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

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 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

$$ARF = \frac{1}{v}\sum_{i=1}^{f} min(d_i, v),$$

где f – число вхождений слова x в корпус, d_i – расстояние от $(i - 1)$-го вхождения до i-го вхождения x, а v – длина сегмента, которая рассчитывается по формуле $v = \frac{L}{f}$, 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

$$w_{ipm}(k) = \frac{w(k)}{L} \cdot 1000000$$

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

$$similarity = \cos(\alpha) = \frac{\vec{A} \cdot \vec{B}}{\|\vec{A}\|\|\vec{B}\|} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n}(A_i)^2} \times \sqrt{\sum_{i=1}^{n}(B_i)^2}}$$

где $\vec{A} \cdot \vec{B}$ – скалярное произведение векторов, а $\|\vec{A}\|$ – длина вектора $\vec{A}$, α – угол между векторами $\vec{A}$ и $\vec{B}$.

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, $\cos(\alpha)$ 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

$$d(p,q) = \|p - q\|_1 = \sum_{i=1}^{n} |p_i - q_i|$$

где p = (p_1, p_2, p_3,…, p_n), q = (q_1, q_2, q_3,…, q_n).

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:

$$d(p,q) = \sqrt{\sum_{i=1}^{n} (p_i - q_i)^2}$$

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 2018a]. 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

$$k = \frac{f_A(w) + n}{f_B(w) + n}$$

где A – так называемый фокусный корпус, а именно корпус, для которого мы находим ключевые слова, B – референциальный корпус, с которым мы сравниваем фокусный, $f_P(w)$ – 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:

$$k(w) = max\left(\frac{f_A(w) + n}{f_B(w) + n}, \frac{f_B(w) + n}{f_A(w) + n}\right)$$

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] 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

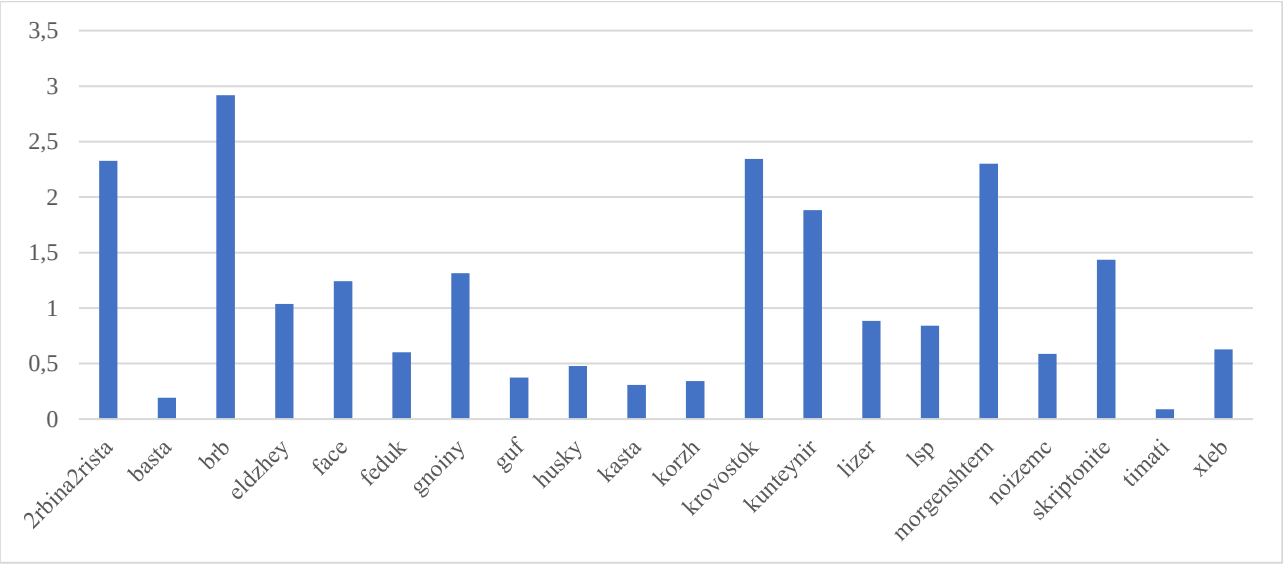
На основании таблиц расстояний между авторами используя леммы, однозначных результатов не получилось. Некоторые исполнители попарно отличаются достаточно сильно, например, Morgenshtern и 2rbina2rista имеют самое большое расстояние между собой. В целом это объясняется тем, что 2rbina2rista

[1] Все таблицы находятся в облаке в файле (https://yadi.sk/d/q9HWCHukOAOT9g)

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

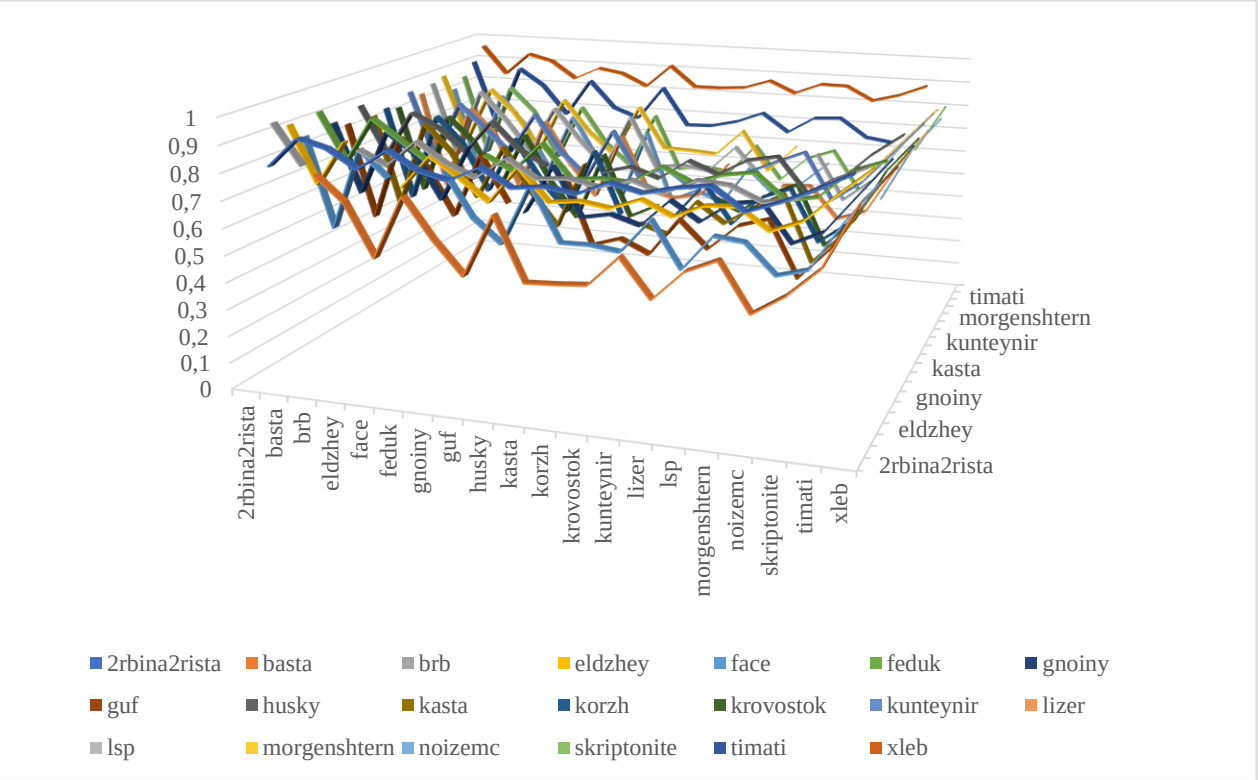
Данный параметр не показал однозначных результатов, поскольку объемы корпусов каждого исполнителя маленькие, частоты слов небольшие, и, соответственно, ранги и расстояния по ключевым словам получаются маленькие.

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.

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.

Изображение 5. Облако слов Поп стиль Изображение 6 Облако слов Рэп стиль

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:

2.1 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:

	Исп_1	Исп_2	Исп_3	Исп_4	Исп_5
Исп_1	0	$x_{1,2}$	$x_{1,3}$	$x_{1,4}$	$x_{1,5}$
Исп_2	$x_{2,1}$	0	$x_{2,3}$	$x_{2,4}$	$x_{2,5}$
Исп_3	$x_{3,1}$	$x_{3,2}$	0	$x_{3,4}$	$x_{3,5}$
Исп_4	$x_{4,1}$	$x_{4,2}$	$x_{4,3}$	0	$x_{4,5}$
Исп_5	$x_{5,1}$	$x_{5,2}$	$x_{5,3}$	$x_{5,4}$	0

Строка расстояний от искомого текста до каждого из них: $dist = (y_1, y_2, y_3, y_4, y_5)$

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

$$(|y_1 - x_{i,1}| \quad |y_2 - x_{i,2}| \quad |y_3 - x_{i,3}| \quad |y_4 - x_{i,4}| \quad |y_5 - x_{i,5}|)$$

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

2.2 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.

3 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

$$L = x_1^2 + x_2^2 + x_3^2 + x_4^2$$

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 (x_1, y_1, z_1), (x_2, y_2, z_2), причем $x_1 < x_2, y_1 < y_2$, а $z_1 > z_2$, 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-граммы, 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, Thomas Proisl, Fotis Jannidis, Isabella Reger, Steffen Pielström, Christof Schöch, Thorsten Vitt – 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

```
import requests
import re
from bs4 import BeautifulSoup
import time
import warnings
import os
from tkinter import *
import nltk
import pymorphy2
from threading import Thread
from nltk import ngrams, skipgrams
from scipy.spatial import distance
from prettytable import PrettyTable

startTime = time.time()
faledLinks = []
inDirectory = ['C:/Users/epish/Desktop/учеба/ДИПЛОМ/ПопТексты/Input/', 'C:/Users/epish/Desktop/учеба/ДИПЛОМ/РэпТексты/Input/']
outDirectory = ['C:/Users/epish/Desktop/учеба/ДИПЛОМ/ПопТексты/Output/', 'C:/Users/epish/Desktop/учеба/ДИПЛОМ/РэпТексты/Output/']
baseDirectory = ['C:/Users/epish/Desktop/учеба/ДИПЛОМ/ПопТексты/Bases/', 'C:/Users/epish/Desktop/учеба/ДИПЛОМ/РэпТексты/Bases/']
resultDirectory = 'C:/Users/epish/Desktop/учеба/ДИПЛОМ/Results/'

'''---------------------------------------------- К Н О П О Ч К И ----------------------------------------------'''

#Вызывает создание базы с нуля
def CreateButton():
    for i in range(len(outDirectory)):
        if len(os.listdir(outDirectory[i])) == 0:
            BaseDownloader(InFileAll(inDirectory[i]), inDirectory, outDirectory, i)
    text.insert('end', 'All bases are created\n')
    text.see("end")

# Вызывает создание базы лемм
def LemmaCorpButton():
    for i in range(len(baseDirectory)):
        if os.path.exists(baseDirectory[i]+'lemmas.txt') != True:
            MorphCorpras(baseDirectory[i])
    text.insert('end', 'All lemma bases are created\n')
    text.see("end")

#Вызывает очистку базы
def CleanButton(*mode):
    if len(mode) == 0:
        for i in range(len(baseDirectory)):
```

```
            if os.path.exists(baseDirectory[i]+'cleanbase.txt'):
                Cleaner(baseDirectory, outDirectory, i)
        text.insert('end', 'All bases are cleaned\n')
        text.see("end")

#Вызывает догрузку файлов в первоначальную базу
def RefreshButton():
    for directoryNum in range(len(inDirectory)):
        notInBase = [name for name in [title for title in list(filter(lambda x: x.endswith('txt'
), os.listdir(inDirectory[directoryNum])))] 
                    if 'out'+name not in list(filter(lambda x: x.startswith('out'), os.listdir(outDire
ctory[directoryNum])))]
        text.insert('end', inDirectory[directoryNum]+':\n')
        if len(notInBase) != 0:
            flag = 'worked'
            for name in notInBase:
                text.insert('end', name + ' is not uploaded\n')
                text.see('end')
            refresh = Tk()
            refresh.title('Refresher')
            refresh.geometry('200x100+300+200')
            btnRefresh = Label(refresh, text = 'Upload new files?')
            btnYes = Button(refresh, text = 'yes', width = 10, command = lambda: Refresher(
directoryNum,notInBase, refresh))
            btnNo = Button(refresh, text = 'close', width = 10, command = refresh.destroy)
            btnYes.place(x = 10, y = 50)
            btnNo.place(x = 110, y = 50)
            btnRefresh.pack()
            refresh.mainloop()
        else:
            try:
                if flag == 'worked':
                    refresh.destroy()
                    text.insert('end','All files are uploaded\n')
            except UnboundLocalError:
                text.insert('end','All files are uploaded\n')

#Закрывает окно приложения
def CloseButton():
    root.destroy()

'''---------------------------------------------- И Н Ф О Р М А Ц И Я ----------------------------
---------------------'''

def FullInformation():
    global inDirectory, outDirectory, baseDirectory
    informationText = 'Base information'
    artists = len(list(filter(lambda x: x.startswith('out'), os.listdir(outDirectory[0])))+
                  list(filter(lambda x: x.startswith('out'), os.listdir(outDirectory[1]))))
    informationText += '\nArtists:\t'+str(artists)
    alboms = 0
    artists = 0
```

```
92.    songs = 0
93.    words = 0
94.    lemmas = 0
95.    for i in range(len(inDirectory)):
96.        for artist in InFileAll(inDirectory[i]):
97.            alboms += len(FileReader(inDirectory[i],artist))
98.        try:
99.            songs += len(FileReader(baseDirectory[i], 'clean.txt'))
100.                except FileNotFoundError:
101.                    text.insert('end', outDirectory[i]+' Need cleaning!\n')
102.                try:
103.                    word = FileReader(baseDirectory[i], 'lemmas.txt')
104.                    words += len(word)
105.                    lemmas += len(list(set(word)))
106.                except:
107.                    text.insert('end', baseDirectory[i]+' Need lemmatization!\n')
108.
109.            informationText += '\nAlbums:\t' + str(alboms) + '\nSongs:\t' + str(songs) + '\n
    Words:\t' + str(words) + '\nLemmas:\t' + str(lemmas)
110.            information['text'] = informationText
111.
112.        '''---------------------------------------- Р А Б О Т А  С  Б А З О Й ----------------
    --------------------------------'''
113.
114.        def FileReader(directory, filename):
115.            with open(directory+filename, 'r', encoding = 'utf-8') as inFile:
116.                return [line for line in inFile]
117.
118.        def FileWriter(directory, filename, corpra):
119.            with open(directory+filename, 'w', encoding = 'utf-8') as outFile:
120.                for line in corpra:
121.                    outFile.write(line+'\n')
122.
123.        #Возвращает html код страницы по ссылке
124.        def GetHTML(link):
125.            #time.sleep(1)
126.            return requests.get(link).text
127.
128.        #Возвращает всё о песне по ссылке
129.        def GetSong(link):
130.            soup = BeautifulSoup(GetHTML(link), features="lxml")
131.            Artist = soup.find('div', {'class':'song_album-
    info'}).find('a', {'class': 'song_album-info-artist'}).text
132.            SongTitle = soup.find('div', {'class':'header_with_cover_art'}).find('h1').text
133.            AlbumTitle = soup.find('div', {'class':'song_album-info'}).find('a').get('title')
134.            SongText = [re.sub(r'\xa0', ' ', line2) for line2 in
135.                    [re.sub(r'\[.*\]', '', line) for line in soup.find('div', {'class':'lyrics'}).text.s
    plit('\n') if len(line) > 0]
136.                    if len(line2) > 0]
137.            return (AlbumTitle, SongTitle, SongText)
138.
139.        #Возвращает список всех песен в альбоме по ссылке на альбом
```

```
140.        def CreateSongLinks(AlbomLink):
141.            return [re.sub(r'<a href=\"(.*)\"', r'\1', link) for link in re.findall(r'<a href=\"[a-
    zA-Z 2:/\.\-]+\-lyrics\"', GetHTML(AlbomLink))]
142.
143.        #Возвращает альбом по списку ссылок на каждую песню
144.        def AlbomMaker(SongsLinks):
145.            albom = []
146.            linkNum = 0
147.            while linkNum < len(SongsLinks):
148.                try:
149.                    albom.append(GetSong(SongsLinks[linkNum]))
150.                    linkNum += 1
151.                except:
152.                    global faledLinks
153.                    faledLinks.append(SongsLinks[linkNum])
154.                    linkNum += 1
155.            return albom
156.
157.        #Возвращает все файлы в директории
158.        def InFileAll(directory):
159.            return list(filter(lambda x: x.endswith('txt'), os.listdir(directory)))
160.
161.        #Возвращает список ссылок на альбомы по имени файла и директории
162.        def AlbomLinksCreator(name, directory):
163.            return [re.sub(r'\n', '', link) for link in open(directory+'/'+name, 'r', encoding = '
    utf-8')]
164.
165.        #Догружает в базу песни по списку ссылок на альбомы, имени файла и дире
    ктории
166.        def BaseCreator(AlbomLinks, outDir, name, directoryNum):
167.            flag = 1
168.            allBase = open(outDir[directoryNum]+'allbase.txt', 'w', encoding = 'utf-8')
169.            for link in AlbomLinks:
170.                print('Loading album ', flag, ' of ', len(AlbomLinks), sep = '', end = '\t')
171.                albom = AlbomMaker(CreateSongLinks(link))
172.                personalBase = open(outDir[directoryNum]+'/out'+name, 'a', encoding = 'utf-
    8')
173.                for song in albom:
174.                    print('-', end = '')
175.                    personalBase.write(song[0]+'\n'+song[1]+'\n')
176.                    for lines in song[2]:
177.                        personalBase.write(lines+'\n')
178.                        allBase.write(lines+'\n')
179.                    personalBase.write('\n\n')
180.                    allBase.write('\n\n')
181.                flag += 1
182.                print('\tDone')
183.
184.        #Запускает создание базы по заданной директории и входящим файлам
185.        def BaseDownloader(infiles, inDir, outDir, directoryNum):
186.            global faledLinks
187.            Number = 1
```

```
188.             for name in infiles:
189.                 print(Number, 'of', len(infiles),name, sep = ' ', end = '\n')
190.                 BaseCreator(AlbomLinksCreator(name, inDir[directoryNum]), outDir, name,
    directoryNum)
191.                 print("--- %s seconds ---", (time.time() - startTime), end = '\n\n')
192.                 Number += 1
193.             faled = open(outDir[directoryNum]+'/faled.txt', 'w', encoding = 'utf-8')
194.             for i in faledLinks:
195.                 faled.write(i+'\n')
196.             falelLinks = []
197.             text.insert('end', 'Base downloaded\n')
198.
199.         #Убирает повторы и приводит базу к виду списка русских слов в заданной д
    иректории
200.         def Cleaner(baseDir, outDir, style):
201.             base = FileReader(outDir[style],'allbase.txt')
202.             result = [re.sub(r'припев', '', song) for song in [song.lower() for song in [re.sub
    (r'[,\'\"\-«:»()—]', '', song) for song in
203.                     [' '.join(song.split()) for song in [re.sub(r'\n', '', song) for song in
204.                                                 [re.sub(r'\u2005', ' ', song) for song in list(set(' '.joi
    n(base).split('\n \n')))]]]] if len(re.findall(r'[a-zA-Z]+', song)) == 0]]
205.             FileWriter(baseDir[style], 'clean.txt', result)
206.             text.insert('end', outDir[style]+'\tcleaned\n')
207.
208.         #Дописывает потеряшек
209.         def Refresher(directoryNum, notInBase, refresh):
210.             BaseDownloader(notInBase, inDirectory, outDirectory, directoryNum)
211.             Cleaner(baseDirectory, outDirectory, outDirectory[directoryNum])
212.             RefreshButton()
213.
214.         #Создает базу Лемм
215.         def MorphCorpras(directory):
216.             if os.path.exists(directory + 'lemmas.txt') != True:
217.                 text.insert('end', directory + 'Lemmas in progres...\n')
218.                 lines = FileReader(directory, 'clean.txt')
219.                 line = re.sub(r'[!?.]+', '', re.sub(r'\n', ' ', ''.join(lines))).split()
220.                 morph = pymorphy2.MorphAnalyzer()
221.                 corpra = [morph.parse(word)[0].normal_form for word in line if len(word) >
    0 and word not in ['пр','припев','куплет']]
222.                 FileWriter(directory, 'lemmas.txt', corpra)
223.                 text.insert('end', 'Done...\n')
224.
225.         """---------------------------------------- П О Д С Ч Е Т  Ч А С Т О Т Н О С Т И -
    ----------------------------------"""
226.
227.         def int_r(num):
228.             return int(num + (0.5 if num > 0 else -0.5))
229.
230.         def SentenceSplitter(text):
231.             split_regex = re.compile(r'[.|!|?|…]')
232.             return list(filter(lambda t: t, [t.strip() for t in split_regex.split(text)]))
233.
```

```
234.         def AverageReducedFrequency(word, n, corpra, L):
235.             f = len(n)
236.             d = [n[p + 1] - n[p] for p in range(len(n) - 1)]
237.             d.append(n[0] + len(corpra) - n[len(n) - 1])
238.             segmentLen = int_r(L/f)
239.             summa = 0
240.             for i in range(len(d)):
241.                 summa += min(d[i],segmentLen)
242.             #print(word, round(summa/segmentLen, 3), sep = '\t')
243.             return (word, round(summa/segmentLen, 3))
244.
245.         # Делает частотные списки по директории
246.         def ARFCounter(directory):
247.             if os.path.exists(directory + 'arf.txt') != True:
248.                 corpra = [re.sub(r'\n', '', word) for word in FileReader(directory, 'lemmas.txt')
    ]
249.                 words = {word: [] for word in list(nltk.FreqDist(corpra).keys())}
250.                 for i in range(len(corpra)):
251.                     words[corpra[i]].append(i + 1)
252.                 L = len(corpra)
253.                 result = [AverageReducedFrequency(word, words[word], corpra, L) for wor
    d in words]
254.                 result.sort(key = lambda x: x[1])
255.                 result = [str(pair[0])+'\t'+str(pair[1]) for pair in result[::-1]]
256.                 FileWriter(directory, 'arf.txt', result)
257.
258.         # Делает частотные списки для первоначальной базы
259.         def ARFMaker():
260.             thread1 = Thread(target = ARFCounter, args = (baseDirectory[1],))
261.             thread2 = Thread(target = ARFCounter, args = (baseDirectory[0],))
262.             thread1.start()
263.             thread2.start()
264.             thread1.join()
265.             thread2.join()
266.
267.         # Создаёт скипграммы по заданной директории
268.         def SkipGramms(inDir, outDir):
269.             for directory in range(len(inDir)):
270.                 if os.path.exists(outDir[directory]+'skipgram.txt') != True:
271.                     base = FileReader(inDir[directory], 'allbase.txt')
272.                     corpra = SentenceSplitter(' '.join(list(set([line4 for line4 in [re.sub(r'[,«\-
    »:—()]', '', line3) for line3 in
273.                                             [re.sub(r'\u2005', ' ', line2) for line2 in
274.                                              base]] if len(re.findall(r'[a-
    z]+', line4)) == 0 and len(line4) > 0])).lower())
275.                     result = []
276.                     for line in corpra:
277.                         skip = list(nltk.skipgrams(line.split(),2,1))
278.                         for skipi in skip:
279.                             result.append(skipi)
280.                     freq = nltk.FreqDist(result)
281.                     l = len(result)
```

```
282.                     freqDist = [(i,freq[i]/l*1000000) for i in list(freq.keys())]
283.                     freqDist.sort(key = lambda x: x[1])
284.                     freqDist = freqDist[::-1]
285.                     result = [str(freqDist[i][0])+'\t'+str(freqDist[i][1]) for i in range(len(freqDi
    st))]
286.                     FileWriter(outDir[directory], 'skipgram.txt', result)
287.
288.         def Ngrams(inDir, outDir):
289.             for directory in range(len(inDir)):
290.                 files = os.listdir(outDir[directory])
291.                 for n in range(2,6):
292.                     if str(n)+'ngrams.txt' not in files:
293.                         base = FileReader(inDir[directory], 'allbase.txt')
294.                         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]
295.                         for p in range(2,6):
296.                             ngramList = []
297.                             for song in lines:
298.                                 for ng in list(nltk.ngrams(song,p)):
299.                                     ngramList.append(ng)
300.                             l = len(ngramList)
301.                             freq = nltk.FreqDist(ngramList)
302.                             freqDist = [(n,freq[n]/ l * 1000000) for n in list(freq.keys())]
303.                             freqDist.sort(key = lambda x: x[1])
304.                             freqDist = freqDist[::-1]
305.                             result = [str(freqDist[i][0])+'\t'+str(freqDist[i][1]) for i in range(len(fr
    eqDist))]
306.                             FileWriter(outDir[directory], str(p)+'ngrams.txt', result)
307.
308.
309.         # Принимает два список из двух файлов и печатает словарь "слово":[попса, р
    эп]
310.         def Vectors(filename, inDir, outDir):
311.             f = [pos.split('\t') for pos in [re.sub(r'\n', '', line) for line in FileReader(inDir[0],
    filename)]]
312.             allDict = {a[0]:float(a[1]) for a in f}
313.             for i in range(1, len(inDir)):
314.                 f2 = [pos.split('\t') for pos in [re.sub(r'\n', '', line) for line in FileReader(inDir
    [i], filename)]]
315.                 dictNew = {a[0]:float(a[1]) for a in f2}
316.                 for word in allDict:
317.                     if word in dictNew:
318.                         allDict[word] = [allDict[word], dictNew[word]]
319.                     else:
320.                         allDict[word] = [allDict[word], 0]
321.                 for word in dictNew:
322.                     if word not in allDict:
323.                         allDict[word] = [0, dictNew[word]]
324.             result = [word+'\t'+'\t\t'.join(map(str, allDict[word])) for word in list(allDict.ke
    ys())]
325.             FileWriter(outDir, 'res'+filename, result)
326.
```

```
327.
328.         """---------------------------------------- Р А С С Т О Я Н И Я ----------------------
    --------------------"""
329.
330.         def Distances(resultDir):
331.             files = list(filter(lambda x: x.startswith('res'), os.listdir(resultDir)))
332.             for file in files:
333.                 lines = FileReader(resultDir, file)
334.                 s = [[],[]]
335.                 for line in lines:
336.                     for i in range(1,3):
337.                         s[i - 1].append(float(line.split()[-i]))
338.                 with open(resultDir+'Final.txt', 'a', encoding = 'utf-8') as out:
339.                     out.write(file[3:-
    4]+'\n'+'косинусное расстояние'+'\t\t'+str(distance.cosine(s[0], s[1]))+'\n')
340.                     out.write('манхэттенское расстояние'+'\t'+str(distance.cityblock(s[0], s[1
    ]))+'\n')
341.                     out.write('евклидово расстояние'+'\t\t'+str(distance.euclidean(s[0], s[1]))
    +'\n')
342.
343.         def KeyWords(baseDir, resultDir):
344.             fwords = []
345.             for directory in baseDir:
346.                 words = FileReader(directory, 'arf.txt')
347.                 if len(words) > 5000:
348.                     fwords.append(words[:5001])
349.                 else:
350.                     fwords.append(words)
351.             fwords[0] = [pos.split() for pos in fwords[0]]
352.             fwords[1] = [pos.split() for pos in fwords[1]]
353.             allwords = {fwords[0][i][0]:[float(fwords[0][i][1])] for i in range(len(fwords[0]
    ))}
354.             rapwords = {fwords[1][i][0]:float(fwords[1][i][1]) for i in range(len(fwords[1])
    )}
355.             for word in allwords:
356.                 if word in rapwords:
357.                     allwords[word].append(rapwords[word])
358.                 else:
359.                     allwords[word].append(0)
360.             for word in rapwords:
361.                 if word not in allwords:
362.                     allwords[word] = [0, rapwords[word]]
363.             for n in [1,10,100,1000]:
364.                 popkeywords = [(word,(allwords[word][0] + n) / (allwords[word][1] + n)) fo
    r word in list(allwords.keys())]
365.                 rapkeywords = [(word,(allwords[word][1] + n) / (allwords[word][0] + n)) for
     word in list(allwords.keys())]
366.                 summ = 0
367.                 for N in range(5000):
368.                     summ += max(popkeywords[N][1], rapkeywords[N][1])
369.                     dist = summ/1000
370.                 with open(resultDir+'KeyWordFinal.txt', 'a',encoding = 'utf-8') as f:
```

```
371.                 f.write("При n равном "+str(n)+":\t"+str(dist)+'\n')
372.             if os.path.exists(baseDir[0]+str(n)+'keywords.txt') != True:
373.                 popkeywords.sort(key = lambda x: x[1])
374.                 popkeywords = popkeywords[::-1]
375.                 result = [word[0]+'\t'+str(word[1]) for word in popkeywords]
376.                 FileWriter(baseDir[0], str(n)+'keywords.txt', result)
377.             if os.path.exists(baseDir[1]+str(n)+'keywords.txt') != True:
378.                 rapkeywords.sort(key = lambda x: x[1])
379.                 rapkeywords = rapkeywords[::-1]
380.                 result = [word[0]+'\t'+str(word[1]) for word in rapkeywords]
381.                 FileWriter(baseDir[1], str(n)+'keywords.txt', result)
382.
383.     """----------------------------------------------
    - 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 ---------
    ----------"""
384.
385.     if os.path.exists(baseDirectory[0]+'arf.txt') != True:
386.         ARFMaker()
387.     if os.path.exists(baseDirectory[0]+'skipgram.txt') != True:
388.         SkipGramms(outDirectory, baseDirectory)
389.     if os.path.exists(resultDirectory+'resarf.txt') != True:
390.         Vectors('arf.txt', baseDirectory, resultDirectory)
391.     if os.path.exists(resultDirectory+'resskipgram.txt') != True:
392.         Vectors('skipgram.txt', baseDirectory, resultDirectory)
393.     Ngrams(outDirectory, baseDirectory)
394.     for i in range(2,6):
395.         if os.path.exists(resultDirectory+'res'+str(i)+'ngrams.txt') != True:
396.             Vectors(str(i)+'ngrams.txt', baseDirectory, resultDirectory)
397.     if os.path.exists(resultDirectory+'KeyWordFinal.txt') != True:
398.         KeyWords(baseDirectory, resultDirectory)
399.     if os.path.exists(resultDirectory+'Final.txt') != True:
400.         Distances(resultDirectory)
401.
402.
403.     """----------------------------------------------------
    - П Р О В Е Р К А  Н О В Ы Х  Т Е К С Т О В ----------------------------"""
404.
405.     def StyleFinder(event, resultLabel,resultLabel2, final):
406.         newDirectory = ['/'.join(str(dirEntry.get()).split('\\'))+'/']
407.         dirEntry.delete(0, END)
408.         os.mkdir(newDirectory[0]+'Output/')
409.         os.mkdir(newDirectory[0]+'Bases/')
410.         outNewDirectory = [newDirectory[0] + 'Output/']
411.         baseNewDirectory = [newDirectory[0] + 'Bases/']
412.         BaseDownloader(InFileAll(newDirectory[0]), newDirectory, outNewDirectory,
    0)
413.         Cleaner(baseNewDirectory, outNewDirectory, 0)
414.         MorphCorpras(baseNewDirectory[0])
415.         ARFCounter(baseNewDirectory[0])
416.         Ngrams(outNewDirectory, baseNewDirectory)
417.         SkipGramms(outNewDirectory, baseNewDirectory)
```

```
            baseStyleCheck = [[baseNewDirectory[0], baseDirectory[0]],[baseNewDirectory[0], baseDirectory[1]]]
            res = ['ResultPop/', 'ResultRap/']
            for i in range(len(baseStyleCheck)):
                os.mkdir(newDirectory[0]+res[i])
                resultNewDirectory = newDirectory[0] + res[i]
                Vectors('arf.txt', baseStyleCheck[i], resultNewDirectory)
                Vectors('skipgram.txt', baseStyleCheck[i], resultNewDirectory)
                for n in range(2,6):
                    Vectors(str(n)+'ngrams.txt', baseStyleCheck[i], resultNewDirectory)
                KeyWords(baseStyleCheck[i], resultNewDirectory)
                Distances(resultNewDirectory)
            ResultWriter(newDirectory[0], resultLabel,resultLabel2, final)

        def StyleFinder2(event, textTaker, song, resultLabel,resultLabel2, final):
            with open(r'C:/Users/epish/Desktop/учеба/ДИПЛОМ/text/Output/allbase.txt', 'w', encoding = 'utf-8') as inf:
                inf.write(song.get('1.0', END))
            textTaker.destroy()
            newDirectory = ['C:/Users/epish/Desktop/учеба/ДИПЛОМ/text/']
            os.mkdir(newDirectory[0]+'Bases/')
            outNewDirectory = [newDirectory[0] + 'Output/']
            baseNewDirectory = [newDirectory[0] + 'Bases/']
            Cleaner(baseNewDirectory, outNewDirectory, 0)
            MorphCorpras(baseNewDirectory[0])
            ARFCounter(baseNewDirectory[0])
            Ngrams(outNewDirectory, baseNewDirectory)
            SkipGramms(outNewDirectory, baseNewDirectory)
            baseStyleCheck = [[baseNewDirectory[0], baseDirectory[0]],[baseNewDirectory[0], baseDirectory[1]]]
            res = ['ResultPop/', 'ResultRap/']
            for i in range(len(baseStyleCheck)):
                os.mkdir(newDirectory[0]+res[i])
                resultNewDirectory = newDirectory[0] + res[i]
                Vectors('arf.txt', baseStyleCheck[i], resultNewDirectory)
                Vectors('skipgram.txt', baseStyleCheck[i], resultNewDirectory)
                for n in range(2,6):
                    Vectors(str(n)+'ngrams.txt', baseStyleCheck[i], resultNewDirectory)
                KeyWords(baseStyleCheck[i], resultNewDirectory)
                Distances(resultNewDirectory)
            ResultWriter(newDirectory[0], resultLabel,resultLabel2, final)

        def StyleFinder3(event, resultLabel,resultLabel2, final):
            newDirectory = ['/'.join(str(dirEntry2.get()).split('\\'))+'/']
            dirEntry2.delete(0, END)
            os.mkdir(newDirectory[0]+'Bases/')
            outNewDirectory = [newDirectory[0] + 'Output/']
            baseNewDirectory = [newDirectory[0] + 'Bases/']
            Cleaner(baseNewDirectory, outNewDirectory, 0)
            MorphCorpras(baseNewDirectory[0])
            ARFCounter(baseNewDirectory[0])
            Ngrams(outNewDirectory, baseNewDirectory)
```

```
467.            SkipGramms(outNewDirectory, baseNewDirectory)
468.            baseStyleCheck = [[baseNewDirectory[0], baseDirectory[0]],[baseNewDirector
    y[0], baseDirectory[1]]]
469.            res = ['ResultPop/', 'ResultRap/']
470.            for i in range(len(baseStyleCheck)):
471.                os.mkdir(newDirectory[0]+res[i])
472.                resultNewDirectory = newDirectory[0] + res[i]
473.                Vectors('arf.txt', baseStyleCheck[i], resultNewDirectory)
474.                Vectors('skipgram.txt', baseStyleCheck[i], resultNewDirectory)
475.                for n in range(2,6):
476.                    Vectors(str(n)+'ngrams.txt', baseStyleCheck[i], resultNewDirectory)
477.                KeyWords(baseStyleCheck[i], resultNewDirectory)
478.                Distances(resultNewDirectory)
479.            ResultWriter(newDirectory[0], resultLabel,resultLabel2, final)
480.
481.        def TextTaker():
482.            textTaker = Tk()
483.            textTaker.title("Songs text")
484.            textTaker['bg'] = 'black'
485.            sizex = 403
486.            sizey = 703
487.            posx  = 200
488.            posy  = 120
489.            textTaker.wm_geometry("%dx%d+%d+%d" % (sizex, sizey, posx, posy))
490.            song = Text(textTaker, width = 50, height = 40)
491.            song.place(x = 0, y = 0)
492.            checkButton2 = Button(textTaker, text = 'Find out style!')
493.            checkButton2.bind('<Button-
    1>', lambda event: StyleFinder2(event, textTaker, song, resultLabel, resultLabel2, final))

494.            checkButton2.place(x = 150, y = 660)
495.            textTaker.mainloop()
496.
497.        def ResultWriter(resDir, resultLabel, resultLabel2, final):
498.            res = ['ResultPop/', 'ResultRap/']
499.            result = PrettyTable()
500.            result.field_names = ["Test", "Param", "Pop", "Rap"]
501.            #resDir = 'C:/Users/epish/Desktop/учеба/ДИПЛОМ/КлассикаТексты/'
502.            resPop = FileReader(resDir+res[0], 'Final.txt')
503.            resRap = FileReader(resDir+res[1], 'Final.txt')
504.            lines = [['','cos'], ['2gram','man'], ['','ev'],
505.                     ['','cos'], ['3gram','man'], ['','ev'],
506.                     ['','cos'], ['4gram','man'], ['','ev'],
507.                     ['','cos'], ['5gram','man'], ['','ev'],
508.                     ['','cos'], ['arf','man'], ['','ev'],
509.                     ['','cos'], ['skipgrams','man'], ['','ev'],]
510.            resPop = [line.split()[-1] for line in resPop if len(line.split()) > 1]
511.            resRap = [line.split()[-1] for line in resRap if len(line.split()) > 1]
512.            for i in range(len(lines)):
513.                lines[i]. append(round(float(resPop[i]), 3))
514.                lines[i]. append(round(float(resRap[i]),3))
515.                result.add_row(lines[i])
```

```
            resultLabel.insert('end', 'FreqDist parameters\n')
            resultLabel.insert('end', result)
            resPop2 = FileReader(resDir+res[0], 'KeyWordFinal.txt')
            resRap2 = FileReader(resDir+res[1], 'KeyWordFinal.txt')
            result = PrettyTable()
            result.field_names = ["N", "Pop", "Rap"]
            lines = [['   1'],['  10'],[' 100'],['1000']]
            resPop2 = [line.split()[-1] for line in resPop]
            resRap2 = [line.split()[-1] for line in resRap]
            for i in range(len(lines)):
                lines[i]. append(round(float(resPop2[i]), 3))
                lines[i]. append(round(float(resRap2[i]),3))
                result.add_row(lines[i])
            resultLabel2.insert('end', 'KeyWords\n')
            resultLabel2.insert('end', result)
            popMax = 0
            resPop += resPop2
            resRap += resRap2
            for i in range(len(resPop)):
                if resPop[i] < resRap[i]:
                    popMax += 1
            res = [['Pop',popMax/len(resPop)*100], ['Rap', (len(resPop) - popMax)/len(resPop)*100]]
            res.sort(key = lambda x: x[1])
            final.insert('end', 'Style result:\n')
            for i in res[::-1]:
                final.insert('end','\t' + i[0] + '\t' + str(round(i[1]))+'%' + '\n')



        """---------------------------------------------- О К Н О  "П Р И Л О Ж Е Н И Я" ----------------------------------------"""

        root = Tk()
        root.title("Base options")
        sizex = 800
        sizey = 700
        posx  = 100
        posy  = 100
        root.wm_geometry("%dx%d+%d+%d" % (sizex, sizey, posx, posy))
        root['bg'] = 'black'
        mainMenu = Menu()

        text = Text(width = 100, height = 10, bg = 'black', fg = 'white', highlightthickness = 0, bd = 0)
        text.pack(side = BOTTOM)

        file_menu = Menu()
        file_menu.add_command(label="Create", command = CreateButton)
        file_menu.add_command(label="Clean", command = CleanButton)
        file_menu.add_command(label="Refresh", command = RefreshButton)
```

```
565.        file_menu.add_command(label="Lemmas", command = LemmaCorpButton)
566.        file_menu.add_separator()
567.        file_menu.add_command(label="Exit", command = CloseButton)
568.
569.        edit_menu = Menu()
570.        edit_menu.add_command(label = 'Refresh', command = FullInformation)
571.
572.        mainMenu.add_cascade(label="Base", menu = file_menu)
573.        mainMenu.add_cascade(label="Information", menu = edit_menu)
574.
575.
576.        root.config(menu = mainMenu)
577.
578.        information = Label(root, bg = 'black', fg = 'white', font = 'Times 14', height = 6,
    width = 14, anchor = NW)
579.        try:
580.            information['text'] = FullInformation()
581.            information.place(x = 10, y = 10)
582.        except:
583.            information['text'] = 'No base found'
584.            information.place(x = 10, y = 10)
585.
586.        newDirectory = StringVar()
587.        classDirectory  = StringVar()
588.
589.        dirTaker = Label(root, text = 'Enter directory:', bg = 'black', fg = 'white', font = 'Ti
    mes 14', height = 1, anchor = NW)
590.        dirTaker.place(x = 150, y = 10)
591.
592.        dirEntry = Entry(root, textvariable = newDirectory, width = 54)
593.        dirEntry.place(x = 300, y = 14)
594.
595.        checkButton = Button(root, text = 'Find out style!')
596.        checkButton.place(x = 650, y = 13)
597.
598.        dirTaker2 = Label(root, text = 'Enter songs\' texts:', bg = 'black', fg = 'white', font
    = 'Times 14', height = 1, anchor = NW)
599.        dirTaker2.place(x = 150, y = 50)
600.
601.        dirEntry2 = Button(root, width = 46, height = 1, bg = 'white', command = TextTak
    er)
602.        dirEntry2.place(x = 300, y = 52)
603.
604.        checkButton2 = Button(root, text = 'Find out style!', command = TextTaker)
605.        checkButton2.place(x = 650, y = 53)
606.
607.        dirTaker2 = Label(root, text = 'Enter allbase dir:', bg = 'black', fg = 'white', font = '
    Times 14', height = 1, anchor = NW)
608.        dirTaker2.place(x = 150, y = 90)
609.
610.        dirEntry2 = Entry(root, textvariable = classDirectory, width = 54)
611.        dirEntry2.place(x = 300, y = 94)
```

```
612.
613.        checkButton3 = Button(root, text = 'Find out style!')
614.        checkButton3.place(x = 650, y = 93)
615.
616.        resultLabel = Text(root, height = 23, width = 50, bg = 'black', fg = 'white', highlig
    htthickness = 0, bd = 0)
617.        resultLabel.place(x = 40, y = 155)
618.
619.        resultLabel2 = Text(root, height = 10, width = 40, bg = 'black', fg = 'white', highli
    ghtthickness = 0, bd = 0)
620.        resultLabel2.place(x = 440, y = 155)
621.
622.        final = Text(root, height = 10, width = 20, bg = 'black', fg = 'white', bd = 0)
623.        final.place(x = 440, y = 333)
624.
625.        checkButton.bind('<Button-
    1>', lambda event: StyleFinder(event, resultLabel,resultLabel2, final))
626.        checkButton3.bind('<Button-
    1>', lambda event: StyleFinder3(event, resultLabel,resultLabel2, final))
627.
628.        root.mainloop()
```

2. Автоматическое определение авторства

```
import os
import pandas as pd
import nltk
from scipy.spatial import distance
import math
import re
import pymorphy2
import time
from operator import itemgetter

        OutDir = 'C:/Users/epish/Desktop/учеба/ДИПЛОМ/authTest/'
        Dir = 'C:/Users/epish/Desktop/учеба/ДИПЛОМ/TEST/Unprocessed/'

        CleanDirk = 'C:/Users/epish/Desktop/учеба/ДИПЛОМ/authTest/Known/Clean/'
        LemmaDirk = 'C:/Users/epish/Desktop/учеба/ДИПЛОМ/authTest/Known/Lemmas/'
        NgramDirk = 'C:/Users/epish/Desktop/учеба/ДИПЛОМ/authTest/Known/Ngrams/'
        ARFDirk = 'C:/Users/epish/Desktop/учеба/ДИПЛОМ/authTest/Known/ARF/'
        SkipDirk = 'C:/Users/epish/Desktop/учеба/ДИПЛОМ/authTest/Known/Skip/'

        CleanDiru = 'C:/Users/epish/Desktop/учеба/ДИПЛОМ/authTest/Unknown/Clean/'
        LemmaDiru = 'C:/Users/epish/Desktop/учеба/ДИПЛОМ/authTest/Unknown/Lemmas/'
        NgramDiru = 'C:/Users/epish/Desktop/учеба/ДИПЛОМ/authTest/Unknown/Ngrams/'
        ARFDiru = 'C:/Users/epish/Desktop/учеба/ДИПЛОМ/authTest/Unknown/ARF/'
        SkipDiru = 'C:/Users/epish/Desktop/учеба/ДИПЛОМ/authTest/Unknown/Skip/'


        def SentenceSplitter(text):
            split_regex = re.compile(r'[.|!|?|…]')
            return list(filter(lambda t: t, [t.strip() for t in split_regex.split(text)]
    ))

        def FilesPrepare():
            pages = os.listdir(Dir)
            for page in pages:
                files = os.listdir(Dir+page)
                for i in range(len(files)):
                    with open(Dir+page+'/'+files[i], 'r', encoding = 'utf-8') as f:
                        lines = [line for line in f]
                    lines = ''.join(''.join(lines).split('\n\n')[1:]).split('\n')
                    if i%8 == 0:
                        with open(OutDir+'Known/'+page+'.txt', 'a', encoding = 'utf-
    8') as f:
                            for line in lines:
                                f.write(line+'\n')
                            f.write('\n')
                    if i%8 == 1:
                        with open(OutDir+'Unknown/'+page+'.txt', 'a', encoding = 'utf-
    8') as f:
                            for line in lines:
                                f.write(line+'\n')
                            f.write('\n')

        def FileReader(directory, name):
```

```
            with open(directory+name, 'r', encoding = "utf-8") as file:
                return [line for line in file if line != '\n']

        def Clean(word):
            files = list(filter(lambda x: x.endswith('txt'), os.listdir(OutDir+word)))
            for file in files:
                base = FileReader(OutDir+word, file)
                base = re.sub(r'\n', ' ', re.sub(r'[,\'\"\-«:»()–…\/?!\.—-0-9;:*]', ' ', ' '.join(base))).lower()
                with open('C:/Users/epish/Desktop/учеба/ДИПЛОМ/authTest/'+ word + 'Clean/'+file, 'w', encoding = 'utf-8') as f:
                    f.write(base)

        def SkipGramms(word):
            files = list(filter(lambda x: x.endswith('txt'), os.listdir(OutDir+word)))
            for file in files:
                corpra = SentenceSplitter(re.sub(r'[,\'\"\-«:»()–\—-0-9;:*]',' ',' '.join(FileReader(OutDir+word, file)).lower()))
                result = []
                for line in corpra:
                    skip = list(nltk.skipgrams(line.split(),2,1))
                    for skipi in skip:
                        result.append(skipi)
                freq = nltk.FreqDist(result)
                l = len(result)
                freqDist = [(i,freq[i]/l*1000000) for i in list(freq.keys())]
                freqDist.sort(key = lambda x: x[1])
                freqDist = freqDist[::-1]
                result = [str(freqDist[i][0])+'\t'+str(freqDist[i][1]) for i in range(len(freqDist))]
                with open('C:/Users/epish/Desktop/учеба/ДИПЛОМ/authTest/' + word + 'Skip/'+file, 'w', encoding = 'utf-8') as f:
                    for skip in result:
                        f.write(skip+'\n')

        def Lemmatisation(word):
            files = os.listdir(CleanDir)
            morph = pymorphy2.MorphAnalyzer()
            for file in files:
                print(file)
                lines = ''.join(FileReader(CleanDir, file)).split(' ')
                base = [morph.parse(word)[0].normal_form for word in lines if len(word) > 0]
                with open('C:/Users/epish/Desktop/учеба/ДИПЛОМ/authTest/' + word + 'Lemmas/'+file, 'w', encoding = 'utf-8') as f:
                    for lemm in base:
                        f.write(lemm+' ')

        def int_r(num):
            return int(num + (0.5 if num > 0 else -0.5))

        def AverageReducedFrequency(word, n, corpra, L):
            f = len(n)
            d = [n[p + 1] - n[p] for p in range(len(n) - 1)]
            d.append(n[0] + len(corpra) - n[len(n) - 1])
```

```
            segmentLen = int_r(L/f)
            summa = 0
            for i in range(len(d)):
                summa += min(d[i],segmentLen)
            return (word, round(summa/segmentLen, 3))

        # Делает частотные списки по директории
        def ARFCounter(word):
            files = os.listdir(LemmaDir)
            for file in files:
                corpra = [arf for arf in ''.join(FileReader(LemmaDir, file)).split() if len(word) > 0]
                words = {arf: [] for arf in list(nltk.FreqDist(corpra).keys())}
                for i in range(len(corpra)):
                    words[corpra[i]].append(i + 1)
                L = len(corpra)
                result = [AverageReducedFrequency(arf, words[arf], corpra, L) for arf in words]
                result.sort(key = lambda x: x[1])
                result = [str(pair[0])+'\t'+str(pair[1]) for pair in result[::-1]]
                with open('C:/Users/epish/Desktop/учеба/ДИПЛОМ/authTest/' + word + 'ARF/'+file, 'w', encoding = 'utf-8') as f:
                    for arf in result:
                        f.write(arf+'\n')

        def Ngrams(word):
            files = list(filter(lambda x: x.endswith('txt'), os.listdir(OutDir+word)))
            for file in files:
                lines = FileReader(OutDir+word, file)
                ngramList = []
                for song in lines:
                    for ng in list(nltk.ngrams(song,4)):
                        ngramList.append(ng)
                l = len(ngramList)
                freq = nltk.FreqDist(ngramList)
                freqDist = [(n,freq[n]/ l * 1000000) for n in list(freq.keys())]
                freqDist.sort(key = lambda x: x[1])
                freqDist = freqDist[::-1]
                result = [str(freqDist[i][0])+'\t'+str(freqDist[i][1]) for i in range(len(freqDist))]
                with open('C:/Users/epish/Desktop/учеба/ДИПЛОМ/authTest/' + word + 'Ngrams/'+file, 'w', encoding = 'utf-8') as f:
                    for n in result:
                        f.write(n+'\n')
        def Known():
            word = 'Known/'
            Clean(word)
            SkipGramms(word)
            Lemmatisation(word)
            ARFCounter(word)
            Ngrams(word)

        def Unknown():
            word = 'Unknown/'
            Clean(word)
```

```
149.          SkipGramms(word)
150.          Lemmatisation(word)
151.          ARFCounter(word)
152.          Ngrams(word)
153.
154.      #FilesPrepare()
155.      #Known()
156.      #Unknown()
157.
158.      def VectorCreator(knownfile, unknownfile, knowndir, unknowndir):
159.          start_time_def = time.time()
160.          p = [knownfile, 'un'+unknownfile]
161.          with open(knowndir+knownfile, 'r', encoding = 'utf-8') as f:
162.              lines = {line.split('\t')[0]:float(line.split('\t')[1]) for line in f}
163.          with open(unknowndir+unknownfile, 'r', encoding = 'utf-8') as f:
164.              unlines = {line.split('\t')[0]:float(line.split('\t')[1]) for line in f}
165.          known = dict.fromkeys(list(lines)+list(unlines))
166.          unknown = {}
167.          for word in known:
168.              if word in list(lines):
169.                  known[word] = lines[word]
170.              else:
171.                  known[word] = 0
172.              if word in list(unlines):
173.                  unknown[word] = unlines[word]
174.              else:
175.                  unknown[word] = 0
176.          vectors = [list(known.values()), list(unknown.values())]
177.          print (distance.cosine(vectors[0], vectors[1]), time.time() - start_time_def
      , sep = '\t')
178.          return distance.cosine(vectors[0], vectors[1])
179.
180.      def SecondType(dist, art):
181.          vectors = []
182.          for i in range(len(dist) - 1):
183.                  sqrt = dist[i][0]**2 + dist[i][1]**2 + dist[i][2]**2
184.                  vectors.append(sqrt)
185.          return art[min(enumerate(vectors), key=itemgetter(1))[0]][:-4]
186.
187.      def FTables():
188.          knownp = [NgramDirk , ARFDirk , SkipDirk]
189.          unknownp = [NgramDiru , ARFDiru , SkipDiru]
190.          art = os.listdir(knownp[0])
191.          restable = pd.DataFrame(columns = ['Result', 'Time'], index = [artist[6:-
      4] for artist in art])
192.          print(restable)
193.          start_time = time.time()
194.          for unknown in range(len(art)):
195.              res = [[]]
196.              for known in range(len(art)):
197.                  print(art[unknown][:-4], ' vs ', art[known][:-4])
198.                  for i in range(len(knownp)):
199.                      res[known].append(VectorCreator(art[known], art[unknown], knownp
      [i], unknownp[i]))
```

```
                res.append([])
            name = SecondType(res, art)
            restable.loc[art[unknown][6:-4], 'Result'] = name
            restable.loc[art[unknown][6:-4], 'Time'] = time.time() - start_time
            print(restable)
            restable.to_excel('C:/Users/epish/Desktop/учеба/ДИПЛОМ/authTest/text.xlsx')

    FTables()
```

3. Два алгоритма определения авторства

```
def FirstType(dist):
    param = ['key1', 'key1000', 'ngram4cos', 'skipcos']
    dist = {param[i]:dist[i] for i in range(4)}
    tables = TableReader()
    res = []
    minIndex = ''
    for p in param:
        minim = 10000000000
        for d in range(len(tables[p][:20])):
                delta = []
                for i in range(len(list(tables[p].iloc[d])[1:])):
                    if math.isnan(list(tables[p].iloc[d])[1:][i]):
                        delta.append(abs(float(dist[p][i]) - 0))
                    else:
                        delta.append(abs(float(dist[p][i]) - list(tables[p].iloc[d])[1:][i]))
                deltaSum = sum(delta)
                #print(deltaSum)
                if deltaSum < minim:
                    minim = deltaSum
                    minIndex = list(tables[p].iloc[d])[0]
            res.append((p, minIndex, minim))
        dist = dict.fromkeys(list(tables['key1']['Unnamed: 0'][:20]), 0)
        print(res)
        for param in res:
            for i in range(20):
                if math.isnan(tables[param[0]].iloc[i][param[1]]):
                    dist[tables['key1']['Unnamed: 0'][i]] += 0
                else:
                    dist[tables['key1']['Unnamed: 0'][i]] += tables[param[0]].iloc[i][param[1]]
        p = [(art,dist[art]) for art in list(dist)]
        p.sort(key = lambda x: x[1])
        print("First param " + p[0][0])

    def SecondType(dist):
        res = [[]]
        for i in range(20):
            for j in range(4):
                res[i].append(float(dist[j][i]))
            res.append([])
        vectors = []
        for i in range(len(res) - 1):
```

```
                mass = int_r(max(res[i]))
                sqrt = (res[i][0]/mass)**2 + (res[i][1]/mass)**2 + res[i][2]**2 + res[i][3]**2
                vectors.append(sqrt)
            auth = ['2rbina2rista', 'basta', 'brb', 'eldzhey', 'face', 'feduk',
        'gnoiny', 'guf', 'husky', 'kasta', 'korzh', 'krovostok', 'kunteynir', 'lizer', 'lsp', 'morgenshtern', 'noizemc', 'skriptonite', 'timati', 'xleb']
            print("Second param " + auth[min(enumerate(vectors), key=itemgetter(1))[0]])
```

Таблицы

1. ARFCos

	2rbina2rist	basta	brb	eldzhey	face	feduk	gnoiny	guf	husky
xleb	0,23693972	0,14332253	0,22113209	0,18058228	0,19047329	0,16515380	0,13339712	0,12926447	0,19721692
timati	0,24773469	0,05321429	0,15604826	0,12623923	0,05833111	0,13208671	0,11016793	0,14642386	0,14033208
skriptonite	0,15109748	0,04379675	0,15654257	0,11258760	0,09633315	0,11904811	0,04197751	0,08437229	0,08381109
noizemc	0,12334716	0,09019611	0,22183071	0,18161303	0,20336646	0,11891808	0,05383801	0,08391847	0,13596838
morgensht	0,28448101	0,09367503	0,13747021	0,08561561	0,04290922	0,19973889	0,12862462	0,09978414	0,14621093
lsp	0,16551740	0,04793529	0,17969543	0,10059344	0,11352852	0,07888203	0,04859724	0,06198559	0,12578628
lizer	0,23500932	0,06054155	0,14590692	0,11301304	0,03426045	0,17445367	0,11341017	0,11378223	0,11941775
kunteynir	0,13161364	0,12227120	0,19752412	0,19631840	0,21105148	0,18730027	0,07806098	0,13039328	0,09510950
krovostok	0,12643921	0,11773589	0,22389756	0,18913405	0,22951544	0,14670921	0,06287034	0,08660498	0,12827272
korzh	0,14531678	0,05629812	0,18221825	0,16268000	0,13187074	0,13181880	0,06053681	0,08989620	0,10938419
kasta	0,12088507	0,08981714	0,21231655	0,1834948	0,20007701	0,11429218	0,05184489	0,08213761	0,13474098
husky	0,16027597	0,07822645	0,14001595	0,13429712	0,10940111	0,16977855	0,07520506	0,12282887	
guf	0,17277582	0,07244141	0,14923103	0,10775194	0,11184115	0,13882377	0,05417200		0,12282887
gnoiny	0,12793413	0,03954548	0,13935545	0,10724000	0,10299882	0,10182264		0,05417200	0,07520506
feduk	0,21927528	0,08341637	0,21198554	0,13842694	0,15639087		0,10182264	0,13882377	0,16977855
face	0,24452648	0,05862547	0,11385599	0,08158434		0,15639087	0,10299882	0,11184115	0,10940111
eldzhey	0,24464504	0,08677122	0,15472217		0,08158434	0,13842694	0,10724000	0,10775194	0,13429712
brb	0,25256206	0,13247892		0,15472217	0,11385599	0,21198554	0,13935545	0,14923103	0,14001595
basta	0,15965294		0,13247892	0,08677122	0,05862547	0,08341637	0,03954548	0,07244141	0,07822645
2rbina2rist		0,15965294	0,25256206	0,24464504	0,24452648	0,21927528	0,12793413	0,17277582	0,16027597

0,13864426	0,15432395	0,16843881	0,20075978	0,20338029	0,13735807	0,21131307	0,14394143	0,15223027	0,20603267	
0,17685313	0,11115971	0,22503398	0,21677415	0,07157561	0,11318132	0,09977997	0,18928114	0,09865157		0,20603267
0,06670765	0,04992908	0,09570824	0,10047111	0,09624252	0,0627449	0,13838389	0,0717805		0,09865157	0,15223027
0,02091257	0,05197080	0,03630515	0,08950615	0,18989921	0,06704079	0,23260874		0,0717805	0,18928114	0,14394143
0,23693049	0,16697711	0,24507955	0,25337261	0,04841530	0,12465258		0,23260874	0,13838389	0,09977997	0,21131307
0,06643662	0,08314769	0,08689168	0,13882580	0,11460095		0,12465258	0,06704079	0,0627449	0,11318132	0,13735807
0,19821344	0,10215031	0,23008455	0,23048970		0,11460095	0,04841530	0,18989921	0,09624252	0,07157561	0,20338029
0,08348879	0,11772758	0,07045015		0,23048970	0,13882580	0,25337261	0,08950615	0,10047111	0,21677415	0,20075978
0,03830924	0,09564480		0,07045015	0,23008455	0,08689168	0,24507955	0,03630515	0,09570824	0,22503398	0,16843881
0,06215316		0,09564480	0,11772758	0,10215031	0,08314769	0,16697711	0,05197080	0,04992908	0,11115971	0,15432395
	0,06215316	0,03830924	0,08348879	0,19821344	0,06643662	0,23693049	0,02091257	0,06670765	0,17685313	0,13864426
0,13474098	0,10938419	0,12827272	0,09510950	0,11941775	0,12578628	0,14621093	0,13596838	0,08381109	0,14033208	0,19721692
0,08213761	0,08989620	0,08660498	0,13039328	0,11378223	0,06198559	0,09978414	0,08391847	0,08437229	0,14642386	0,12926447
0,05184489	0,06053681	0,06287034	0,07806098	0,11341017	0,04859724	0,12862462	0,05383801	0,04197751	0,11016793	0,13339712
0,11429218	0,13181880	0,14670921	0,18730027	0,17445367	0,07888203	0,19973889	0,11891808	0,11904811	0,13208671	0,16515380
0,20007701	0,13187074	0,22951544	0,21105148	0,03426045	0,11352852	0,04290922	0,20336646	0,09633315	0,05833111	0,19047329
0,1834948	0,16268000	0,18913405	0,19631840	0,11301304	0,10059344	0,08561561	0,18161303	0,11258760	0,12623923	0,18058228
0,21231655	0,18221825	0,22389756	0,19752412	0,14590692	0,17969543	0,13747021	0,22183071	0,15654257	0,15604826	0,22113209
0,08981714	0,05629812	0,11773589	0,12227120	0,06054155	0,04793529	0,09367503	0,09019611	0,04379675	0,05321429	0,14332253
0,12088507	0,14531678	0,12643921	0,13161364	0,23500932	0,16551740	0,28448101	0,12334716	0,15109748	0,24773469	0,23693972
kasta	**korzh**	**krovostok**	**kunteynir**	**lizer**	**lsp**	**morgensht**	**noizemc**	**skriptonite**	**timati**	**xleb**

2. Максимальные расстояния между исполнителями

xleb	2rbina2rista	noizemc	noizemc	lizer	lizer	lizer	2rbina2rist	2rbina2rista
timati	2rbina2rista	noizemc	noizemc	noizemc	noizemc	korzh	xleb	xleb
skriptonite	brb	noizemc	noizemc	brb	brb	2rbina2ri	xleb	xleb
noizemc	morgenshtern	brb	brb	brb	brb	2rbina2ri	xleb	xleb
morgenshtern	2rbina2rista	noizemc	noizemc	noizemc	noizemc	brb	xleb	2rbina2rista
lsp	brb	noizemc	noizemc	noizemc	noizemc	noizemc	xleb	xleb
lizer	2rbina2rista	noizemc	brb	brb	brb	2rbina2ri	xleb	xleb
kunteynir	morgenshtern	noizemc	noizemc	noizemc	noizemc	lizer	xleb	xleb
krovostok	morgenshtern	noizemc	noizemc	brb	brb	2rbina2ri	xleb	xleb
korzh	brb	noizemc	noizemc	brb	brb	2rbina2ri	xleb	xleb
kasta	morgenshtern	noizemc	brb	brb	brb	brb	xleb	xleb
husky	xleb	noizemc	noizemc	noizemc	noizemc	noizemc	xleb	xleb
guf	2rbina2rista	noizemc	noizemc	noizemc	noizemc	noizemc	xleb	xleb
gnoiny	brb	noizemc	noizemc	noizemc	noizemc	2rbina2ri	xleb	xleb
feduk	2rbina2rista	noizemc	noizemc	noizemc	lizer	lizer	2rbina2rist	2rbina2rista
face	2rbina2rista	noizemc	noizemc	noizemc	noizemc	lizer	xleb	2rbina2rista
eldzhey	2rbina2rista	noizemc	noizemc	noizemc	noizemc	lizer	xleb	xleb
brb	2rbina2rista	noizemc	noizemc	noizemc	lizer	lizer	xleb	xleb
basta	2rbina2rista	noizemc	noizemc	brb	brb	2rbina2ri	xleb	xleb
2rbina2rista	morgenshtern	noizemc	noizemc	noizemc	lizer	lizer	xleb	xleb
	arfcos	**arfcity**	**arfeu**	**key1**	**key10**	**key100**	**key1000**	**ngram2cos**	**ngram2city**

2rbina2rist	2rbina2rista	2rbina2rista	2rbina2rista	2rbina2rist	2rbina2rista	brb	2rbina2rista	2rbina2rista	brb
xleb	xleb	2rbina2rista	xleb	2rbina2rist	brb	xleb	2rbina2rista	brb	xleb
xleb	xleb	2rbina2rista	xleb	xleb	2rbina2rista	xleb	2rbina2rista	2rbina2rista	xleb
xleb	xleb	xleb	xleb	xleb	brb	xleb	xleb	brb	xleb
xleb	xleb	2rbina2rista	xleb	xleb	2rbina2rista	xleb	2rbina2rista	2rbina2rista	xleb
xleb	xleb	xleb	xleb	xleb	brb	xleb	2rbina2rista	brb	xleb
xleb	xleb	2rbina2rista	xleb	xleb	2rbina2rista	xleb	2rbina2rista	2rbina2rista	xleb
xleb	xleb	xleb	xleb	xleb	brb	xleb	xleb	2rbina2rista	xleb
xleb	xleb	xleb	xleb	xleb	xleb	xleb	xleb	brb	xleb
xleb	xleb	brb	xleb	xleb	brb	xleb	2rbina2rista	brb	xleb
xleb	xleb	xleb	xleb	xleb	brb	xleb	xleb	brb	xleb
xleb	xleb	xleb	xleb	xleb	xleb	xleb	xleb	xleb	xleb
xleb	xleb	2rbina2rista	xleb	2rbina2rist	2rbina2rista	xleb	2rbina2rista	2rbina2rista	xleb
xleb	xleb	xleb	xleb	xleb	xleb	xleb	xleb	brb	xleb
xleb	xleb	2rbina2rista	xleb	2rbina2rist	2rbina2rista	xleb	2rbina2rista	2rbina2rista	xleb
xleb	xleb	2rbina2rista	xleb	xleb	2rbina2rista	xleb	2rbina2rista	2rbina2rista	xleb
xleb	xleb	2rbina2rista	xleb	xleb	2rbina2rista	xleb	2rbina2rista	2rbina2rista	xleb
xleb	xleb	2rbina2rista	xleb	xleb	2rbina2rista	xleb	2rbina2rista	2rbina2rista	xleb
xleb	xleb	xleb	xleb	xleb	brb	xleb	xleb	2rbina2rista	xleb
xleb	xleb	xleb	xleb	xleb	xleb	xleb	xleb	xleb	xleb
ngram2eu	**ngram3cos**	**ngram3city**	**ngram3eu**	**ngram4cos**	**ngram4city**	**ngram4eu**	**ngram5cos**	**ngram5city**	**ngram5eu**

brb	brb	feduk	
xleb	2rbina2ri	xleb	
xleb	2rbina2ri	xleb	
xleb		brb	xleb
xleb	2rbina2ri	xleb	
xleb	brb	xleb	
xleb	2rbina2ri	xleb	
xleb	brb	xleb	
xleb	brb	xleb	
xleb	2rbina2ri	xleb	
xleb	brb	xleb	
xleb	brb	xleb	
xleb	brb	xleb	
xleb	brb	xleb	
xleb	2rbina2ri	xleb	
xleb	2rbina2ri	xleb	
xleb	brb	xleb	
xleb	feduk	xleb	
xleb	brb	xleb	
xleb	brb	xleb	
skipcos	**skipcity**	**skipeu**	

3. Минимальные расстояния между авторами

xleb	guf	feduk	feduk	kunteynir	kunteynir
timati	basta	feduk	eldzhey	kunteynir	kunteynir
skriptonite	gnoiny	morgenshtern	basta	krovostok	krovostok
noizemc	kasta	kasta	kasta	kasta	kasta
morgenshtern	face	face	face	kunteynir	gnoiny
lsp	basta	timati	guf	kunteynir	guf
lizer	face	face	basta	basta	basta
kunteynir	krovostok	feduk	eldzhey	guf	timati
krovostok	noizemc	guf	gnoiny	kasta	gnoiny
korzh	skriptonite	face	basta	kasta	kasta
kasta	noizemc	korzh	skriptonite	krovostok	krovostok
husky	gnoiny	timati	guf	kunteynir	kunteynir
guf	gnoiny	timati	lsp	kunteynir	krovostok
gnoiny	basta	face	krovostok	krovostok	krovostok
feduk	lsp	brb	brb	eldzhey	eldzhey
face	lizer	timati	morgenshtern	kunteynir	kunteynir
eldzhey	face	feduk	feduk	kunteynir	kunteynir
brb	face	feduk	feduk	feduk	feduk
basta	gnoiny	face	gnoiny	kasta	krovostok
2rbina2rista	kasta	brb	brb	brb	brb
	arfcos	**arfcity**	**arfeu**	**key1**	**key10**

eldzhey	eldzhey	face	skriptonite	skriptonite	guf	kasta	kasta	noizemc	korzh
kunteynir	eldzhey	basta	basta	basta	basta	basta	basta	basta	basta
krovostok	basta	kasta	kasta	kasta	kasta	kasta	kasta	noizemc	noizemc
kasta	basta	kasta	kasta	kasta	kasta	kasta	kasta	kasta	kasta
gnoiny	basta	face	face	face	face	face	face	lizer	lizer
gnoiny	face	noizemc	kasta	noizemc	noizemc	noizemc	noizemc	noizemc	noizemc
basta	basta	face	korzh	face	basta	korzh	basta	basta	korzh
eldzhey	brb	krovostok	krovostok	krovostok	krovostok	krovostok	krovostok	krovostok	krovostok
basta	basta	kasta	kasta	kasta	noizemc	noizemc	noizemc	noizemc	noizemc
krovostok	basta	kasta	basta	kasta	noizemc	basta	noizemc	noizemc	basta
basta	krovostok	krovostok	noizemc	krovostok	noizemc	noizemc	noizemc	noizemc	noizemc
kunteynir	face	krovostok	krovostok	gnoiny	krovostok	krovostok	krovostok	noizemc	noizemc
gnoiny	face	krovostok	krovostok	krovostok	noizemc	noizemc	noizemc	noizemc	noizemc
basta	basta	krovostok	krovostok	krovostok	krovostok	kasta	krovostok	noizemc	noizemc
brb	brb	kasta	kasta	kasta	kasta	basta	kasta	noizemc	basta
gnoiny	gnoiny	skriptonite	lizer	skriptonite	lizer	lizer	skriptonite	lizer	lizer
kunteynir	brb	noizemc	noizemc	noizemc	noizemc	noizemc	noizemc	noizemc	noizemc
feduk	eldzhey	skriptonite	kasta	gnoiny	kasta	kasta	kasta	noizemc	basta
gnoiny	gnoiny	kasta	kasta	kasta	noizemc	noizemc	noizemc	noizemc	noizemc
brb	brb	kasta	krovostok	husky	krovostok	kasta	krovostok	noizemc	noizemc
key100	**key1000**	**ngram2cos**	**ngram2city**	**ngram2eu**	**ngram3cos**	**ngram3city**	**ngram3eu**	**ngram4cos**	**ngram4city**

noizemc	noizemc	lizer	noizemc	noizemc	lizer	noizemc
basta	basta	basta	basta	basta	lizer	noizemc
noizemc	noizemc	noizemc	noizemc	lizer	lizer	noizemc
kasta	guf	kasta	kasta	lizer	kasta	krovostok
face	lizer	lizer	noizemc	lizer	lizer	noizemc
noizemc	noizemc	noizemc	noizemc	lizer	lizer	noizemc
basta	noizemc	korzh	noizemc	skriptonite	face	noizemc
krovostok	noizemc	krovostok	noizemc	noizemc	lizer	noizemc
noizemc	noizemc	noizemc	noizemc	noizemc	noizemc	noizemc
noizemc	noizemc	basta	noizemc	lizer	lizer	noizemc
noizemc	noizemc	noizemc	noizemc	noizemc	noizemc	noizemc
noizemc	noizemc	noizemc	noizemc	lizer	lizer	noizemc
noizemc	noizemc	noizemc	noizemc	lizer	lizer	noizemc
noizemc	noizemc	noizemc	noizemc	lizer	lizer	noizemc
kasta	noizemc	lizer	noizemc	noizemc	skriptonite	noizemc
noizemc	lizer	lizer	noizemc	lizer	lizer	noizemc
noizemc	noizemc	lizer	noizemc	lizer	lizer	noizemc
noizemc	basta	lizer	noizemc	lizer	morgenshtern	noizemc
noizemc	noizemc	noizemc	noizemc	lizer	lizer	noizemc
noizemc	noizemc	basta	noizemc	skriptonite	lizer	noizemc
ngram4eu	**ngram5cos**	**ngram5city**	**ngram5eu**	**skipcos**	**skipcity**	**skipeu**