов современного поэта и музыканта Эм Калинина и поэтов серебряного века. Основным инструментом, который использовался в данной работе, был подсчёт расстояний между текстами (с помощью частотных списков лемм, n-грамм, скипграм и ключевых слов). Он же применялись при подсчёте расстояния между корпусами и автоматического определения авторства в работах [Gomaa, Fahmy 2013, Кукушкина, Поликарпов, Хмелёв 2001] и доказали свою эффективность. В нашем исследовании использовались аналогичные параметры.

В качестве материала был собран корпус, содержащий тексты 40 исполнителей (20 рэп исполнителей, таких как Баста, Big Russian Boss, Лизер, Гуф и другие и 20 поп исполнителей соответственно, например, Басков, Лазарев, Пугачева, Киркоров и другие). Списки исполнителей были составлены на основе предложенных сервисом Яндекс.Музыка по заданным жанрам артистов.

Предполагалось, что расстояние между жанрами не будет сильно отличаться, однако для рэпа будет характерна матерная лексика, в то время как для поп стиля нет.

Результатом нашего исследования является описание полученных расстояний между текстами стилей, сравнение рэп исполнителей между собой, а также сравнение двух альтернативных методов определения авторства.

Работа состоит из 3 глав. Глава 1 посвящена описанию процесса сбора корпуса и обсуждению параметров, по которым проводилось сравнение. Вторая глава содержит в себе описание и анализ результатов сравнения рэп исполнителей между собой, а также сравнение рэп и поп стиля в целом. В третьей главе описываются два алгоритма определения авторства с полным сравнением эффективности каждого из них. В заключительной части подводятся итоги исследования.

# Глава 1. Подсчёт расстояний между корпусами

## 1.1 Корпус

### 1.1.1 Сбор корпуса

В качестве материала для исследования мы выбрали по 20 исполнителей для каждого из стилей из открытых списков исполнителей Яндекс Музыки. Исполнители выбирались исключительно по количеству альбомов – отбирались только те, у которых больше 5 альбомов. Тексты песен качались автоматически с сайта [Genius](https://genius.com/) с помощью написанной нами программы на языке Python и библиотек requests и BeautifulSoup, поскольку этот ресурс предоставляет возможность большой выкачки данных. Для каждого из исполнителей был создан отдельный файл, а также по одному большому файлу на каждый из стилей, в котором были собраны все тексты всех музыкантов данного стиля. Таким образом, была собрана база, которая состоит из двух общих файлов и 40 файлов для каждого из исполнителей. Общий объем корпуса составил 2321 песен и 464318 словоформ, что представлено в таблице 1.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Исполнитель | Стиль | Количество альбомов | Количество песен | Количество знаков |
| Басков | Поп | 11 | 111 | 102927 |
| Билан | Поп | 11 | 138 | 158458 |
| ВиаГра | Поп | 7 | 53 | 61307 |
| Витас | Поп | 18 | 154 | 134595 |
| Земфира | Поп | 12 | 97 | 69193 |
| Киркоров | Поп | 26 | 313 | 318435 |
| Лазарев | Поп | 9 | 101 | 133421 |
| Леонтьев | Поп | 24 | 218 | 242165 |
| Лолита | Поп | 6 | 13 | 13227 |
| Максим | Поп | 6 | 61 | 57237 |
| Меладзе | Поп | 9 | 73 | 68205 |
| Михайлов | Поп | 5 | 30 | 29739 |
| Натали | Поп | 10 | 96 | 68268 |
| Николаев | Поп | 17 | 120 | 118628 |
| Орбакайте | Поп | 11 | 110 | 86271 |
| Пугачева | Поп | 18 | 181 | 189159 |
| Руки Вверх | Поп | 8 | 23 | 27726 |
| Тату | Поп | 8 | 73 | 62204 |
| Чай Вдвоём | Поп | 8 | 77 | 89572 |
| Шура | Поп | 5 | 48 | 41857 |
| 2rbina2rista | Рэп | 7 | 38 | 82181 |
| Big Russian Boss | Рэп | 7 | 62 | 125075 |
| Face | Рэп | 12 | 68 | 105764 |
| Feduk | Рэп | 7 | 33 | 37870 |
| Kunteynir | Рэп | 9 | 70 | 120370 |
| Lsp | Рэп | 9 | 55 | 99073 |
| Morgenshtern | Рэп | 9 | 40 | 55456 |
| Noize MC | Рэп | 23 | 183 | 420330 |
| Баста | Рэп | 8 | 79 | 189872 |
| Гнойный | Рэп | 17 | 121 | 233884 |
| Гуф | Рэп | 5 | 68 | 153929 |
| Каста | Рэп | 10 | 80 | 181170 |
| Корж | Рэп | 7 | 61 | 93955 |
| Кровосток | Рэп | 9 | 73 | 124148 |
| Лизер | Рэп | 8 | 66 | 95185 |
| Скриптонит | Рэп | 7 | 61 | 134263 |
| Тимати | Рэп | 8 | 46 | 108603 |
| Хаски | Рэп | 7 | 34 | 72324 |
| Хлеб | Рэп | 11 | 49 | 64208 |
| Элджей | Рэп | 11 | 70 | 103071 |
| **Всего** | | **420** | **3447** | **4673325** |

Таблица 1. Общий объем корпуса

Стоить отметить, что написанная нами программа включает в себя функцию добавления текстов в корпус. Тем самым, в любой момент можно увеличить объемы обрабатываемой информации.

### 1.1.2 Подготовка корпуса

После того, как был собран корпус песен, из них были убраны все песни, в которых имелись английские слова, либо слова, написанные латиницей. Дальше мы приступили к его подготовке к подсчету расстояния.

Как уже говорилось во введении, для подсчёта расстояний использовались леммы, символьные n-граммы (от 2 до 5), скипграммы и ключевые слова. Символьные n-граммы – это последовательность символов длины n, включая пунктуацию, пробелы и переносы строки. Например, для *«казнить, нельзя помиловать»,* n-граммы длины 2 будут выглядеть как

('к', 'а'), ('а', 'з'), ('з', 'н'), ('н', 'и'), ('и', 'т'), ('т', 'ь'), ('ь', ','), (',', ' '), (' ', 'н'), ('н', 'е'), ('е', 'л'), ('л', 'ь'), ('ь', 'з'), ('з', 'я'), ('я', ' '), (' ', 'п'), ('п', 'о'), ('о', 'м'), ('м', 'и'), ('и', 'л'), ('л', 'о'), ('о', 'в'), ('в', 'а'), ('а', 'т'), ('т', 'ь')

Мы использовали именно символьные последовательности, поскольку они лучше подходят для корпусов меньшего объема. Во-первых, символьных последовательностей больше, чем последовательностей слов, а во-вторых, последовательности символов лучше описываю разные формы слова. То есть, например, последовательности, включающие «дома», «дому», «домом» будут иметь больше общего, если рассматривать их на уровне символов, в то время как на уровне лемм, общего у них ничего не будет. Более того, согласно [Cavnar, Trenkle 1994], распределение символьных n-грамм подчиняется закону Ципфа – номер позиции n-граммы в частотном списке обратно пропорционален его частотности, что даёт возможность рассматривать их так же, как и слова. Убедиться в этом можно посмотрев на график, приведенный ниже.

Изображение 1. Частотный список биграмм

Под скипграммами в данной работе подразумеваются пары слов внутри одного предложения, расстояние между которыми не больше двух.

Для составления частотных списков слов и скипграмм была создана база лемм с помощью библиотеки *pymorphy2*. Таким образом получился корпус лемм из 27830 позиций. Списки n-грамм составлялись на основе первоначальных файлов с помощью библиотеки *nltk*.

Для анализа текстов внутри рэп стиля был создан подкорпус лемм и n-грамм для каждого рэп исполнителя отдельно.

## 1.2. Параметры сравнения текстов и подсчет расстояний

Расстояние между корпусами используется в компьютерной лингвистике для количественного сравнения корпусов и отдельных текстов, а также для автоматического определения авторства. В работах [Kilgariff 1997; Rayson et al. 2000] были заложены теоретические основы подобных исследований. Авторы предлагают разные параметры, по которым можно считать расстояния – начиная с частотных списков слов, символьных n-грамм, морфологических показателей и заканчивая пунктуацией.

### 1.2.1 Леммы

Что касается подсчёта частотности лемм, то здесь стоит учесть, что корпус не сбалансирован по количеству лемм в каждом из стилей, таким образом, использовать абсолютную частотность нельзя. Более того, в рэп стиле очень сильно распространено повторение одного и того же слова на протяжении всего припева. Тем самым, частота такого рода лемм будет описывать не весь корпус, а конкретную песню. То есть, если у нас встретилось 10 раз одно и то же слово в рэпе в одной песне, а в поп стиле на всём корпусе, то частотность данного слова будет одинаковой для двух стилей, однако совсем не показательной. Для решения данной проблемы используется ARF (average reduces frequency), который является одним из вариантов, предложенных в работах [Savický, Hlavácová 2002]. В [Пиперски 2018б] описывается принцип нахождения ARF, а также приводятся несколько формул нахождения. Соответственно, была написана программа, которая составляет список arf частотности для корпуса по формуле, которая али лучше всего подходит для автоматического подсчета, поскольку работает быстрее всего

,

где *f* – число вхождений слова *x* в корпус, – расстояние от *(i - 1)*-го вхождения до *i*-го вхождения *x*, а *v* – длина сегмента, которая рассчитывается по формуле , где *L* – длина корпуса.

Сами списки лемм были получены с помощью библиотеки *pymorhpy2.* В силу особенности данного лемматизатора, имелось несколько вариантов определения леммы. Для упрощения задачи брался первый из вариантов, который, по внутренним параметрам библиотеки, является самым вероятным.

### 1.2.2 N-граммы

Символьные последовательности дают рассматривать тексты исполнителей ниже уровня слов. Поскольку в нашем исследовании в n-граммы включены символы (пунктуация, переносы строк), авторов можно сравнивать также по этим параметрам.

Мы рассматривали последовательности длины от 2 до 5, поскольку это даёт разные объекты сравнения:

* Биграммы включают в себя предлоги, часть местоимений и аффиксы
* 3-4 граммы включают в себя уровень основ и корней слов, более того, согласно [Piperski 2019], n-граммы длины 4 являются лучшим показателем в автоматическом определении авторства
* N-граммы длины 5 могут включать в себя больше пунктуации, что даёт рассматривать тексты не только на уровне слов.

Для нахождения n-грамм мы воспользовались методом *ngrams()* библиотеки nltk, которая возвращает список n-грамм заданной длины. В данном случае, в силу несбалансированности корпусов по объему, для подсчета частотности n-грамм и скипграмм, использовалась *ipm* (instances per million), то есть

где *w(k)* – количество вхождений слова *k* в корпус длины *L*.

### 1.2.3 Скипграммы

Как уже говорилось, скипграммы в нашей работе – пары слов внутри предложения, расстояния между которыми не больше двух. Например, для предложения *«Катится Колобок по дороге, навстречу ему Заяц*», список скипграмм будет выглядеть как: *(катится, колобок), (катится, по), (катится, дороге), (колобок, по), (колобок, дороге), (колобок, навстречу), (по, дороге), (по, навстречу), (по, ему), (дороге, навстречу), (дороге, ему), (дороге, заяц), (навстречу, ему), (навстречу, заяц), (ему, заяц).*

Данный параметр позволяет рассматривать тексты на уровне выше слов, включая комбинаторные свойства слов, которые могут отличаться в зависимости от стиля и автора.

Дня нахождения списков скипграмм мы использовали ту же библиотеку *nltk,* функцию *skipgrams.* Частотность так же, как и для n-грамм рассматривалась в *ipm*.

### 1.2.4 Подсчет расстояний (леммы, n-граммы, скипграммы)

Согласно [Gomaa, Fahmy 2013], расстояние между корпусами можно находить как геометрические расстояния между многомерными векторами частотности. Вектора частотности – это такие вектора, выходящие из начала координат и имеющие частоты в качестве координат точки конца. Например, если имеется частотный список *я:100, и:20, по:10*, то вектор частотности будет выглядеть как *(100, 20,10)*.

Для получения векторов были созданы таблицы из трех столбцов для каждого из параметров. Первый столбец – список всех позиций (лемм, n-грамм и скипграмм) из корпусов рэпа и попа, второй – частота данной позиции в поп стиле, и третий, соответственно, в рэп стиле. Вырезка из таблицы ARF частотного списка приведена ниже.

|  |  |  |
| --- | --- | --- |
| Слово | Поп | Рэп |
| любовь | 1151.5 | 92.277 |
| но | 990.686 | 869.363 |
| а | 986.433 | 736.654 |
| твой | 840.0 | 326.327 |
| так | 713.688 | 535.559 |

Таблица 2. ARF список

На основании такого рода таблиц были созданы все необходимые вектора и мы приступили к подсчету расстояний.

#### 1.2.4.1 Косинусное расстояние

Первым расстоянием, которое мы посчитали, было косинусное. Оно находится через *сходство* векторов, а именно, через косинус угла между ними. Соответственно, формула *сходства* выглядит как

где – скалярное произведение векторов, а – длина вектора , – угол между векторами и .

Косинусное расстояние равно *1 – similarity*. Стоит отметить, что на материале частот, угол между векторами не может оказаться больше 90°, поскольку все координаты вектора неотрицательные. Таким образом, лежит в промежутке от 0 до 1, на котором значение косинуса уменьшается при увеличении угла. Таким образом, *similarity* так же уменьшается, а расстояние увеличивается. Получается, что минимальное расстояние между векторами – 0, максимальное – 1.

Для нахождения косинусного расстояния между векторами *a,b* в python есть функция *cosine(a,b)* библиотеки *scipy* [Jones et al 2001].

#### 1.2.4.2 Манхэттенское расстояние

Манхэттенское расстояние или *расстояние городских кварталов* между двумя векторами *p* и *q* в *n-мерном* пространстве рассчитывается по формуле

где p = (,,,…,), q = (,,,…,).

В той же библиотеке *scipy* есть функция *cityblock(a,b)* для нахождения манхэттенского расстояния между векторами *a,b.*

#### 1.2.4.3 Евклидово расстояние

Третья мера, которую мы использовали, было Евклидово расстояние. Для его нахождения между точками *p, q* в *n-мерном* используется такая формула:

В нашем случае вектора выходят из начала координат, тем самым координаты точек конца и есть координаты точек, между которыми нужно найти расстояние. Данное расстояние было посчитано с помощью функции *euclidean(a,b)* всё той же библиотеки *scipy.*

Стоит отметить, что согласно [Evert, Prois 2017], при нормализации векторов, все 3 вида расстояний дают одинаковые результаты. В том числе косинусное расстояние – единственное из набора, которое не нуждается в нормализации. Таким образом, поскольку в данной работе мы не занимались нормализацией, основным расстоянием будем считать косинусное.

### 1.2.5 Ключевые слова

Что касается нахождения расстояния по частотным спискам ключевых слов, оно было описано в работе [Kilgarriff 2009], а также [Пиперски 2018а]. В последней работе приводится алгоритм подсчета расстояния между корпусами по ключевым словам. Для получения списков ключевых слов, за каждым словом закрепляется ранг, который получается по формуле

где *A* – так называемый фокусный корпус, а именно корпус, для которого мы находим ключевые слова, B – референциальный корпус, с которым мы сравниваем фокусный, – частотность слова *w* в корпусе *P*, а *n* – свободный параметр, значение которого определяет выборку слов. Поскольку *n* никогда не принимает значение 0 (если слово в референциальном корпусе имеет частотность 0, то получится деление на 0, что невозможно), минимально возможное значение *n – 1*.

В нашей работе рассматривались 4 значения *n* – 1, 10, 100 и 1000. При *n = 1*, в верх списка ключевых слов фокусного корпуса входят слова, которые имеют очень высокую частотность в сравнении с частотностью в референциальном корпусе (в том числе те слова, которые вообще не встретились в нем). Повышая значения параметра, разница между числителем и знаменателем сглаживается, и более высокий ранг получают слова с меньшей разницей в частотности.

Далее, для нахождения самого расстояния, вычисляется среднее арифметическое списка коэффициентов, полученного по формуле:

Данный параметр позволяет получить списки слов, которые отличают сравниваемые тексы, выявить особенности лексикона исполнителя или жанра в целом.

# Глава 2. Анализ результатов

## 2.1 Сравнение рэп исполнителей

Посчитав все расстояния, мы получили сводные таблицы расстояний между каждой парой исполнителей. По каждому из параметров для каждого автора был определен самый похожий и самый не похожий исполнитель. Также, для лучшей визуализации результатов, все таблицы были представлены в виде *heatmaps* с помощью встроенной в excel функции. В результате получились таблицы[[1]](#footnote-1) вида

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **2rbina2rista** | **basta** | **brb** | **eldzhey** | **face** | **feduk** |
| **2rbina2rista** |  | 0,628351911 | 0,805314081 | 0,744591078 | 0,654267864 | 0,755163647 |
| **basta** | 0,628351911 |  | 0,600576034 | 0,446108108 | 0,269459922 | 0,490861891 |
| **brb** | 0,805314081 | 0,600576034 |  | 0,687043914 | 0,622998747 | 0,740965345 |
| **eldzhey** | 0,744591078 | 0,446108108 | 0,687043914 |  | 0,474170534 | 0,632598165 |
| **face** | 0,654267864 | 0,269459922 | 0,622998747 | 0,474170534 |  | 0,514389133 |
| **feduk** | 0,755163647 | 0,490861891 | 0,740965345 | 0,632598165 | 0,514389133 |  |
| **gnoiny** | 0,651782344 | 0,319895068 | 0,639374963 | 0,511899967 | 0,33439622 | 0,55959566 |
| **guf** | 0,63849052 | 0,249658255 | 0,617409349 | 0,452927859 | 0,298127324 | 0,522277133 |
| **husky** | 0,705812969 | 0,448924183 | 0,703967436 | 0,592792881 | 0,49526277 | 0,653942177 |
| **kasta** | 0,613326591 | 0,222846602 | 0,606278376 | 0,443122802 | 0,29089199 | 0,484326513 |
| **korzh** | 0,632676827 | 0,229624894 | 0,630831062 | 0,466463296 | 0,290843425 | 0,505936876 |
| **krovostok** | 0,612215581 | 0,238312654 | 0,60033419 | 0,445105225 | 0,292355005 | 0,49690858 |
| **kunteynir** | 0,663607469 | 0,32755972 | 0,638859411 | 0,477271703 | 0,398308656 | 0,572414612 |
| **lizer** | 0,652917713 | 0,246338896 | 0,624970983 | 0,463294171 | 0,252624987 | 0,537239896 |
| **lsp** | 0,675947944 | 0,314752967 | 0,660860594 | 0,497277208 | 0,349945043 | 0,55805995 |
| **morgenshtern** | 0,699743786 | 0,378701723 | 0,660241634 | 0,518327126 | 0,345619775 | 0,601313731 |
| **noizemc** | 0,599675918 | 0,205313638 | 0,594297985 | 0,418215164 | 0,266124994 | 0,483762814 |
| **skriptonite** | 0,648305422 | 0,261353682 | 0,615182233 | 0,4655982 | 0,267348255 | 0,500577054 |
| **timati** | 0,740004778 | 0,402263154 | 0,707603995 | 0,586813023 | 0,462811597 | 0,631182779 |
| **xleb** | 0,845517107 | 0,659574268 | 0,813102371 | 0,752344022 | 0,655292801 | 0,716449269 |

Таблица 3.4-граммы. Косинусное расстояние

Стоит отметить, что дальше будут рассмотрены в основном исполнители, которые отличаются большими значениями расстояний. Однако, при составлении таблиц, выделялись так же и самые похоже исполнители.

### 2.1.1 Леммы

На основании таблиц расстояний между авторами используя леммы, однозначных результатов не получилось. Некоторые исполнители попарно отличаются достаточно сильно, например, Morgenshtern и 2rbina2rista имеют самое большое расстояние между собой. В целом это объясняется тем, что 2rbina2rista имеет бóльшие расстояния до исполнителей, чем остальные в силу особого стиля текстов. В них много черных слов, связанных со смертью и насилием, при чем не только нецензурных. Для визуализации частотных слов данного исполнителя использовалось облако слов, сделанное с помощью интернет-инструмента WordArt. В данной выборке были исключены 100 самых частотных слов русского языка.



Изображение 2. Облако слов 2rbina2rista

Однако, если посмотреть таблицы Евклидова и Манхеттенского расстояния, то можно увидеть еще несколько исполнителей с достаточно высокими показателями расстояний, в сравнении с остальными. Особенно это видно на Манхеттенском расстоянии – очень сильно выделяется Noize MC, дальше Каста, Кровосток, Баста, Макс Корж, Лизер, Скриптонит. Можно предположить, что это связано с количеством нецензурной лексики, которая характерна для рэп стиля в целом, но отличается между исполнителями, например, Noize MC использует явно меньше матерной лексики, чем Morgenshtern. Посчитав какой процент составляет данная лексика в текстах каждого из исполнителей, мы получили такие результаты:

Изображение 3. Процент использования нецензурной лексики

Таким образом, данный параметр не является единственным, который дал такие результаты расстояний. Второе предположение – данные исполнители имеют больший словарный запас, либо используют специфическую лексику, которая характерна именно для их стиля текстов. Более детальный анализ, в том числе семантический, не входит в рамки нашего исследования.

### 2.1.2 Ключевые слова

Данный параметр не показал однозначных результатов, поскольку объемы корпусов каждого исполнителя маленькие, частоты слов небольшие, и, соответственно, ранги и расстояния по ключевым словам получаются маленькие.

Рассмотрев сводные таблицы ключевых слов, было обнаружено, что несколько исполнителей (Noize MC, Лизер), которые выделяются из картины наличием высоких показателей расстояния. Однако, они сильно отличаются не от всех исполнителей, то есть, например, Noize MC имеет большие расстояния с 2rbina2rista, Big Russian Boss, Feduk, однако с остальными показатели расстояний небольшие. Таким образом, это означает лишь совокупность особенностей каждого из пары исполнителей, которые отличают их друг от друга, а не особенности Noize MC и Лизера, которые отличают их от всех исполнителей в целом

Тем самым, данный параметр на нашем материале не дал важных результатов и не подтвердил наше предположение из 2.1.1 об особом словарном запасе Noize MC, Касты, Кровостока, Басты, Макса Коржа, Лизера и Скриптонита.

### 2.1.3 N-граммы

Данный параметр дал более однозначные результаты. А именно, исполнители, которые так или иначе выделялись наличием больших расстояний, в данном случае имели в большинстве низкие значения. Самыми «далекими» исполнителями получились 2rbina2rista, Big Russian Boss и Хлеб, причем расстояния между данными исполнителями и каждым из остальных отличаются на порядок, что можно увидеть на схеме, приведенной ниже.

Изображение 4. 4-граммы. Косинусное расстояние

На приведенном графике каждая линия отвечает за расстояния между исполнителем и каждым из остальных. Во-первых, можно увидеть, что на каждой из линий есть пики, которые сильно отличаются от остальных значений. Причем эти пики расположены на каждой из линий на одних и тех же координатах, что значит, что все сильно отличаются от одних и тех же исполнителей. Во-вторых, высота линии, на которой расположена линия показывает на сколько высокие значения содержаться у каждого из исполнителей. Как можно заметить, имеются несколько линий, которые находятся явно выше остальных, что тоже показывает на их отличие от остальных исполнителей.

Как уже было описано в пункте 1.2.2, n-граммы разной длины содержат разные уровни языка.

1. Биграммы. Сравнив списки биграмм четырех исполнителей (2rbina2rista, Хлеб, Big Russian Boss, Noize MC), мы получили предположение: у первых трех исполнителей высокая частотность переносов строк в отличие от последнего. А именно, у 2rbina2rista первый перенос строки встречается на 4 позиции частотного списка, у Хлеба на 54, у Big Russian Boss на 11, а у Noize MC на 74. Причем, следующие переносы встречаются гораздо ближе, чем у Noize MC. Просмотрев списки биграмм остальных исполнителей, мы убедились, что частый перенос строк, а значит более короткие строки, либо более длинные тексты песен (даже с учетом того, что корпус Нойза гораздо больше остальных) — это особенность 2rbina2rista, Хлеб и Big Russian Boss.
2. Триграммы-четырехграммы, как уже говорилось, содержат больше корней, часть грамматики. Согласно работе [Piperski 2019], именно четырехграммы лучше всего определяют особенности автора, тем самым, лучше, чем остальные, подходят для определения авторства. В совокупности с пунктами 2.1.1, 2.1.2 можно предположить, что авторы имеют особые стили использования слов, свои коллокации. Это будет проверено ниже на материале скипграмм. Просмотрев списки, стало понятно, что особый признак трех исполнителей (2rbina2rista, Хлеб, Big Russian Boss) в большом количестве звукоподражаний, которые находятся в вершине списка. Именно это отличие делает расстояния до них такими большими. У остальных исполнителей верх списка занимают высокочастотные слова русского языка, окруженные пунктуацией.
3. Пятиграммы. Данный параметр в нашем случае показал аналогичный трехграммам результат. При данной длине n-граммы, как уже говорилось, встречаются более длинные последовательности знаков препинания и специальных символов. Что касается содержания списков, то ситуация аналогична 3-4 граммам. Можно предположить, что высокая частотность ономатопоэтических слов получается в результате меньшего объема текстов рассматриваемых трех исполнителей по сравнению с остальными в совокупности с особой любовью к такого рода лексики.

### 2.1.4 Скипграммы

Данный параметр показал схожие результаты по разным мерам расстояний (косинусное, манхеттенское и евклидово). Самый однозначный результат дало третье расстояние. По нему самым не похожим на остальные получилась группа Хлеб. Сравнив списки скипграмм, мы пришли к выводу, что именно то, о чем говорилось в 1.2.1 о ARF является особенностью именно этого исполнителя, а именно, частый повтор одного или пары слов внутри одной песни.

Остальные две меры расстояний показали менее однозначные результаты. Максимальные расстояния всё еще принадлежат группе Хлеб, однако есть еще два исполнителя: 2rbina2rista и Big Russian Boss, которые так же имеют высокие показатели расстояний. Это стоит объяснить тем, что в скипграммы объединяют в себе несколько параметров, а именно, ключевые слова и словоформы. Таким образом, в совокупности с пунктом 2.1.3, мы получили особый стиль текстов данных трех исполнителей в области звукоподражания и особых комбинаций (в том числе повторных) слов.

### 2.1.5 Общий результат

В качестве визуализации сравнения исполнителей внутри рэп стиля, мы составили две таблицы: пары исполнителей с самыми высокими значениями расстояний и вторую, наоборот, с самыми низкими. С ними можно ознакомиться в приложении (Максимальное и минимальное расстояние между авторами). Как видно, по параметрам, которые анализируют уровень слов (леммы и ключевые слова), самым особенным и непохожим исполнителем оказался Noize Mc, а по всем остальным – Хлеб и 2rbina2rista.

Таблица с минимальными расстояниями не выглядит так же однозначно, однако просматривается тенденция к тому, что более старые исполнители, такие как Noize MC, Kasta, Баста, Timati и другие, имеют минимальные расстояния с исполнителями того же времени.

## 2.2 Сравнение поп и рэп стиля

В результате работы нашей программы по подсчету описанных в пункте 1.2 расстояний, были получены две таблицы – одна для расстояний по ключевым словам и вторая для остальных параметров. Ниже они приведены обе.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Мера | Тип подсчёта | | | Расстояние между поп и рэп |
| 2-граммы | Косинусное расстояние | | | 0.015490653705118151 |
| Манхэттенское расстояние | | | 208597.70908782276 |
| Евклидово расстояние | | | 12089.970646777116 |
| 3-граммы | Косинусное расстояние | | | 0.053494394928124134 |
| Манхэттенское расстояние | | | 400017.95044323266 |
| Евклидово расстояние | | | 9243.327240942906 |
| 4-граммы | Косинусное расстояние | | | 0.1213510427307859 |
| Манхэттенское расстояние | | | 645290.072356079 |
| Евклидово расстояние | | | 7203.173872692175 |
| 5-граммы | Косинусное расстояние | | | 0.2179507239757017 |
| Манхэттенское расстояние | | | 935496.9998199752 |
| Евклидово расстояние | | | 5740.934720082486 |
| ARF | Косинусное расстояние | | | 0.045483697200666695 |
| Манхэттенское расстояние | | | 75923.801 |
| Евклидово расстояние | | | 6527.147358912236 |
| Скипграммы | Косинусное расстояние | | | 0.1901860270008765 |
| Манхэттенское расстояние | | | 1565998.4066267947 |
| Евклидово расстояние | | | 4197.505013651678 |
| Ключевые слова | | Значение n | Расстояние между поп и рэп | | |
| 1 | 3.5500238314293506 | | |
| 10 | 1.3814590443269903 | | |
| 100 | 1.0560339958346343 | | |
| 1000 | 1.0072187869321347 | | |

Таблица 4. Результаты подсчета расстояний Поп vs Рэп

Рассмотрим результаты по порядку, аналогично 2.1

### 2.2.1 Леммы

 Как видно из таблицы, расстояние между текстами на уровне лемм (ARF) очень маленькое. Поскольку, согласно 1.2.4, разные меры расстояний показывают соизмеримые расстояния. Тем самым, можно ориентироваться по одной мере. Также, поскольку мы определили минимальное и максимальное значение косинусного расстояния, то значение 0,04 является низким.

Изображение 6 Облако слов Рэп стиль

Изображение 5. Облако слов Поп стиль

Для визуализации разницы между корпусами, мы создали облака слов для каждого из стилей, предварительно убрав 100 самых частотных слов русского языка для того, чтобы можно было увидеть разницу между корпусами.

Стоит отметить, что самые частотные слова в обоих списках – это самые частотные слова русского языка, а их количество в разы больше, чем слов, приведенных в таблице (первые позиции ARF списка имеют ранги более 4000, в то время как слова из облаков – менее 400). Таким образом, данный параметр показывает такие маленькие расстояния в силу объема корпуса и большого совпадения высокочастотных слов русского языка.

### 2.2.2 Ключевые слова

В данном разделе, как мы и предполагали, основное различие корпусов – наличие большого количества мата в корпусе рэпа. Именно это является причиной столь высокого расстояния при N = 1 поскольку, если сравнить списки ключевых слов, то при именно этот пласт лексики занимает верхние позиции списка с высокими показателями ранга.

При N = 10, 100 расстояние уменьшилось, поскольку по формуле нахождения расстояния по ключевым словам, сглаживается разница между частотностями, тем самым в списке появляется большее общей лексики, а именно, той, которая присуща русскому языку. А в силу того, что корпуса большого объема, частоты таких слов более-менее одинаковы и, тем самым, уменьшают расстояние.

При N = 1000 количество нецензурной лексики уменьшилось, хотя не ушло совсем. Зато, появилась разница между списками по тематике слова. А именно, рэп стиль обилует словами достаточно отличной от поп стиля тематики: деньги, пацан, район, брат, смерть. Как известно, современный рэп изначально был протестом, что объясняет наличие этих слов в описываемых списках.

Таким образом, при N=1000 помимо матерных слов на расстояние оказывает влияние и тематика слов.

### 2.2.3 N-граммы

Данный параметр на материале корпусов поп и рэп стиля рассматривать как основополагающий нельзя – большой объем корпусов даёт похожие списки, поскольку все списки n-грамм построены на материале одного языка. То есть последовательности, характерные для языка на большом материале будут иметь схожие частотности. А чем больше объем корпуса, тем больше характерных последовательностей.

Однако, стоит заметить, что с увеличением длины последовательности, увеличивается и расстояние между корпусами. Это можно объяснить тем, что в последовательности большей длины входят словоформы, и, соответственно, корни. А как мы выяснили в 2.2.2, для поп и рэп стиля характеры свои слова, что немного увеличивает расстояние между корпусами.

### 2.2.4 Скипграммы

Данный параметр не показал высокого показателя расстояний по причине, описанной в 2.2.3 – оба корпуса большого объема, на русском языке. Таким образом, бóльшая часть списка скипграмм – высокочастотные биграммы слов русского языка. Таким образом, для получения различий между корпусами текстов поп и рэп стилей, необходимо убрать из списков такие последовательности. Однако это приведет к смешению параметров, которые мы рассматриваем, а именно, это уже будет близкий к ключевым словам метод.

## 2.3 Выводы

В результате анализа полученных количественных сравнений текстов рэп исполнителей, мы получили однозначный вывод – тексты рэперов «старой» школы сильно отличаются от текстов современных рэперов. Более того, в совокупности с 2.2, мы пришли к выводу, что на меньших объемах n-граммы и скипграммы длины 2 показывают более четкие результаты, а на больших стоит использовать параметр ключевых слов.

Что касается сравнения поп и рэп стиля по данным параметрам, то они оказались схожи по многим показателям, поскольку корпуса имеют большой объем, а тем самым во всех частотных списках будет иметься пласт позиций, которые присущи русскому языку в целом. Однако, нам удалось подтвердить предположение об основном различие текстов данных стилей – наличие нецензурной лексики в рэп стиле и почти полное её отсутствие в поп стиле, а также нашли особенности лексики каждого из стилей.

# Глава 3. Автоматическое определение авторства

Для определения авторства мы использовали новый способ – комбинацию самых значимых параметров, которые мы определили во второй главе. Основная идея данного алгоритма – найти исполнителя, который по всем рассмотренным параметрам похож на неизвестного.

Реализация данного алгоритма будет предложена в двух вариантах.

## 3.1 Параметры

Таким образом, мы рассматривали параметры:

* 4 граммы, поскольку, согласно [Piprski 2019] n-граммы именно этой длины лучше всего подходят для автоматического определения авторства
* Ключевые слова (N = 1, 1000). Хотя они и не показали однозначных результатов, этот параметр при рассмотрении двух значений N даёт более точные результаты
* Скипграммы. Они показали достаточно хорошие результаты на нашем материале, тем самым будут хорошим показателем при автоматическом определении авторства.

Все расстояния, кроме расстояния по ключевым словам, мы считали с помощью косинусного расстояния, поскольку именно значение косинусного расстояния ближе всего к расстояниям по другим мерам (на нашем материале).

## 3.2 Алгоритмы определения автора

### 3.2.1 Первый вариант

Данный вариант реализации работает, если имеются матрицы расстояний по описанным выше параметрам между исполнителями, среди которых нужно найти автора текста. Алгоритм состоит из нескольких этапов.

Для удобства, текст, автора которого нам нужно определить, будем называть *искомым.*

1. Посчитать расстояния между искомым текстом и каждым из имеющихся исполнителей. Тем самым, мы получим новую строчку матрицы. Аналогично для всех четырех параметров.
2. Определить, на какую строчку из имеющейся первоначально матрицы больше всего похож искомый текст. Здесь есть два пути:
   1. Первый – сравнивать сумму разниц между каждой позицией в строке. Для более понятного объяснения, ниже приведена матрица, строка расстояний искомых текстов и результат. По сути, данный шаг схож с алгоритмом Хмелёва [Дроздова 2017].

Пусть имеется матрица расстояний между 5 исполнителями:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
|  | 0 |  |  |  |  |
|  |  | 0 |  |  |  |
|  |  |  | 0 |  |  |
|  |  |  |  | 0 |  |
|  |  |  |  |  | 0 |

Строка расстояний от искомого текста до каждого из них:

Для определения максимально близкой последовательности каждая строчка матрицы преобразуется в модуль разницы расстояний:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ( |  |  |  | ) |

Далее выбирается строчка с самой маленькой суммой. Таким образом, мы получаем исполнителя, на которого искомый больше всего похож по данному параметру.

* 1. Второй вариант – перейти к многомерным векторам и посчитать, какой из них ближе всего к вектору искомого текста. Сведение к векторам и нахождение расстояний между ними мы уже рассматривали в 1.2.4. Этот вариант будет работать медленнее, поэтому мы решили использовать в данном варианте алгоритма первый вариант.

В любом из вариантов нахождения, мы получаем список из 4 авторов (по автору на параметр). То есть, у нас искомый автор по первому параметру похож на одного, по второму на другого и т. д. Возможна ситуация, когда какие-то авторы среди 4 совпадают, но в нашем случае это не влияет на алгоритм.

1. Определить среди имеющихся авторов, который лучше всех остальных сочетает в себе схожести и различия. То есть нам нужно найти исполнителя, который будет так же, как и искомый, по каждому параметру похож на полученного в пункте 2. Для этого мы снова используем имеющиеся матрицы и создаём еще одну, которая является таблицей расстояний между каждым из найденных 4 авторов по каждому параметру. То есть, строки таблицы – все имеющиеся авторы, столбцы – расстояния между авторами по параметру.

Получается, что таблица будет 20 на 4 (на нашем материале). Составив таблицу, мы считаем сумму расстояний по каждой строке и находим среди них минимальное. Таким образом, мы получаем одного автора, который по 4 параметрам похож на найденных исполнителей. Этот автор и есть автор искомого текста.

### 3.2.2 Второй вариант

Данный вариант никак не связан с имеющимися таблицами. Фактически, данный способ сводит задачу к нахождению самого короткого вектора.

1. Первый пункт аналогичен первому пункту 3.2.1 – найти расстояния между искомым тестом и каждым из исполнителей по всем четырем параметрам.
2. Мы составляем четверки чисел (расстояние до искомого текста по каждому параметру) для каждого из исполнителей. Таким образом, мы получаем 20 (на нашем материале) четверок чисел: первое число – расстояния по ключевым словам при N = 1, второе – при N = 1000, третье – косинусное расстояние по символьным n-граммам и четвертое – косинусное расстояние по скипграмам.
3. Самым простым способом среди таких последовательностей найти минимальную – представить их в виде четырехмерных векторов из начала координат. Поскольку у нас числа не равноправные, а именно, изменение на 1 расстояния по ключевым словам не равняется изменению на единицу косинусного расстояния, то использовать первоначальные числа нельзя. Для того, чтоб сделать их изменения более близкими, первые два числа мы делим на целую часть большего из них. Таким образом, у нас получится четыре числа от 0 до 1.
4. Находим квадраты длин данных векторов по формуле

где L – длина вектора. Среди полученных длин выбираем самую маленькую. Автор, которому соответствует этот вектор и есть искомый вектор.

## 3.3 Результаты

### 3.3.1 Внутренняя проверка

Внутреннее тестирование проводилось на базе уже имеющихся текстов. То есть, задачей внутреннего тестирования было определить работоспособность алгоритмов на материале, который полностью совпадает или пересекается. Если алгоритм не сработает даже на таком материале, то с большей вероятностью он не будет работать и на непересекающемся материале.

После нескольких тестирований и просмотров результатов, стало понятно, что параметр ключевых слов на нашем материале не работает. А именно, именно он при разборе промежуточных матриц и векторов неправильно определял авторство даже на полные тексты исполнителей. Поэтому было решено попробовать 2 варианта: оставить два параметра – косинусное расстояние по символьным n-граммам и по скипграммам, либо заменить ключевые слова на косинусное расстояние по ARF частотности. Оба этих варианта улучшают результат, поскольку все расстояния получаются одной природы и сравнивать их более правильно. В том числе пропадает необходимость в нормализации векторов и добавлении масс.

Таким образом, у нас получилось два улучшенных алгоритма, реализации которых на языке *python* можно найти в приложении. Для второго тестирования эффективности алгоритмов мы добавили изменяемый параметр – количество текстов. Для сравнения результатов и подсчета процента верных определений были рассмотрены определения авторства по половине, четверти, восьмой и шестнадцатой количества текстов (каждого из исполнителей). В результате мы получили две таблицы результатов, которые можно увидеть ниже. “Faled” означает, что в определяемом тексте не нашлось ни одной песни без английский слов.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Верный ответ | 2 | 4 | 8 | 16 |
| 2rbina2rista | 2rbina2rista | 2rbina2rista | 2rbina2rista | 2rbina2rista |
| basta | basta | lsp | feduk | brb |
| brb | brb | brb | brb | brb |
| eldzhey | eldzhey | xleb | xleb | xleb |
| face | face | morgenshtern | feduk | eldzhey |
| feduk | feduk | Faled | Faled | Faled |
| gnoiny | morgenshtern | morgenshtern | husky | husky |
| guf | gnoiny | morgenshtern | 2rbina2rista | 2rbina2rista |
| husky | brb | xleb | xleb | 2rbina2rista |
| kasta | gnoiny | kunteynir | morgenshtern | feduk |
| korzh | korzh | morgenshtern | timati | feduk |
| krovostok | krovostok | gnoiny | kunteynir | feduk |
| kunteynir | husky | xleb | xleb | xleb |
| lizer | lizer | lsp | morgenshtern | feduk |
| lsp | timati | brb | xleb | xleb |
| morgenshtern | timati | brb | xleb | xleb |
| noizemc | kasta | guf | kunteynir | eldzhey |
| skriptonite | skriptonite | lsp | eldzhey | feduk |
| timati | timati | feduk | xleb | Faled |
| xleb | xleb | xleb | xleb | xleb |

Таблица 5. Результаты первого варианта алгоритма

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Верный ответ | 2 | 4 | 8 | 16 |
| 2rbina2rista | 2rbina2rista | 2rbina2rista | 2rbina2rista | 2rbina2rista |
| basta | basta | basta | basta | basta |
| brb | brb | brb | brb | brb |
| eldzhey | eldzhey | eldzhey | eldzhey | eldzhey |
| face | face | face | face | face |
| feduk | feduk | Faled | Faled | Faled |
| gnoiny | gnoiny | gnoiny | gnoiny | gnoiny |
| guf | guf | guf | guf | guf |
| husky | husky | husky | husky | husky |
| kasta | kasta | kasta | kasta | kasta |
| korzh | korzh | korzh | korzh | korzh |
| krovostok | krovostok | krovostok | krovostok | krovostok |
| kunteynir | kunteynir | kunteynir | kunteynir | kunteynir |
| lizer | lizer | lizer | lizer | lizer |
| lsp | lsp | lsp | lsp | lsp |
| morgenshtern | morgenshtern | morgenshtern | morgenshtern | morgenshtern |
| noizemc | noizemc | noizemc | noizemc | noizemc |
| skriptonite | skriptonite | skriptonite | skriptonite | skriptonite |
| timati | timati | timati | timati | Faled |
| xleb | xleb | xleb | xleb | xleb |

Таблица 6. Результаты второго варианта алгоритма

Как можно заметить, первый вариант дал очень малый процент правильного определения авторства 21%. Это связано с тем, что разницы расстояний, которые мы считали, очень малы и сравниваются очень близкие дельты. Второй вариант показал 100% правильное определение авторства. Здесь стоит учесть, что тексты, по которым проводилось тестирование, уже были в базе, с которой считались расстояния. Тем самым, это не совсем чистый эксперимент.

### 3.3.2 Внешняя проверка

Для того, чтоб подтвердить эффективность второго варианта алгоритма, мы решили проверить его на базе стихов, которую мы собрали с помощью поэтического подкорпуса НКРЯ. В базу вошли 30 поэтов, со списком которых можно ознакомиться ниже.

|  |  |  |  |
| --- | --- | --- | --- |
| Автор | Произведения | Словоформы | Символы |
| А. А. Ахматова | 945 | 8463 | 244390 |
| А. А. Блок | 1350 | 18388 | 554988 |
| А. А. Фет | 920 | 14298 | 462043 |
| А. И. Несмелов | 451 | 11943 | 438618 |
| А. К. Толстой | 301 | 17783 | 695868 |
| А. Н. Апухтин | 354 | 9120 | 303218 |
| А. Н. Майков | 561 | 18492 | 625182 |
| А. П. Сумароков | 283 | 11575 | 432734 |
| А. С. Пушкин | 904 | 36846 | 1110286 |
| А. Т. Твардовский | 346 | 20554 | 649976 |
| Андрей Белый | 503 | 12160 | 356530 |
| Б. А. Слуцкий | 1287 | 21098 | 668062 |
| Б. Л. Пастернак | 531 | 13520 | 461222 |
| Б. П. Корнилов | 188 | 10298 | 315078 |
| Б. Ю. Поплавский | 547 | 8940 | 294473 |
| В. А. Жуковский | 695 | 43437 | 1450724 |
| В. А. Луговской | 150 | 9860 | 341785 |
| В. В. Маяковский | 634 | 28403 | 925093 |
| В. В. Набоков | 591 | 10560 | 344474 |
| В. В. Хлебников | 290 | 11507 | 288945 |
| В. Г. Бенедиктов | 271 | 9633 | 296257 |
| В. И. Иванов | 1180 | 23095 | 762129 |
| В. И. Майков | 112 | 8944 | 304696 |
| В. К. Тредиаковский | 162 | 9079 | 354902 |
| В. Я. Брюсов | 1683 | 31266 | 1079235 |
| Г. Н. Оболдуев | 387 | 14370 | 356965 |
| Г. Р. Державин | 415 | 18308 | 543659 |
| Д. Л. Андреев | 452 | 17110 | 582978 |
| Д. С. Мережковский | 338 | 13666 | 476057 |
| Д. Самойлов | 900 | 13095 | 388983 |
| **Всего:** | **17731** | **495811** | **16109550** |

Таблица 7. Сводная базы проверки алгоритма

Для более чистых результатов тексты каждого автора были разбиты на две части. В первый группу попали тексты с четными номерами в нашем корпусе, во вторую – с нечетными. Это было сделано, поскольку, согласно [Балуева 2020], тексты могут иметь разные расстояния в зависимости от времени написания (от периода творчества), поэтому мы их смешали между собой, чтобы сгладить разницу.

Таким образом, у нас получилось 2 базу текстов: первая группа – база по которой будет определяться авторство, вторая группа – тексты, авторов которых необходимо определить. При условии, что объему базы большие, задачка определения не составляет особо труда – в отличие от рэп исполнителей, расстояния между поэтами гораздо выше, и на большем материале эти различия лучше видны.

Поэтому мы уменьшили базы в 8 раз и сравнили результаты. В результате выполнения программы, из 30 авторов правильно определилось 11, что можно увидеть в таблице ниже. Последний столбец – время, прошедшее с начала выполнения программы.

|  |  |  |
| --- | --- | --- |
| Правильный ответ | **Result** | **Time** |
| Ахматова | В. А. Жуковский | 1515,450807 |
| Блок | В. Я. Брюсов | 3479,837254 |
| Фет | Д. С. Мережковский | 5141,6134 |
| **Несмелов** | **А. И. Несмелов** | **6677,798303** |
| **Толстой** | **А. К. Толстой** | **11608,68454** |
| Апухтин | Д. С. Мережковский | 13164,28758 |
| Майков | В. А. Жуковский | 52898,84768 |
| Сумароков | В. А. Жуковский | 17100,32224 |
| Пушкин | В. А. Жуковский | 21381,45144 |
| **Твардовский** | **А. Т. Твардовский** | **24534,65339** |
| Белый | В. В. Маяковский | 26172,83766 |
| **Слуцкий** | **Б. А. Слуцкий** | **28539,72935** |
| **Пастернак** | **Б. Л. Пастернак** | **32518,70748** |
| **Корнилов** | **Б. П. Корнилов** | **33798,52487** |
| Поплавский | В. Я. Брюсов | 35098,64169 |
| **Жуковский** | **В. А. Жуковский** | **39759,75738** |
| Луговской | Б. П. Корнилов | 41033,12651 |
| **Маяковский** | **В. В. Маяковский** | **44174,65431** |
| Набоков | В. Я. Брюсов | 45350,2912 |
| Хлебников | В. А. Жуковский | 46659,75242 |
| Бенедиктов | В. А. Жуковский | 47945,35991 |
| **Иванов** | **В. И. Иванов** | **50398,54836** |
| В. И.Майков | В. А. Жуковский | 52898,84768 |
| **Тредиаковский** | **В. К. Тредиаковский** | **54233,38045** |
| **Брюсов** | **В. Я. Брюсов** | **57519,63748** |
| Оболдуев | А. Т. Твардовский | 59302,77829 |
| Державин | В. А. Жуковский | 60929,04808 |
| Андреев | В. В. Маяковский | 62844,07809 |
| **Мережковский** | **Д. С. Мережковский** | **65062,47163** |
| Самойлов | Б. А. Слуцкий | 66425,69303 |

Таблица 8. Результаты внешнего тестирования

Просмотрев вектора, мы пришли к выводу, что слабым местом данного алгоритма является может быть две вещи:

1. квадрат длин векторов, который находится через сумму квадратов координат. А квадрат сильно влияет на результат. Например, если есть векторы , , причем , а , то длина второго первого вектора может оказаться больше длины второго и, тогда, вектор, который подходит по двум параметрам не войдет в результат, хотя он больше подходит.
2. для задачи автоматического определения авторов на материале текстов классических поэтов, значимыми мерами являются отличные от тех, которые мы определили для рэп исполнителей.

Таким образом, предложенный алгоритм нуждается в доработке.

# Заключение

В работе было две основные подзадачи. Во-первых, мы должны были количественно проанализовать тексты рэп исполнителей и сравнить их с текстами в жанре поп. Для этого мы автоматически собрали корпуса и разметили их. Анализ проводился на основании расстояний между текстами. Второй подзадачей был анализ и тестирование двух алгоритмов автоматического определения авторства.

Анализ проводился на основании расстояний между текстами. В результате, на нашем материале из набора *n-грамммы, скипграммы, ключевые слова, ARF частотности*, значимыми оказались только *n-граммы и скипграммы*. Остальные же меры показали очень близкие результаты без сильно отличающихся авторов. Однако, нам удалось получить, что рэп исполнителей можно разделить на две большие группы – «старая» и «новая» школа. А именно, исполнители старшего поколения (Noize MC, Basta, Timati и тд) и нового (Face, Feduc, Morgenshtern).

Аналогично этому, проводился анализ сравнения текстов поп и рэп стилей. Как результат, мы получили значительное подобие текстов на уровне символьных последовательностей, которое объясняется большим объемом корпусов. А на уровне слов и последовательностей слов, тексты стилей имели достаточно высокие значения расстояний. Это можно объяснить специализированными лексиконами, которые отличают данные стили (начиная от нецензурной лексики, присущей в большей степени рэп стилю, и заканчивая лексикой протеста/любви/романтики).

Эта часть исследования проводилась на русских текстах выбранных исполнителей, то есть в силу алгоритма сбора базы, все песни, в которых находились английские слова (слова, написанные латиницей), не вошли в финальную базу. Это немного снизило расстояния между авторами и уменьшило наполненность частотных списков. Многие рэперы активно использую иностранную лексику, в том числе названия своих групп/свои имена, которые написаны в большинстве случаев латиницей.

Что касается второй части работы про автоматическое определение авторства, то нами было предложено два альтернативных алгоритма определения авторства, которые совмещают в себе сразу несколько самых значимых для нашего материала параметров. Идея первого параметра основана на использовании дельты расстояний между текстом неизвестного автора и каждым из корпусов известных, второй – на длине вектора, который образуется из расстояний по каждому из выбранных нами критериев. На нашем материале второй вариант оказался более действенным, однако после проверки на базе стихов классических поэтов, эффективность данного параметра не подтвердилась. Мы пришли к выводу, что второй вариант нуждается в более детальной проверке, а также доработке, поскольку сравнение длин векторов не приводит к желаемому результату. Самое вероятное продолжение данного исследования – скрещивание двух вариантов алгоритма и их проверка на разном материале.

Фактически, в данной работе приведены значимые меры при сравнении текстов поп и рэп исполнителей, а также идея и реализация алгоритма, который может совместить в себе необходимые для автоматического определения авторства меры.

# Список литературы

1. Cavnar, W.B. and Trenkle, J. M. 1994. N-grambased text categorization. In Proceedings of SDAIR-94 / 3rd Annual Symposium on Document Analysis and Information Retrieval, Las Vegas, 161-175, 1994
2. Cavnar, William B. and Vayda, Alan J., Using superimposed coding of N-gramlists for Efficient Inexact Matching / Pro-ceedings of the Fifth USPS Advanced Tech-nology Conference, Washington D.C., 1992.
3. Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schutze, Introduction to Information Retrieval, Cambridge University Press, New York, USA, pp.126–129,2008.
4. D’hondt E., Verberne S., Weber N., Koster K., Boves. L. Using skipgrams and PoS-based feature selection for patent classification. Computational Linguistics in the Netherlands Journal 2: 52–70, 2012.
5. Gerard Salton, A. Wong, and C. S. Yang. A vector space model for information retrieval. Communications of the ACM, 18(11):613–620, 1975.
6. Gomaa W. H., Fahmy A. A. A Survey of Text Similarity Approaches International / Journal of Computer Applications, 68(13), April, pp. 13–18, 2013.
7. Jones E, Oliphant E, Peterson P, et al. SciPy: Open Source Scientific Tools for Python, 2001.// URL: http://www.scipy.org
8. Kilgarriff A. Simple maths for keywords / Proceedings of Corpus Linguistics Conference CL2009, University of Liverpool, UK, July 2009.
9. Piperski A. Authorship Attribution with a Very Naïve Bayes Model and What It Can Tell Us about Russian Poetry. In Computational Linguistics and Intellectual Technologies. Papers from the Annual International. Conference “Dialogue”. Issue 18, 2019.
10. Rayson, P., & Garside, R. Comparing corpora using frequency profiling. / In Proceedings of the Comparing Corpora Workshop at ACL 2000. Hong Kong, 2000.
11. Savický P., Hlavácová J. Measures of Word Commonness, Journal of Quantitative Linguistics, 9:3, 215-231, 2002
12. Scott M. PC analysis of key words — and key key words / System 25(2),pp. 233–245, 1997.
13. [Stefan Evert](javascript:;), [Thomas Proisl](javascript:;), [Fotis Jannidis](javascript:;), [Isabella Reger](javascript:;), [Steffen Pielström](javascript:;), [Christof Schöch](javascript:;), [Thorsten Vitt](javascript:;) – Understanding and explaining Delta measures for authorship attribution/Digital Scholarship in the Humanities, Volume 32, Issue suppl\_2, December 2017, Pages ii4–ii16, <https://doi.org/10.1093/llc/fqx023>
14. Tan, Chade-Meng, Yuan-Fang Wang, and Chan-Do Lee (2002), The use of bigrams to enhance text categorization, Information Processing and Management 38 (4), pp. 529–546.
15. Балуева Д. В. Автоматическая периодизация авторских корпусов. Конференция диалог, 18 июня 2020
16. Балуева Д. В. Курсовая работа, Современный поэт и поэты серебряного века: количественное сравнение текстов, М.2019
17. Дроздова, И. И. Определение авторства текста по частотным характеристикам / И. И. Дроздова, А. Д. Обухова. — Текст : непосредственный // Технические науки в России и за рубежом : материалы VII Междунар. науч. конф. (г. Москва, ноябрь 2017 г.). — Москва : Буки-Веди, 2017. — С. 18-21. — URL: https://moluch.ru/conf/tech/archive/286/13237/ (дата обращения: 10.06.2020)
18. О. В. Кукушкина, А. А. Поликарпов, Д. В. Хмелёв – Определение авторства текста с использованием буквенной и грамматической информации/Пробл. передачи информ., 2001,том 37, выпуск 2, 96–109
19. Пиперски А.Ч. Работа 1 – Измерение расстояний между текстами / Малый Мехмат, 10 февраля 2018
20. Пиперски А.Ч. Работа 2 – Как и зачем считать частотность слов в текстах / Малый Мехмат, 12 мая 2018

## Интернет ресурсы

1. <http://ruscorpora.ru/new/sbornik2008/05.pdf>
2. <http://www.ruscorpora.ru/search-poetic.html>
3. https://genius.com/

# Приложение

## Программы

1. Программа 1. Определение стиля и подсчет расстояний
2. **import** requests
3. **import** re
4. **from** bs4 **import** BeautifulSoup
5. **import** time
6. **import** warnings
7. **import** os
8. **from** tkinter **import** \*
9. **import** nltk
10. **import** pymorphy2
11. **from** threading **import** Thread
12. **from** nltk **import** ngrams, skipgrams
13. **from** scipy.spatial **import** distance
14. **from** prettytable **import** PrettyTable
16. startTime = time.time()
17. faledLinks = []
18. inDirectory = ['C:/Users/epish/Desktop/учеба/ДИПЛОМ/ПопТексты/Input/', 'C:/Users/epish/Desktop/учеба/ДИПЛОМ/РэпТексты/Input/']
19. outDirectory = ['C:/Users/epish/Desktop/учеба/ДИПЛОМ/ПопТексты/Output/', 'C:/Users/epish/Desktop/учеба/ДИПЛОМ/РэпТексты/Output/']
20. baseDirectory = ['C:/Users/epish/Desktop/учеба/ДИПЛОМ/ПопТексты/Bases/', 'C:/Users/epish/Desktop/учеба/ДИПЛОМ/РэпТексты/Bases/']
21. resultDirectory = 'C:/Users/epish/Desktop/учеба/ДИПЛОМ/Results/'
23. '''------------------------------------------------- К Н О П О Ч К И -------------------------------------------------------------'''
25. #Вызывает создание базы с нуля
26. **def** CreateButton():
27. **for** i **in** range(len(outDirectory)):
28. **if** len(os.listdir(outDirectory[i])) == 0:
29. BaseDownloader(InFileAll(inDirectory[i]), inDirectory, outDirectory, i)
30. text.insert('end', 'All bases are created\n')
31. text.see("end")
33. # Вызывает создание базы лемм
34. **def** LemmaCorpButton():
35. **for** i **in** range(len(baseDirectory)):
36. **if** os.path.exists(baseDirectory[i]+'lemmas.txt') != True:
37. MorphCorpras(baseDirectory[i])
38. text.insert('end', 'All lemma bases are created\n')
39. text.see("end")
41. #Вызывает очистку базы
42. **def** CleanButton(\*mode):
43. **if** len(mode) == 0:
44. **for** i **in** range(len(baseDirectory)):
45. **if** os.path.exists(baseDirectory[i]+'cleanbase.txt'):
46. Cleaner(baseDirectory, outDirectory, i)
47. text.insert('end', 'All bases are cleaned\n')
48. text.see("end")
50. #Вызывает догрузку файлов в первоначальную базу
51. **def** RefreshButton():
52. **for** directoryNum **in** range(len(inDirectory)):
53. notInBase = [name **for** name **in** [title **for** title **in** list(filter(**lambda** x: x.endswith('txt'), os.listdir(inDirectory[directoryNum])))]
54. **if** 'out'+name **not** **in** list(filter(**lambda** x: x.startswith('out'), os.listdir(outDirectory[directoryNum])))]
55. text.insert('end', inDirectory[directoryNum]+':\n')
56. **if** len(notInBase) != 0:
57. flag = 'worked'
58. **for** name **in** notInBase:
59. text.insert('end', name + ' is not uploaded\n')
60. text.see('end')
61. refresh = Tk()
62. refresh.title('Refresher')
63. refresh.geometry('200x100+300+200')
64. btnRefresh = Label(refresh, text = 'Upload new files?')
65. btnYes = Button(refresh, text = 'yes', width = 10, command = **lambda**: Refresher(directoryNum,notInBase, refresh))
66. btnNo = Button(refresh, text = 'close', width = 10, command = refresh.destroy)
67. btnYes.place(x = 10, y = 50)
68. btnNo.place(x = 110, y = 50)
69. btnRefresh.pack()
70. refresh.mainloop()
71. **else**:
72. **try**:
73. **if** flag == 'worked':
74. refresh.destroy()
75. text.insert('end','All files are uploaded\n')
76. **except** UnboundLocalError:
77. text.insert('end','All files are uploaded\n')
79. #Закрывает окно приложения
80. **def** CloseButton():
81. root.destroy()
83. '''------------------------------------------------ И Н Ф О Р М А Ц И Я -------------------------------------------------'''
85. **def** FullInformation():
86. **global** inDirectory, outDirectory, baseDirectory
87. informationText = 'Base information'
88. artists = len(list(filter(**lambda** x: x.startswith('out'), os.listdir(outDirectory[0])))+
89. list(filter(**lambda** x: x.startswith('out'), os.listdir(outDirectory[1]))))
90. informationText += '\nArtists:\t'+str(artists)
91. alboms = 0
92. artists = 0
93. songs = 0
94. words = 0
95. lemmas = 0
96. **for** i **in** range(len(inDirectory)):
97. **for** artist **in** InFileAll(inDirectory[i]):
98. alboms += len(FileReader(inDirectory[i],artist))
99. **try**:
100. songs += len(FileReader(baseDirectory[i], 'clean.txt'))
101. **except** FileNotFoundError:
102. text.insert('end', outDirectory[i]+' Need cleaning!\n')
103. **try**:
104. word = FileReader(baseDirectory[i], 'lemmas.txt')
105. words += len(word)
106. lemmas += len(list(set(word)))
107. **except**:
108. text.insert('end', baseDirectory[i]+' Need lemmatization!\n')
110. informationText += '\nAlbums:\t' + str(alboms) + '\nSongs:\t' + str(songs) + '\nWords:\t' + str(words) + '\nLemmas:\t' + str(lemmas)
111. information['text'] = informationText
113. '''--------------------------------------------- Р А Б О Т А  С  Б А З О Й -------------------------------------------------'''
115. **def** FileReader(directory, filename):
116. with open(directory+filename, 'r', encoding = 'utf-8') as inFile:
117. **return** [line **for** line **in** inFile]
119. **def** FileWriter(directory, filename, corpra):
120. with open(directory+filename, 'w', encoding = 'utf-8') as outFile:
121. **for** line **in** corpra:
122. outFile.write(line+'\n')
124. #Возвращает html код страницы по ссылке
125. **def** GetHTML(link):
126. #time.sleep(1)
127. **return** requests.get(link).text
129. #Возвращает всё о песне по ссылке
130. **def** GetSong(link):
131. soup = BeautifulSoup(GetHTML(link), features="lxml")
132. Artist = soup.find('div', {'class':'song\_album-info'}).find('a', {'class': 'song\_album-info-artist'}).text
133. SongTitle = soup.find('div', {'class':'header\_with\_cover\_art'}).find('h1').text
134. AlbumTitle = soup.find('div', {'class':'song\_album-info'}).find('a').get('title')
135. SongText = [re.sub(r'\xa0', ' ', line2) **for** line2 **in**
136. [re.sub(r'\[.\*\]', '', line) **for** line **in** soup.find('div', {'class':'lyrics'}).text.split('\n') **if** len(line) > 0]
137. **if** len(line2) > 0]
138. **return** (AlbumTitle, SongTitle, SongText)
140. #Возвращает список всех песен в альбоме по ссылке на альбом
141. **def** CreateSongLinks(AlbomLink):
142. **return** [re.sub(r'<a href=\"(.\*)\"', r'\1', link) **for** link **in** re.findall(r'<a href=\"[a-zA-Z 2:/\.\-]+\-lyrics\"', GetHTML(AlbomLink))]
144. #Возвращает альбом по списку ссылок на каждую песню
145. **def** AlbomMaker(SongsLinks):
146. albom = []
147. linkNum = 0
148. **while** linkNum < len(SongsLinks):
149. **try**:
150. albom.append(GetSong(SongsLinks[linkNum]))
151. linkNum += 1
152. **except**:
153. **global** faledLinks
154. faledLinks.append(SongsLinks[linkNum])
155. linkNum += 1
156. **return** albom
158. #Возвращает все файлы в директории
159. **def** InFileAll(directory):
160. **return** list(filter(**lambda** x: x.endswith('txt'), os.listdir(directory)))
162. #Возвращает список ссылок на альбомы по имени файла и директории
163. **def** AlbomLinksCreator(name, directory):
164. **return** [re.sub(r'\n', '', link) **for** link **in** open(directory+'/'+name, 'r', encoding = 'utf-8')]
166. #Догружает в базу песни по списку ссылок на альбомы, имени файла и директории
167. **def** BaseCreator(AlbomLinks, outDir, name, directoryNum):
168. flag = 1
169. allBase = open(outDir[directoryNum]+'allbase.txt', 'w', encoding = 'utf-8')
170. **for** link **in** AlbomLinks:
171. **print**('Loading album ', flag, ' of ', len(AlbomLinks), sep = '', end = '\t')
172. albom = AlbomMaker(CreateSongLinks(link))
173. personalBase = open(outDir[directoryNum]+'/out'+name, 'a', encoding = 'utf-8')
174. **for** song **in** albom:
175. **print**('-', end = '')
176. personalBase.write(song[0]+'\n'+song[1]+'\n')
177. **for** lines **in** song[2]:
178. personalBase.write(lines+'\n')
179. allBase.write(lines+'\n')
180. personalBase.write('\n\n')
181. allBase.write('\n\n')
182. flag += 1
183. **print**('\tDone')
185. #Запускает создание базы по заданной директории и входящим файлам
186. **def** BaseDownloader(infiles, inDir, outDir, directoryNum):
187. **global** faledLinks
188. Number = 1
189. **for** name **in** infiles:
190. **print**(Number, 'of', len(infiles),name, sep = ' ', end = '\n')
191. BaseCreator(AlbomLinksCreator(name, inDir[directoryNum]), outDir, name, directoryNum)
192. **print**("--- %s seconds ---", (time.time() - startTime), end = '\n\n')
193. Number += 1
194. faled = open(outDir[directoryNum]+'/faled.txt', 'w', encoding = 'utf-8')
195. **for** i **in** faledLinks:
196. faled.write(i+'\n')
197. falelLinks = []
198. text.insert('end', 'Base downloaded\n')
200. #Убирает повторы и приводит базу к виду списка русских слов в заданной директории
201. **def** Cleaner(baseDir, outDir, style):
202. base = FileReader(outDir[style],'allbase.txt')
203. result = [re.sub(r'припев', '', song) **for** song **in** [song.lower() **for** song **in** [re.sub(r'[,\'\"\-«:»()—]', '', song) **for** song **in**
204. [' '.join(song.split()) **for** song **in** [re.sub(r'\n', '', song) **for** song **in**
205. [re.sub(r'\u2005', ' ', song) **for** song **in** list(set(' '.join(base).split('\n \n')))]]]] **if** len(re.findall(r'[a-zA-Z]+', song)) == 0]]
206. FileWriter(baseDir[style], 'clean.txt', result)
207. text.insert('end', outDir[style]+'\tcleaned\n')
209. #Дописывает потеряшек
210. **def** Refresher(directoryNum, notInBase, refresh):
211. BaseDownloader(notInBase, inDirectory, outDirectory, directoryNum)
212. Cleaner(baseDirectory, outDirectory, outDirectory[directoryNum])
213. RefreshButton()
215. #Создает базу Лемм
216. **def** MorphCorpras(directory):
217. **if** os.path.exists(directory + 'lemmas.txt') != True:
218. text.insert('end', directory + 'Lemmas in progres...\n')
219. lines = FileReader(directory, 'clean.txt')
220. line = re.sub(r'[!?.]+', '', re.sub(r'\n', ' ', ''.join(lines))).split()
221. morph = pymorphy2.MorphAnalyzer()
222. corpra = [morph.parse(word)[0].normal\_form **for** word **in** line **if** len(word) > 0 **and** word **not** **in** ['пр','припев','куплет']]
223. FileWriter(directory, 'lemmas.txt', corpra)
224. text.insert('end', 'Done...\n')
226. '''''-------------------------------------------- П О Д С Ч Е Т  Ч А С Т О Т Н О С Т И -----------------------------------'''
228. **def** int\_r(num):
229. **return** int(num + (0.5 **if** num > 0 **else** -0.5))
231. **def** SentenceSplitter(text):
232. split\_regex = re.compile(r'[.|!|?|…]')
233. **return** list(filter(**lambda** t: t, [t.strip() **for** t **in** split\_regex.split(text)]))
235. **def** AverageReducedFrequency(word, n, corpra, L):
236. f = len(n)
237. d = [n[p + 1] - n[p] **for** p **in** range(len(n) - 1)]
238. d.append(n[0] + len(corpra) - n[len(n) - 1])
239. segmentLen = int\_r(L/f)
240. summa = 0
241. **for** i **in** range(len(d)):
242. summa += min(d[i],segmentLen)
243. #print(word, round(summa/segmentLen, 3), sep = '\t')
244. **return** (word, round(summa/segmentLen, 3))
246. # Делает частотные списки по директории
247. **def** ARFCounter(directory):
248. **if** os.path.exists(directory + 'arf.txt') != True:
249. corpra = [re.sub(r'\n', '', word) **for** word **in** FileReader(directory, 'lemmas.txt')]
250. words = {word: [] **for** word **in** list(nltk.FreqDist(corpra).keys())}
251. **for** i **in** range(len(corpra)):
252. words[corpra[i]].append(i + 1)
253. L = len(corpra)
254. result = [AverageReducedFrequency(word, words[word], corpra, L) **for** word **in** words]
255. result.sort(key = **lambda** x: x[1])
256. result = [str(pair[0])+'\t'+str(pair[1]) **for** pair **in** result[::-1]]
257. FileWriter(directory, 'arf.txt', result)
259. # Делает частотные списки для первоначальной базы
260. **def** ARFMaker():
261. thread1 = Thread(target = ARFCounter, args = (baseDirectory[1],))
262. thread2 = Thread(target = ARFCounter, args = (baseDirectory[0],))
263. thread1.start()
264. thread2.start()
265. thread1.join()
266. thread2.join()
268. # Создаёт скипграммы по заданной директории
269. **def** SkipGramms(inDir, outDir):
270. **for** directory **in** range(len(inDir)):
271. **if** os.path.exists(outDir[directory]+'skipgram.txt') != True:
272. base = FileReader(inDir[directory], 'allbase.txt')
273. corpra = SentenceSplitter(' '.join(list(set([line4 **for** line4 **in** [re.sub(r'[,«\-»:—()]', '', line3) **for** line3 **in**
274. [re.sub(r'\u2005', ' ', line2) **for** line2 **in**
275. base]] **if** len(re.findall(r'[a-z]+', line4)) == 0 **and** len(line4) > 0]))).lower())
276. result = []
277. **for** line **in** corpra:
278. skip = list(nltk.skipgrams(line.split(),2,1))
279. **for** skipi **in** skip:
280. result.append(skipi)
281. freq = nltk.FreqDist(result)
282. l = len(result)
283. freqDist = [(i,freq[i]/l\*1000000) **for** i **in** list(freq.keys())]
284. freqDist.sort(key = **lambda** x: x[1])
285. freqDist = freqDist[::-1]
286. result = [str(freqDist[i][0])+'\t'+str(freqDist[i][1]) **for** i **in** range(len(freqDist))]
287. FileWriter(outDir[directory], 'skipgram.txt', result)
289. **def** Ngrams(inDir, outDir):
290. **for** directory **in** range(len(inDir)):
291. files = os.listdir(outDir[directory])
292. **for** n **in** range(2,6):
293. **if** str(n)+'ngrams.txt' **not** **in** files:
294. base = FileReader(inDir[directory], 'allbase.txt')
295. lines = [line **for** line **in** [re.sub(r'\u2005', ' ', song) **for** song **in** ''.join(base).split('\n\n')] **if** len(re.findall(r'[a-zA-Z]+', line)) == 0]
296. **for** p **in** range(2,6):
297. ngramList = []
298. **for** song **in** lines:
299. **for** ng **in** list(nltk.ngrams(song,p)):
300. ngramList.append(ng)
301. l = len(ngramList)
302. freq = nltk.FreqDist(ngramList)
303. freqDist = [(n,freq[n]/ l \* 1000000) **for** n **in** list(freq.keys())]
304. freqDist.sort(key = **lambda** x: x[1])
305. freqDist = freqDist[::-1]
306. result = [str(freqDist[i][0])+'\t'+str(freqDist[i][1]) **for** i **in** range(len(freqDist))]
307. FileWriter(outDir[directory], str(p)+'ngrams.txt', result)

310. # Принимает два список из двух файлов и печатает словарь "слово":[попса, рэп]
311. **def** Vectors(filename, inDir, outDir):
312. f = [pos.split('\t') **for** pos **in** [re.sub(r'\n', '', line) **for** line **in** FileReader(inDir[0], filename)]]
313. allDict = {a[0]:float(a[1]) **for** a **in** f}
314. **for** i **in** range(1, len(inDir)):
315. f2 = [pos.split('\t') **for** pos **in** [re.sub(r'\n', '', line) **for** line **in** FileReader(inDir[i], filename)]]
316. dictNew = {a[0]:float(a[1]) **for** a **in** f2}
317. **for** word **in** allDict:
318. **if** word **in** dictNew:
319. allDict[word] = [allDict[word], dictNew[word]]
320. **else**:
321. allDict[word] = [allDict[word], 0]
322. **for** word **in** dictNew:
323. **if** word **not** **in** allDict:
324. allDict[word] = [0, dictNew[word]]
325. result = [word+'\t'+'\t\t'.join(map(str, allDict[word])) **for** word **in** list(allDict.keys())]
326. FileWriter(outDir, 'res'+filename, result)

329. '''''--------------------------------------------- Р А С С Т О Я Н И Я -------------------------------------------'''
331. **def** Distances(resultDir):
332. files = list(filter(**lambda** x: x.startswith('res'), os.listdir(resultDir)))
333. **for** file **in** files:
334. lines = FileReader(resultDir, file)
335. s = [[],[]]
336. **for** line **in** lines:
337. **for** i **in** range(1,3):
338. s[i - 1].append(float(line.split()[-i]))
339. with open(resultDir+'Final.txt', 'a', encoding = 'utf-8') as out:
340. out.write(file[3:-4]+'\n'+'косинусное расстояние'+'\t\t'+str(distance.cosine(s[0], s[1]))+'\n')
341. out.write('манхэттенское расстояние'+'\t'+str(distance.cityblock(s[0], s[1]))+'\n')
342. out.write('евклидово расстояние'+'\t\t'+str(distance.euclidean(s[0], s[1]))+'\n')
344. **def** KeyWords(baseDir, resultDir):
345. fwords = []
346. **for** directory **in** baseDir:
347. words = FileReader(directory, 'arf.txt')
348. **if** len(words) > 5000:
349. fwords.append(words[:5001])
350. **else**:
351. fwords.append(words)
352. fwords[0] = [pos.split() **for** pos **in** fwords[0]]
353. fwords[1] = [pos.split() **for** pos **in** fwords[1]]
354. allwords = {fwords[0][i][0]:[float(fwords[0][i][1])] **for** i **in** range(len(fwords[0]))}
355. rapwords = {fwords[1][i][0]:float(fwords[1][i][1]) **for** i **in** range(len(fwords[1]))}
356. **for** word **in** allwords:
357. **if** word **in** rapwords:
358. allwords[word].append(rapwords[word])
359. **else**:
360. allwords[word].append(0)
361. **for** word **in** rapwords:
362. **if** word **not** **in** allwords:
363. allwords[word] = [0, rapwords[word]]
364. **for** n **in** [1,10,100,1000]:
365. popkeywords = [(word,(allwords[word][0] + n) / (allwords[word][1] + n)) **for** word **in** list(allwords.keys())]
366. rapkeywords = [(word,(allwords[word][1] + n) / (allwords[word][0] + n)) **for** word **in** list(allwords.keys())]
367. summ = 0
368. **for** N **in** range(5000):
369. summ += max(popkeywords[N][1], rapkeywords[N][1])
370. dist = summ/1000
371. with open(resultDir+'KeyWordFinal.txt', 'a',encoding = 'utf-8') as f:
372. f.write("При n равном "+str(n)+":\t"+str(dist)+'\n')
373. **if** os.path.exists(baseDir[0]+str(n)+'keywords.txt') != True:
374. popkeywords.sort(key = **lambda** x: x[1])
375. popkeywords = popkeywords[::-1]
376. result = [word[0]+'\t'+str(word[1]) **for** word **in** popkeywords]
377. FileWriter(baseDir[0], str(n)+'keywords.txt', result)
378. **if** os.path.exists(baseDir[1]+str(n)+'keywords.txt') != True:
379. rapkeywords.sort(key = **lambda** x: x[1])
380. rapkeywords = rapkeywords[::-1]
381. result = [word[0]+'\t'+str(word[1]) **for** word **in** rapkeywords]
382. FileWriter(baseDir[1], str(n)+'keywords.txt', result)
384. '''''------------------------------------------------ Д Е Л А Е Т  Ч А С Т О Т Ы  Д Л Я  П Е Р В О Н А Ч А Л Ь Н О Й  Б А З Ы ----------------------------'''
386. **if** os.path.exists(baseDirectory[0]+'arf.txt') != True:
387. ARFMaker()
388. **if** os.path.exists(baseDirectory[0]+'skipgram.txt') != True:
389. SkipGramms(outDirectory, baseDirectory)
390. **if** os.path.exists(resultDirectory+'resarf.txt') != True:
391. Vectors('arf.txt', baseDirectory, resultDirectory)
392. **if** os.path.exists(resultDirectory+'resskipgram.txt') != True:
393. Vectors('skipgram.txt', baseDirectory, resultDirectory)
394. Ngrams(outDirectory, baseDirectory)
395. **for** i **in** range(2,6):
396. **if** os.path.exists(resultDirectory+'res'+str(i)+'ngrams.txt') != True:
397. Vectors(str(i)+'ngrams.txt', baseDirectory, resultDirectory)
398. **if** os.path.exists(resultDirectory+'KeyWordFinal.txt') != True:
399. KeyWords(baseDirectory, resultDirectory)
400. **if** os.path.exists(resultDirectory+'Final.txt') != True:
401. Distances(resultDirectory)

404. '''''------------------------------------------------------ П Р О В Е Р К А  Н О В Ы Х  Т Е К С Т О В -----------------------------'''
406. **def** StyleFinder(event, resultLabel,resultLabel2, final):
407. newDirectory = ['/'.join(str(dirEntry.get()).split('\\'))+'/']
408. dirEntry.delete(0, END)
409. os.mkdir(newDirectory[0]+'Output/')
410. os.mkdir(newDirectory[0]+'Bases/')
411. outNewDirectory = [newDirectory[0] + 'Output/']
412. baseNewDirectory = [newDirectory[0] + 'Bases/']
413. BaseDownloader(InFileAll(newDirectory[0]), newDirectory, outNewDirectory, 0)
414. Cleaner(baseNewDirectory, outNewDirectory, 0)
415. MorphCorpras(baseNewDirectory[0])
416. ARFCounter(baseNewDirectory[0])
417. Ngrams(outNewDirectory, baseNewDirectory)
418. SkipGramms(outNewDirectory, baseNewDirectory)
419. baseStyleCheck = [[baseNewDirectory[0], baseDirectory[0]],[baseNewDirectory[0], baseDirectory[1]]]
420. res = ['ResultPop/', 'ResultRap/']
421. **for** i **in** range(len(baseStyleCheck)):
422. os.mkdir(newDirectory[0]+res[i])
423. resultNewDirectory = newDirectory[0] + res[i]
424. Vectors('arf.txt', baseStyleCheck[i], resultNewDirectory)
425. Vectors('skipgram.txt', baseStyleCheck[i], resultNewDirectory)
426. **for** n **in** range(2,6):
427. Vectors(str(n)+'ngrams.txt', baseStyleCheck[i], resultNewDirectory)
428. KeyWords(baseStyleCheck[i], resultNewDirectory)
429. Distances(resultNewDirectory)
430. ResultWriter(newDirectory[0], resultLabel,resultLabel2, final)
432. **def** StyleFinder2(event, textTaker, song, resultLabel,resultLabel2, final):
433. with open(r'C:/Users/epish/Desktop/учеба/ДИПЛОМ/text/Output/allbase.txt', 'w', encoding = 'utf-8') as inf:
434. inf.write(song.get('1.0', END))
435. textTaker.destroy()
436. newDirectory = ['C:/Users/epish/Desktop/учеба/ДИПЛОМ/text/']
437. os.mkdir(newDirectory[0]+'Bases/')
438. outNewDirectory = [newDirectory[0] + 'Output/']
439. baseNewDirectory = [newDirectory[0] + 'Bases/']
440. Cleaner(baseNewDirectory, outNewDirectory, 0)
441. MorphCorpras(baseNewDirectory[0])
442. ARFCounter(baseNewDirectory[0])
443. Ngrams(outNewDirectory, baseNewDirectory)
444. SkipGramms(outNewDirectory, baseNewDirectory)
445. baseStyleCheck = [[baseNewDirectory[0], baseDirectory[0]],[baseNewDirectory[0], baseDirectory[1]]]
446. res = ['ResultPop/', 'ResultRap/']
447. **for** i **in** range(len(baseStyleCheck)):
448. os.mkdir(newDirectory[0]+res[i])
449. resultNewDirectory = newDirectory[0] + res[i]
450. Vectors('arf.txt', baseStyleCheck[i], resultNewDirectory)
451. Vectors('skipgram.txt', baseStyleCheck[i], resultNewDirectory)
452. **for** n **in** range(2,6):
453. Vectors(str(n)+'ngrams.txt', baseStyleCheck[i], resultNewDirectory)
454. KeyWords(baseStyleCheck[i], resultNewDirectory)
455. Distances(resultNewDirectory)
456. ResultWriter(newDirectory[0], resultLabel,resultLabel2, final)
458. **def** StyleFinder3(event, resultLabel,resultLabel2, final):
459. newDirectory = ['/'.join(str(dirEntry2.get()).split('\\'))+'/']
460. dirEntry2.delete(0, END)
461. os.mkdir(newDirectory[0]+'Bases/')
462. outNewDirectory = [newDirectory[0] + 'Output/']
463. baseNewDirectory = [newDirectory[0] + 'Bases/']
464. Cleaner(baseNewDirectory, outNewDirectory, 0)
465. MorphCorpras(baseNewDirectory[0])
466. ARFCounter(baseNewDirectory[0])
467. Ngrams(outNewDirectory, baseNewDirectory)
468. SkipGramms(outNewDirectory, baseNewDirectory)
469. baseStyleCheck = [[baseNewDirectory[0], baseDirectory[0]],[baseNewDirectory[0], baseDirectory[1]]]
470. res = ['ResultPop/', 'ResultRap/']
471. **for** i **in** range(len(baseStyleCheck)):
472. os.mkdir(newDirectory[0]+res[i])
473. resultNewDirectory = newDirectory[0] + res[i]
474. Vectors('arf.txt', baseStyleCheck[i], resultNewDirectory)
475. Vectors('skipgram.txt', baseStyleCheck[i], resultNewDirectory)
476. **for** n **in** range(2,6):
477. Vectors(str(n)+'ngrams.txt', baseStyleCheck[i], resultNewDirectory)
478. KeyWords(baseStyleCheck[i], resultNewDirectory)
479. Distances(resultNewDirectory)
480. ResultWriter(newDirectory[0], resultLabel,resultLabel2, final)
482. **def** TextTaker():
483. textTaker = Tk()
484. textTaker.title("Songs text")
485. textTaker['bg'] = 'black'
486. sizex = 403
487. sizey = 703
488. posx  = 200
489. posy  = 120
490. textTaker.wm\_geometry("%dx%d+%d+%d" % (sizex, sizey, posx, posy))
491. song = Text(textTaker, width = 50, height = 40)
492. song.place(x = 0, y = 0)
493. checkButton2 = Button(textTaker, text = 'Find out style!')
494. checkButton2.bind('<Button-1>', **lambda** event: StyleFinder2(event, textTaker, song, resultLabel, resultLabel2, final))
495. checkButton2.place(x = 150, y = 660)
496. textTaker.mainloop()
498. **def** ResultWriter(resDir, resultLabel, resultLabel2, final):
499. res = ['ResultPop/', 'ResultRap/']
500. result = PrettyTable()
501. result.field\_names = ["Test", "Param", "Pop", "Rap"]
502. #resDir = 'C:/Users/epish/Desktop/учеба/ДИПЛОМ/КлассикаТексты/'
503. resPop = FileReader(resDir+res[0], 'Final.txt')
504. resRap = FileReader(resDir+res[1], 'Final.txt')
505. lines = [['','cos'], ['2gram','man'], ['','ev'],
506. ['','cos'], ['3gram','man'], ['','ev'],
507. ['','cos'], ['4gram','man'], ['','ev'],
508. ['','cos'], ['5gram','man'], ['','ev'],
509. ['','cos'], ['arf','man'], ['','ev'],
510. ['','cos'], ['skipgrams','man'], ['','ev'],]
511. resPop = [line.split()[-1] **for** line **in** resPop **if** len(line.split()) > 1]
512. resRap = [line.split()[-1] **for** line **in** resRap **if** len(line.split()) > 1]
513. **for** i **in** range(len(lines)):
514. lines[i]. append(round(float(resPop[i]), 3))
515. lines[i]. append(round(float(resRap[i]),3))
516. result.add\_row(lines[i])
517. resultLabel.insert('end', 'FreqDist parameters\n')
518. resultLabel.insert('end', result)
519. resPop2 = FileReader(resDir+res[0], 'KeyWordFinal.txt')
520. resRap2 = FileReader(resDir+res[1], 'KeyWordFinal.txt')
521. result = PrettyTable()
522. result.field\_names = ["N", "Pop", "Rap"]
523. lines = [['   1'],['  10'],[' 100'],['1000']]
524. resPop2 = [line.split()[-1] **for** line **in** resPop]
525. resRap2 = [line.split()[-1] **for** line **in** resRap]
526. **for** i **in** range(len(lines)):
527. lines[i]. append(round(float(resPop2[i]), 3))
528. lines[i]. append(round(float(resRap2[i]),3))
529. result.add\_row(lines[i])
530. resultLabel2.insert('end', 'KeyWords\n')
531. resultLabel2.insert('end', result)
532. popMax = 0
533. resPop += resPop2
534. resRap += resRap2
535. **for** i **in** range(len(resPop)):
536. **if** resPop[i] < resRap[i]:
537. popMax += 1
538. res = [['Pop',popMax/len(resPop)\*100], ['Rap', (len(resPop) - popMax)/len(resPop)\*100]]
539. res.sort(key = **lambda** x: x[1])
540. final.insert('end', 'Style result:\n')
541. **for** i **in** res[::-1]:
542. final.insert('end','\t' + i[0] + '\t' + str(round(i[1]))+'%' + '\n')



547. '''''--------------------------------------------------- О К Н О  "П Р И Л О Ж Е Н И Я" -------------------------------------------'''
549. root = Tk()
550. root.title("Base options")
551. sizex = 800
552. sizey = 700
553. posx  = 100
554. posy  = 100
555. root.wm\_geometry("%dx%d+%d+%d" % (sizex, sizey, posx, posy))
556. root['bg'] = 'black'
557. mainMenu = Menu()
559. text = Text(width = 100, height = 10, bg = 'black', fg = 'white', highlightthickness = 0, bd = 0)
560. text.pack(side = BOTTOM)
562. file\_menu = Menu()
563. file\_menu.add\_command(label="Create", command = CreateButton)
564. file\_menu.add\_command(label="Clean", command = CleanButton)
565. file\_menu.add\_command(label="Refresh", command = RefreshButton)
566. file\_menu.add\_command(label="Lemmas", command = LemmaCorpButton)
567. file\_menu.add\_separator()
568. file\_menu.add\_command(label="Exit", command = CloseButton)
570. edit\_menu = Menu()
571. edit\_menu.add\_command(label = 'Refresh', command = FullInformation)
573. mainMenu.add\_cascade(label="Base", menu = file\_menu)
574. mainMenu.add\_cascade(label="Information", menu = edit\_menu)

577. root.config(menu = mainMenu)
579. information = Label(root, bg = 'black', fg = 'white', font = 'Times 14', height = 6, width = 14, anchor = NW)
580. **try**:
581. information['text'] = FullInformation()
582. information.place(x = 10, y = 10)
583. **except**:
584. information['text'] = 'No base found'
585. information.place(x = 10, y = 10)
587. newDirectory = StringVar()
588. classDirectory  = StringVar()
590. dirTaker = Label(root, text = 'Enter directory:', bg = 'black', fg = 'white', font = 'Times 14', height = 1, anchor = NW)
591. dirTaker.place(x = 150, y = 10)
593. dirEntry = Entry(root, textvariable = newDirectory, width = 54)
594. dirEntry.place(x = 300, y = 14)
596. checkButton = Button(root, text = 'Find out style!')
597. checkButton.place(x = 650, y = 13)
599. dirTaker2 = Label(root, text = 'Enter songs\' texts:', bg = 'black', fg = 'white', font = 'Times 14', height = 1, anchor = NW)
600. dirTaker2.place(x = 150, y = 50)
602. dirEntry2 = Button(root, width = 46, height = 1, bg = 'white', command = TextTaker)
603. dirEntry2.place(x = 300, y = 52)
605. checkButton2 = Button(root, text = 'Find out style!', command = TextTaker)
606. checkButton2.place(x = 650, y = 53)
608. dirTaker2 = Label(root, text = 'Enter allbase dir:', bg = 'black', fg = 'white', font = 'Times 14', height = 1, anchor = NW)
609. dirTaker2.place(x = 150, y = 90)
611. dirEntry2 = Entry(root, textvariable = classDirectory, width = 54)
612. dirEntry2.place(x = 300, y = 94)
614. checkButton3 = Button(root, text = 'Find out style!')
615. checkButton3.place(x = 650, y = 93)
617. resultLabel = Text(root, height = 23, width = 50, bg = 'black', fg = 'white', highlightthickness = 0, bd = 0)
618. resultLabel.place(x = 40, y = 155)
620. resultLabel2 = Text(root, height = 10, width = 40, bg = 'black', fg = 'white', highlightthickness = 0, bd = 0)
621. resultLabel2.place(x = 440, y = 155)
623. final = Text(root, height = 10, width = 20, bg = 'black', fg = 'white', bd = 0)
624. final.place(x = 440, y = 333)
626. checkButton.bind('<Button-1>', **lambda** event: StyleFinder(event, resultLabel,resultLabel2, final))
627. checkButton3.bind('<Button-1>', **lambda** event: StyleFinder3(event, resultLabel,resultLabel2, final))
629. root.mainloop()
630. Автоматическое определение авторства
631. **import** os
632. **import** pandas as pd
633. **import** nltk
634. **from** scipy.spatial **import** distance
635. **import** math
636. **import** re
637. **import** pymorphy2
638. **import** time
639. **from** operator **import** itemgetter
641. OutDir = 'C:/Users/epish/Desktop/учеба/ДИПЛОМ/authTest/'
642. Dir = 'C:/Users/epish/Desktop/учеба/ДИПЛОМ/TEST/Unprocessed/'
644. CleanDirk = 'C:/Users/epish/Desktop/учеба/ДИПЛОМ/authTest/Known/Clean/'
645. LemmaDirk = 'C:/Users/epish/Desktop/учеба/ДИПЛОМ/authTest/Known/Lemmas/'
646. NgramDirk = 'C:/Users/epish/Desktop/учеба/ДИПЛОМ/authTest/Known/Ngrams/'
647. ARFDirk = 'C:/Users/epish/Desktop/учеба/ДИПЛОМ/authTest/Known/ARF/'
648. SkipDirk = 'C:/Users/epish/Desktop/учеба/ДИПЛОМ/authTest/Known/Skip/'
650. CleanDiru = 'C:/Users/epish/Desktop/учеба/ДИПЛОМ/authTest/Unknown/Clean/'
651. LemmaDiru = 'C:/Users/epish/Desktop/учеба/ДИПЛОМ/authTest/Unknown/Lemmas/'
652. NgramDiru = 'C:/Users/epish/Desktop/учеба/ДИПЛОМ/authTest/Unknown/Ngrams/'
653. ARFDiru = 'C:/Users/epish/Desktop/учеба/ДИПЛОМ/authTest/Unknown/ARF/'
654. SkipDiru = 'C:/Users/epish/Desktop/учеба/ДИПЛОМ/authTest/Unknown/Skip/'

657. **def** SentenceSplitter(text):
658. split\_regex = re.compile(r'[.|!|?|…]')
659. **return** list(filter(**lambda** t: t, [t.strip() **for** t **in** split\_regex.split(text)]))
661. **def** FilesPrepare():
662. pages = os.listdir(Dir)
663. **for** page **in** pages:
664. files = os.listdir(Dir+page)
665. **for** i **in** range(len(files)):
666. with open(Dir+page+'/'+files[i], 'r', encoding = 'utf-8') as f:
667. lines = [line **for** line **in** f]
668. lines = ''.join(''.join(lines).split('\n\n')[1:]).split('\n')
669. **if** i%8 == 0:
670. with open(OutDir+'Known/'+page+'.txt', 'a', encoding = 'utf-8') as f:
671. **for** line **in** lines:
672. f.write(line+'\n')
673. f.write('\n')
674. **if** i%8 == 1:
675. with open(OutDir+'Unknown/'+page+'.txt', 'a', encoding = 'utf-8') as f:
676. **for** line **in** lines:
677. f.write(line+'\n')
678. f.write('\n')
680. **def** FileReader(directory, name):
681. with open(directory+name, 'r', encoding = "utf-8") as file:
682. **return** [line **for** line **in** file **if** line != '\n']
684. **def** Clean(word):
685. files = list(filter(**lambda** x: x.endswith('txt'), os.listdir(OutDir+word)))
686. **for** file **in** files:
687. base = FileReader(OutDir+word, file)
688. base = re.sub(r'\n', ' ', re.sub(r'[,\'\"\-«:»()—…\/?!\.―–0-9;:\*]', ' ', ' '.join(base))).lower()
689. with open('C:/Users/epish/Desktop/учеба/ДИПЛОМ/authTest/'+ word + 'Clean/'+file, 'w', encoding = 'utf-8') as f:
690. f.write(base)
692. **def** SkipGramms(word):
693. files = list(filter(**lambda** x: x.endswith('txt'), os.listdir(OutDir+word)))
694. **for** file **in** files:
695. corpra = SentenceSplitter(re.sub(r'[,\'\"\-«:»()—\–―–0-9;:\*]',' ',' '.join(FileReader(OutDir+word, file)).lower()))
696. result = []
697. **for** line **in** corpra:
698. skip = list(nltk.skipgrams(line.split(),2,1))
699. **for** skipi **in** skip:
700. result.append(skipi)
701. freq = nltk.FreqDist(result)
702. l = len(result)
703. freqDist = [(i,freq[i]/l\*1000000) **for** i **in** list(freq.keys())]
704. freqDist.sort(key = **lambda** x: x[1])
705. freqDist = freqDist[::-1]
706. result = [str(freqDist[i][0])+'\t'+str(freqDist[i][1]) **for** i **in** range(len(freqDist))]
707. with open('C:/Users/epish/Desktop/учеба/ДИПЛОМ/authTest/' + word + 'Skip/'+file, 'w', encoding = 'utf-8') as f:
708. **for** skip **in** result:
709. f.write(skip+'\n')
711. **def** Lemmatisation(word):
712. files = os.listdir(CleanDir)
713. morph = pymorphy2.MorphAnalyzer()
714. **for** file **in** files:
715. **print**(file)
716. lines = ''.join(FileReader(CleanDir, file)).split(' ')
717. base = [morph.parse(word)[0].normal\_form **for** word **in** lines **if** len(word) > 0]
718. with open('C:/Users/epish/Desktop/учеба/ДИПЛОМ/authTest/' + word + 'Lemmas/'+file, 'w', encoding = 'utf-8') as f:
719. **for** lemm **in** base:
720. f.write(lemm+' ')
722. **def** int\_r(num):
723. **return** int(num + (0.5 **if** num > 0 **else** -0.5))
725. **def** AverageReducedFrequency(word, n, corpra, L):
726. f = len(n)
727. d = [n[p + 1] - n[p] **for** p **in** range(len(n) - 1)]
728. d.append(n[0] + len(corpra) - n[len(n) - 1])
729. segmentLen = int\_r(L/f)
730. summa = 0
731. **for** i **in** range(len(d)):
732. summa += min(d[i],segmentLen)
733. **return** (word, round(summa/segmentLen, 3))
735. # Делает частотные списки по директории
736. **def** ARFCounter(word):
737. files = os.listdir(LemmaDir)
738. **for** file **in** files:
739. corpra = [arf **for** arf **in** ''.join(FileReader(LemmaDir, file)).split() **if** len(word) > 0]
740. words = {arf: [] **for** arf **in** list(nltk.FreqDist(corpra).keys())}
741. **for** i **in** range(len(corpra)):
742. words[corpra[i]].append(i + 1)
743. L = len(corpra)
744. result = [AverageReducedFrequency(arf, words[arf], corpra, L) **for** arf **in** words]
745. result.sort(key = **lambda** x: x[1])
746. result = [str(pair[0])+'\t'+str(pair[1]) **for** pair **in** result[::-1]]
747. with open('C:/Users/epish/Desktop/учеба/ДИПЛОМ/authTest/' + word + 'ARF/'+file, 'w', encoding = 'utf-8') as f:
748. **for** arf **in** result:
749. f.write(arf+'\n')
751. **def** Ngrams(word):
752. files = list(filter(**lambda** x: x.endswith('txt'), os.listdir(OutDir+word)))
753. **for** file **in** files:
754. lines = FileReader(OutDir+word, file)
755. ngramList = []
756. **for** song **in** lines:
757. **for** ng **in** list(nltk.ngrams(song,4)):
758. ngramList.append(ng)
759. l = len(ngramList)
760. freq = nltk.FreqDist(ngramList)
761. freqDist = [(n,freq[n]/ l \* 1000000) **for** n **in** list(freq.keys())]
762. freqDist.sort(key = **lambda** x: x[1])
763. freqDist = freqDist[::-1]
764. result = [str(freqDist[i][0])+'\t'+str(freqDist[i][1]) **for** i **in** range(len(freqDist))]
765. with open('C:/Users/epish/Desktop/учеба/ДИПЛОМ/authTest/' + word + 'Ngrams/'+file, 'w', encoding = 'utf-8') as f:
766. **for** n **in** result:
767. f.write(n+'\n')
768. **def** Known():
769. word = 'Known/'
770. Clean(word)
771. SkipGramms(word)
772. Lemmatisation(word)
773. ARFCounter(word)
774. Ngrams(word)
776. **def** Unknown():
777. word = 'Unknown/'
778. Clean(word)
779. SkipGramms(word)
780. Lemmatisation(word)
781. ARFCounter(word)
782. Ngrams(word)
784. #FilesPrepare()
785. #Known()
786. #Unknown()
788. **def** VectorCreator(knownfile, unknownfile, knowndir, unknowndir):
789. start\_time\_def = time.time()
790. p = [knownfile, 'un'+unknownfile]
791. with open(knowndir+knownfile, 'r', encoding = 'utf-8') as f:
792. lines = {line.split('\t')[0]:float(line.split('\t')[1]) **for** line **in** f}
793. with open(unknowndir+unknownfile, 'r', encoding = 'utf-8') as f:
794. unlines = {line.split('\t')[0]:float(line.split('\t')[1]) **for** line **in** f}
795. known = dict.fromkeys(list(lines)+list(unlines))
796. unknown = {}
797. **for** word **in** known:
798. **if** word **in** list(lines):
799. known[word] = lines[word]
800. **else**:
801. known[word] = 0
802. **if** word **in** list(unlines):
803. unknown[word] = unlines[word]
804. **else**:
805. unknown[word] = 0
806. vectors = [list(known.values()), list(unknown.values())]
807. **print** (distance.cosine(vectors[0], vectors[1]), time.time() - start\_time\_def, sep = '\t')
808. **return** distance.cosine(vectors[0], vectors[1])
810. **def** SecondType(dist, art):
811. vectors = []
812. **for** i **in** range(len(dist) - 1):
813. sqrt = dist[i][0]\*\*2 + dist[i][1]\*\*2 + dist[i][2]\*\*2
814. vectors.append(sqrt)
815. **return** art[min(enumerate(vectors), key=itemgetter(1))[0]][:-4]
817. **def** FTables():
818. knownp = [NgramDirk , ARFDirk , SkipDirk]
819. unknownp = [NgramDiru , ARFDiru , SkipDiru]
820. art = os.listdir(knownp[0])
821. restable = pd.DataFrame(columns = ['Result', 'Time'], index = [artist[6:-4] **for** artist **in** art])
822. **print**(restable)
823. start\_time = time.time()
824. **for** unknown **in** range(len(art)):
825. res = [[]]
826. **for** known **in** range(len(art)):
827. **print**(art[unknown][:-4], ' vs ', art[known][:-4])
828. **for** i **in** range(len(knownp)):
829. res[known].append(VectorCreator(art[known], art[unknown], knownp[i], unknownp[i]))
830. res.append([])
831. name = SecondType(res, art)
832. restable.loc[art[unknown][6:-4], 'Result'] = name
833. restable.loc[art[unknown][6:-4], 'Time'] = time.time() - start\_time
834. **print**(restable)
835. restable.to\_excel('C:/Users/epish/Desktop/учеба/ДИПЛОМ/authTest/text.xlsx')
837. FTables()
838. Два алгоритма определения авторства
839. **def** FirstType(dist):
840. param = ['key1', 'key1000', 'ngram4cos', 'skipcos']
841. dist = {param[i]:dist[i] **for** i **in** range(4)}
842. tables = TableReader()
843. res = []
844. minIndex = ''
845. **for** p **in** param:
846. minim = 10000000000
847. **for** d **in** range(len(tables[p][:20])):
848. delta = []
849. **for** i **in** range(len(list(tables[p].iloc[d])[1:])):
850. **if** math.isnan(list(tables[p].iloc[d])[1:][i]):
851. delta.append(abs(float(dist[p][i]) - 0))
852. **else**:
853. delta.append(abs(float(dist[p][i]) - list(tables[p].iloc[d])[1:][i]))
854. deltaSum = sum(delta)
855. #print(deltaSum)
856. **if** deltaSum < minim:
857. minim = deltaSum
858. minIndex = list(tables[p].iloc[d])[0]
859. res.append((p, minIndex, minim))
860. dist = dict.fromkeys(list(tables['key1']['Unnamed: 0'][:20]), 0)
861. **print**(res)
862. **for** param **in** res:
863. **for** i **in** range(20):
864. **if** math.isnan(tables[param[0]].iloc[i][param[1]]):
865. dist[tables['key1']['Unnamed: 0'][i]] += 0
866. **else**:
867. dist[tables['key1']['Unnamed: 0'][i]] += tables[param[0]].iloc[i][param[1]]
868. p = [(art,dist[art]) **for** art **in** list(dist)]
869. p.sort(key = **lambda** x: x[1])
870. **print**("First param " + p[0][0])
872. **def** SecondType(dist):
873. res = [[]]
874. **for** i **in** range(20):
875. **for** j **in** range(4):
876. res[i].append(float(dist[j][i]))
877. res.append([])
878. vectors = []
879. **for** i **in** range(len(res) - 1):
880. mass = int\_r(max(res[i]))
881. sqrt = (res[i][0]/mass)\*\*2 + (res[i][1]/mass)\*\*2 + res[i][2]\*\*2 + res[i][3]\*\*2
882. vectors.append(sqrt)
883. auth = ['2rbina2rista', 'basta', 'brb', 'eldzhey', 'face', 'feduk',
884. 'gnoiny', 'guf', 'husky', 'kasta', 'korzh', 'krovostok', 'kunteynir', 'lizer', 'lsp', 'morgenshtern', 'noizemc', 'skriptonite', 'timati', 'xleb']
885. **print**("Second param " + auth[min(enumerate(vectors), key=itemgetter(1))[0]])

## Таблицы

1. ARFCos

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **2rbina2rista** | **basta** | **brb** | **eldzhey** | **face** | **feduk** | **gnoiny** | **guf** | **husky** | **kasta** | **korzh** | **krovostok** | **kunteynir** | **lizer** | **lsp** | **morgenshtern** | **noizemc** | **skriptonite** | **timati** | **xleb** |
| **2rbina2rista** |  | 0,15965294 | 0,252562061 | 0,244645044 | 0,244526483 | 0,21927528 | 0,127934132 | 0,172775827 | 0,160275977 | 0,120885078 | 0,145316789 | 0,126439212 | 0,131613642 | 0,235009322 | 0,165517402 | 0,284481014 | 0,123347163 | 0,151097482 | 0,247734696 | 0,236939728 |
| **basta** | 0,15965294 |  | 0,132478924 | 0,086771224 | 0,058625476 | 0,083416373 | 0,039545488 | 0,072441419 | 0,078226458 | 0,08981714 | 0,056298125 | 0,117735894 | 0,122271206 | 0,060541559 | 0,047935291 | 0,093675032 | 0,090196117 | 0,043796756 | 0,05321429 | 0,143322533 |
| **brb** | 0,252562061 | 0,132478924 |  | 0,154722176 | 0,113855995 | 0,211985547 | 0,139355453 | 0,149231035 | 0,140015956 | 0,21231655 | 0,182218258 | 0,223897563 | 0,197524121 | 0,145906921 | 0,179695439 | 0,137470218 | 0,221830715 | 0,156542578 | 0,156048265 | 0,221132098 |
| **eldzhey** | 0,244645044 | 0,086771224 | 0,154722176 |  | 0,081584348 | 0,138426945 | 0,107240004 | 0,107751943 | 0,134297129 | 0,1834948 | 0,162680005 | 0,189134055 | 0,196318409 | 0,11301304 | 0,100593446 | 0,085615618 | 0,181613031 | 0,112587604 | 0,126239234 | 0,180582289 |
| **face** | 0,244526483 | 0,058625476 | 0,113855995 | 0,081584348 |  | 0,15639087 | 0,102998829 | 0,111841154 | 0,109401112 | 0,200077015 | 0,131870748 | 0,229515448 | 0,211051488 | 0,034260454 | 0,113528525 | 0,042909225 | 0,203366467 | 0,096333158 | 0,058331118 | 0,190473293 |
| **feduk** | 0,21927528 | 0,083416373 | 0,211985547 | 0,138426945 | 0,15639087 |  | 0,10182264 | 0,138823772 | 0,169778554 | 0,114292188 | 0,131818801 | 0,146709211 | 0,187300272 | 0,174453676 | 0,078882038 | 0,199738892 | 0,118918085 | 0,119048113 | 0,132086715 | 0,165153809 |
| **gnoiny** | 0,127934132 | 0,039545488 | 0,139355453 | 0,107240004 | 0,102998829 | 0,10182264 |  | 0,054172003 | 0,075205069 | 0,05184489 | 0,060536812 | 0,062870347 | 0,07806098 | 0,113410177 | 0,048597245 | 0,128624628 | 0,053838016 | 0,041977519 | 0,110167939 | 0,133397124 |
| **guf** | 0,172775827 | 0,072441419 | 0,149231035 | 0,107751943 | 0,111841154 | 0,138823772 | 0,054172003 |  | 0,122828876 | 0,082137614 | 0,089896203 | 0,086604988 | 0,130393284 | 0,113782235 | 0,061985599 | 0,099784143 | 0,08391847 | 0,084372298 | 0,146423866 | 0,129264473 |
| **husky** | 0,160275977 | 0,078226458 | 0,140015956 | 0,134297129 | 0,109401112 | 0,169778554 | 0,075205069 | 0,122828876 |  | 0,134740984 | 0,10938419 | 0,128272722 | 0,095109508 | 0,119417759 | 0,125786284 | 0,146210931 | 0,135968387 | 0,083811095 | 0,140332084 | 0,197216921 |
| **kasta** | 0,120885078 | 0,08981714 | 0,21231655 | 0,1834948 | 0,200077015 | 0,114292188 | 0,05184489 | 0,082137614 | 0,134740984 |  | 0,06215316 | 0,038309241 | 0,08348879 | 0,198213446 | 0,066436624 | 0,236930498 | 0,020912577 | 0,066707656 | 0,176853138 | 0,138644268 |
| **korzh** | 0,145316789 | 0,056298125 | 0,182218258 | 0,162680005 | 0,131870748 | 0,131818801 | 0,060536812 | 0,089896203 | 0,10938419 | 0,06215316 |  | 0,095644807 | 0,117727582 | 0,102150319 | 0,083147693 | 0,166977111 | 0,051970801 | 0,049929088 | 0,111159711 | 0,154323956 |
| **krovostok** | 0,126439212 | 0,117735894 | 0,223897563 | 0,189134055 | 0,229515448 | 0,146709211 | 0,062870347 | 0,086604988 | 0,128272722 | 0,038309241 | 0,095644807 |  | 0,070450153 | 0,230084558 | 0,086891685 | 0,245079557 | 0,036305159 | 0,095708245 | 0,225033982 | 0,168438817 |
| **kunteynir** | 0,131613642 | 0,122271206 | 0,197524121 | 0,196318409 | 0,211051488 | 0,187300272 | 0,07806098 | 0,130393284 | 0,095109508 | 0,08348879 | 0,117727582 | 0,070450153 |  | 0,230489701 | 0,138825802 | 0,253372613 | 0,08950615 | 0,100471111 | 0,216774153 | 0,200759787 |
| **lizer** | 0,235009322 | 0,060541559 | 0,145906921 | 0,11301304 | 0,034260454 | 0,174453676 | 0,113410177 | 0,113782235 | 0,119417759 | 0,198213446 | 0,102150319 | 0,230084558 | 0,230489701 |  | 0,114600956 | 0,048415302 | 0,189899214 | 0,096242521 | 0,071575611 | 0,203380298 |
| **lsp** | 0,165517402 | 0,047935291 | 0,179695439 | 0,100593446 | 0,113528525 | 0,078882038 | 0,048597245 | 0,061985599 | 0,125786284 | 0,066436624 | 0,083147693 | 0,086891685 | 0,138825802 | 0,114600956 |  | 0,124652583 | 0,067040794 | 0,0627449 | 0,113181329 | 0,137358077 |
| **morgenshtern** | 0,284481014 | 0,093675032 | 0,137470218 | 0,085615618 | 0,042909225 | 0,199738892 | 0,128624628 | 0,099784143 | 0,146210931 | 0,236930498 | 0,166977111 | 0,245079557 | 0,253372613 | 0,048415302 | 0,124652583 |  | 0,232608748 | 0,138383895 | 0,099779976 | 0,21131307 |
| **noizemc** | 0,123347163 | 0,090196117 | 0,221830715 | 0,181613031 | 0,203366467 | 0,118918085 | 0,053838016 | 0,08391847 | 0,135968387 | 0,020912577 | 0,051970801 | 0,036305159 | 0,08950615 | 0,189899214 | 0,067040794 | 0,232608748 |  | 0,0717805 | 0,189281148 | 0,143941435 |
| **skriptonite** | 0,151097482 | 0,043796756 | 0,156542578 | 0,112587604 | 0,096333158 | 0,119048113 | 0,041977519 | 0,084372298 | 0,083811095 | 0,066707656 | 0,049929088 | 0,095708245 | 0,100471111 | 0,096242521 | 0,0627449 | 0,138383895 | 0,0717805 |  | 0,098651575 | 0,152230275 |
| **timati** | 0,247734696 | 0,05321429 | 0,156048265 | 0,126239234 | 0,058331118 | 0,132086715 | 0,110167939 | 0,146423866 | 0,140332084 | 0,176853138 | 0,111159711 | 0,225033982 | 0,216774153 | 0,071575611 | 0,113181329 | 0,099779976 | 0,189281148 | 0,098651575 |  | 0,206032673 |
| **xleb** | 0,236939728 | 0,143322533 | 0,221132098 | 0,180582289 | 0,190473293 | 0,165153809 | 0,133397124 | 0,129264473 | 0,197216921 | 0,138644268 | 0,154323956 | 0,168438817 | 0,200759787 | 0,203380298 | 0,137358077 | 0,21131307 | 0,143941435 | 0,152230275 | 0,206032673 |  |

1. Максимальные расстояния между исполнителями

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | **2rbina2rista** | **basta** | **brb** | **eldzhey** | **face** | **feduk** | **gnoiny** | **guf** | **husky** | **kasta** | **korzh** | **krovostok** | **kunteynir** | **lizer** | **lsp** | **morgenshtern** | **noizemc** | **skriptonite** | **timati** | **xleb** |
| **arfcos** | morgenshtern | 2rbina2rista | 2rbina2rista | 2rbina2rista | 2rbina2rista | 2rbina2rista | brb | 2rbina2rista | xleb | morgenshtern | brb | morgenshtern | morgenshtern | 2rbina2rista | brb | 2rbina2rista | morgenshtern | brb | 2rbina2rista | 2rbina2rista |
| **arfcity** | noizemc | noizemc | noizemc | noizemc | noizemc | noizemc | noizemc | noizemc | noizemc | noizemc | noizemc | noizemc | noizemc | noizemc | noizemc | noizemc | brb | noizemc | noizemc | noizemc |
| **arfeu** | noizemc | noizemc | noizemc | noizemc | noizemc | noizemc | noizemc | noizemc | noizemc | brb | noizemc | noizemc | noizemc | brb | noizemc | noizemc | brb | noizemc | noizemc | noizemc |
| **key1** | noizemc | brb | noizemc | noizemc | noizemc | noizemc | noizemc | noizemc | noizemc | brb | brb | brb | noizemc | brb | noizemc | noizemc | brb | brb | noizemc | noizemc |
| **key10** | lizer | brb | lizer | noizemc | noizemc | lizer | noizemc | noizemc | noizemc | brb | brb | brb | noizemc | brb | noizemc | noizemc | brb | brb | noizemc | lizer |
| **key100** | lizer | brb | lizer | lizer | noizemc | lizer | noizemc | noizemc | noizemc | brb | brb | noizemc | noizemc | brb | noizemc | noizemc | brb | brb | noizemc | lizer |
| **key1000** | lizer | 2rbina2rista | lizer | lizer | lizer | lizer | 2rbina2rista | noizemc | noizemc | brb | 2rbina2rista | 2rbina2rista | lizer | 2rbina2rista | noizemc | brb | 2rbina2rista | 2rbina2rista | korzh | lizer |
| **ngram2cos** | xleb | xleb | xleb | xleb | xleb | 2rbina2rista | xleb | xleb | xleb | xleb | xleb | xleb | xleb | xleb | xleb | xleb | xleb | xleb | xleb | 2rbina2rista |
| **ngram2city** | xleb | xleb | xleb | xleb | 2rbina2rista | 2rbina2rista | xleb | xleb | xleb | xleb | xleb | xleb | xleb | xleb | xleb | 2rbina2rista | xleb | xleb | xleb | 2rbina2rista |
| **ngram2eu** | xleb | xleb | xleb | xleb | xleb | xleb | xleb | xleb | xleb | xleb | xleb | xleb | xleb | xleb | xleb | xleb | xleb | xleb | xleb | 2rbina2rista |
| **ngram3cos** | xleb | xleb | xleb | xleb | xleb | xleb | xleb | xleb | xleb | xleb | xleb | xleb | xleb | xleb | xleb | xleb | xleb | xleb | xleb | 2rbina2rista |
| **ngram3city** | xleb | xleb | 2rbina2rista | 2rbina2rista | 2rbina2rista | 2rbina2rista | xleb | 2rbina2rista | xleb | xleb | brb | xleb | xleb | 2rbina2rista | xleb | 2rbina2rista | xleb | 2rbina2rista | 2rbina2rista | 2rbina2rista |
| **ngram3eu** | xleb | xleb | xleb | xleb | xleb | xleb | xleb | xleb | xleb | xleb | xleb | xleb | xleb | xleb | xleb | xleb | xleb | xleb | xleb | 2rbina2rista |
| **ngram4cos** | xleb | xleb | xleb | xleb | xleb | 2rbina2rista | xleb | 2rbina2rista | xleb | xleb | xleb | xleb | xleb | xleb | xleb | xleb | xleb | xleb | 2rbina2rista | 2rbina2rista |
| **ngram4city** | xleb | brb | 2rbina2rista | 2rbina2rista | 2rbina2rista | 2rbina2rista | xleb | 2rbina2rista | xleb | brb | brb | xleb | brb | 2rbina2rista | brb | 2rbina2rista | brb | 2rbina2rista | brb | 2rbina2rista |
| **ngram4eu** | xleb | xleb | xleb | xleb | xleb | xleb | xleb | xleb | xleb | xleb | xleb | xleb | xleb | xleb | xleb | xleb | xleb | xleb | xleb | brb |
| **ngram5cos** | xleb | xleb | 2rbina2rista | 2rbina2rista | 2rbina2rista | 2rbina2rista | xleb | 2rbina2rista | xleb | xleb | 2rbina2rista | xleb | xleb | 2rbina2rista | 2rbina2rista | 2rbina2rista | xleb | 2rbina2rista | 2rbina2rista | 2rbina2rista |
| **ngram5city** | xleb | 2rbina2rista | 2rbina2rista | 2rbina2rista | 2rbina2rista | 2rbina2rista | brb | 2rbina2rista | xleb | brb | brb | brb | 2rbina2rista | 2rbina2rista | brb | 2rbina2rista | brb | 2rbina2rista | brb | 2rbina2rista |
| **ngram5eu** | xleb | xleb | xleb | xleb | xleb | xleb | xleb | xleb | xleb | xleb | xleb | xleb | xleb | xleb | xleb | xleb | xleb | xleb | xleb | brb |
| **skipcos** | xleb | xleb | xleb | xleb | xleb | xleb | xleb | xleb | xleb | xleb | xleb | xleb | xleb | xleb | xleb | xleb | xleb | xleb | xleb | brb |
| **skipcity** | brb | brb | feduk | 2rbina2rista | 2rbina2rista | brb | brb | brb | brb | brb | 2rbina2rista | brb | brb | 2rbina2rista | brb | 2rbina2rista | brb | 2rbina2rista | 2rbina2rista | brb |
| **skipeu** | xleb | xleb | xleb | xleb | xleb | xleb | xleb | xleb | xleb | xleb | xleb | xleb | xleb | xleb | xleb | xleb | xleb | xleb | xleb | feduk |

1. Минимальные расстояния между авторами

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **2rbina2rista** | **basta** | **brb** | **eldzhey** | **face** | **feduk** | **gnoiny** | **guf** | **husky** | **kasta** | **korzh** | **krovostok** | **kunteynir** | **lizer** | **lsp** | **morgenshtern** | **noizemc** | **skriptonite** | **timati** | **xleb** |
| **arfcos** | kasta | gnoiny | face | face | lizer | lsp | basta | gnoiny | gnoiny | noizemc | skriptonite | noizemc | krovostok | face | basta | face | kasta | gnoiny | basta | guf |
| **arfcity** | brb | face | feduk | feduk | timati | brb | face | timati | timati | korzh | face | guf | feduk | face | timati | face | kasta | morgenshtern | feduk | feduk |
| **arfeu** | brb | gnoiny | feduk | feduk | morgenshtern | brb | krovostok | lsp | guf | skriptonite | basta | gnoiny | eldzhey | basta | guf | face | kasta | basta | eldzhey | feduk |
| **key1** | brb | kasta | feduk | kunteynir | kunteynir | eldzhey | krovostok | kunteynir | kunteynir | krovostok | kasta | kasta | guf | basta | kunteynir | kunteynir | kasta | krovostok | kunteynir | kunteynir |
| **key10** | brb | krovostok | feduk | kunteynir | kunteynir | eldzhey | krovostok | krovostok | kunteynir | krovostok | kasta | gnoiny | timati | basta | guf | gnoiny | kasta | krovostok | kunteynir | kunteynir |
| **key100** | brb | gnoiny | feduk | kunteynir | gnoiny | brb | basta | gnoiny | kunteynir | basta | krovostok | basta | eldzhey | basta | gnoiny | gnoiny | kasta | krovostok | kunteynir | eldzhey |
| **key1000** | brb | gnoiny | eldzhey | brb | gnoiny | brb | basta | face | face | krovostok | basta | basta | brb | basta | face | basta | basta | basta | eldzhey | eldzhey |
| **ngram2cos** | kasta | kasta | skriptonite | noizemc | skriptonite | kasta | krovostok | krovostok | krovostok | krovostok | kasta | kasta | krovostok | face | noizemc | face | kasta | kasta | basta | face |
| **ngram2city** | krovostok | kasta | kasta | noizemc | lizer | kasta | krovostok | krovostok | krovostok | noizemc | basta | kasta | krovostok | korzh | kasta | face | kasta | kasta | basta | skriptonite |
| **ngram2eu** | husky | kasta | gnoiny | noizemc | skriptonite | kasta | krovostok | krovostok | gnoiny | krovostok | kasta | kasta | krovostok | face | noizemc | face | kasta | kasta | basta | skriptonite |
| **ngram3cos** | krovostok | noizemc | kasta | noizemc | lizer | kasta | krovostok | noizemc | krovostok | noizemc | noizemc | noizemc | krovostok | basta | noizemc | face | kasta | kasta | basta | guf |
| **ngram3city** | kasta | noizemc | kasta | noizemc | lizer | basta | kasta | noizemc | krovostok | noizemc | basta | noizemc | krovostok | korzh | noizemc | face | kasta | kasta | basta | kasta |
| **ngram3eu** | krovostok | noizemc | kasta | noizemc | skriptonite | kasta | krovostok | noizemc | krovostok | noizemc | noizemc | noizemc | krovostok | basta | noizemc | face | kasta | kasta | basta | kasta |
| **ngram4cos** | noizemc | noizemc | noizemc | noizemc | lizer | noizemc | noizemc | noizemc | noizemc | noizemc | noizemc | noizemc | krovostok | basta | noizemc | lizer | kasta | noizemc | basta | noizemc |
| **ngram4city** | noizemc | noizemc | basta | noizemc | lizer | basta | noizemc | noizemc | noizemc | noizemc | basta | noizemc | krovostok | korzh | noizemc | lizer | kasta | noizemc | basta | korzh |
| **ngram4eu** | noizemc | noizemc | noizemc | noizemc | noizemc | kasta | noizemc | noizemc | noizemc | noizemc | noizemc | noizemc | krovostok | basta | noizemc | face | kasta | noizemc | basta | noizemc |
| **ngram5cos** | noizemc | noizemc | basta | noizemc | lizer | noizemc | noizemc | noizemc | noizemc | noizemc | noizemc | noizemc | noizemc | noizemc | noizemc | lizer | guf | noizemc | basta | noizemc |
| **ngram5city** | basta | noizemc | lizer | lizer | lizer | lizer | noizemc | noizemc | noizemc | noizemc | basta | noizemc | krovostok | korzh | noizemc | lizer | kasta | noizemc | basta | lizer |
| **ngram5eu** | noizemc | noizemc | noizemc | noizemc | noizemc | noizemc | noizemc | noizemc | noizemc | noizemc | noizemc | noizemc | noizemc | noizemc | noizemc | noizemc | kasta | noizemc | basta | noizemc |
| **skipcos** | skriptonite | lizer | lizer | lizer | lizer | noizemc | lizer | lizer | lizer | noizemc | lizer | noizemc | noizemc | skriptonite | lizer | lizer | lizer | lizer | basta | noizemc |
| **skipcity** | lizer | lizer | morgenshtern | lizer | lizer | skriptonite | lizer | lizer | lizer | noizemc | lizer | noizemc | lizer | face | lizer | lizer | kasta | lizer | lizer | lizer |
| **skipeu** | noizemc | noizemc | noizemc | noizemc | noizemc | noizemc | noizemc | noizemc | noizemc | noizemc | noizemc | noizemc | noizemc | noizemc | noizemc | noizemc | krovostok | noizemc | noizemc | noizemc |

1. Все таблицы находятся в [облаке](https://yadi.sk/d/q9HWCHukOAOT9g) в файле (https://yadi.sk/d/q9HWCHukOAOT9g) [↑](#footnote-ref-1)