Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ Фундаментальные науки

КАФЕДРА Математическое моделирование

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ

НА ТЕМУ:

Реализация многокритериального поиска неточных смысловых копий текста на основе применения методов машинного обучения

Студент	ФН12-41М		Кононенко А.А.
	(группа)	(подпись)	(инициалы, фамилия)
Руководитель ВКР			Виноградова М.С.
		(подпись)	(инициалы, фамилия)
Нормоконтролер			Велищанский М.А.
		(подпись)	(инициалы, фамилия)

2025 год

РЕФЕРАТ

Расчётно-пояснительная записка содержит страниц 47, рисунков 4, приложений 1.

TF-IDF, WORD2VEC, SENTENCE-BERT, BLUE, ROGUE, METEOR, BERT-SCORE, CBOW, CHRF, CHRF++, TRANSFORMER, КОСИНУСНОЕ СХОДСТВО, EMBEDDING

Цель отчёта – разработать и апробировать методы автоматического обнаружения семантически близких или слегка изменённых копий текстовых фрагментов в большом корпусе документов.

В работе изучены современные подходы к представлению текстов в векторном пространстве, включая традиционные TF-IDF-векторы, плотные embeddings слов и предложений (Word2Vec, Sentence-BERT) и методы на их основе (косинусное сходство, скалярное произведение, расстояние Левенштейна).

Рассмотрены алгоритмы кластеризации (K-Means) для эффективного сравнения больших объёмов данных, а также приёмы снижения размерности и ускорения вычислений (LSA, SVD-разложение). Проведен обзор метрик качества: точности, полноты, F1-меры.

По результатам экспериментов показано, что комбинация Sentence-BERT embeddings с косинусным сходством обеспечивает наилучший компромисс между точностью и полнотой при приемлемой скорости обработки.

Содержание

ВВЕДЕНИЕ

Многокритериальный поиск неточных смысловых копий текста — это задача обнаружения текстов, которые, хотя и не являются точными дубликатами, передают схожий смысл с исходным текстом. Они могут отличаться по формулировкам, структуре предложений, использованию синонимов или порядке слов, но остаются семантически эквивалентными.

В данной задаче учитывается несколько критериев сходства. Эти критерии могут включать семантическую близость, синтаксическое сходство, стилистические особенности и другие параметры.

Решение данной задачи является важным и актуальным в таких сферах, как:

- Обнаружение плагиата: выявление перефразированного контента без надлежащей атрибуции, то есть без указания автора произведения, для обеспечения академической честности и защиты авторских прав.

- Кластеризация документов — это неконтролируемое обучение (unsupervised learning), цель которого — автоматически группировать тексты по сходству без предварительной разметки. В основе лежит предположение: схожие по содержанию документы должны находиться в одном кластере.

- Контент-фильтрация: удаление дубликатов или похожих статей в новостных агрегаторах и социальных сетях.

- Поисковая оптимизация: улучшение релевантности результатов поиска путем учета похожих по смыслу запросов и документов.

- Чат-боты и ассистенты: распознавание перефразированных вопросов для предоставления правильных ответов.

- Социальные сети и медиа: обнаружение повторяющейся информации, даже если она представлена в разных формах.

- Аналитика и мониторинг: выявление трендов и повторяющихся тем в больших объемах текстовых данных.

1 ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

1.1 Задача классификации

Классификация текстов представляет собой процесс отнесения текста к заранее определённой категории. Эта задача принадлежит к области обработки естественного языка (Natural Language Processing, NLP). Обычно её решают посредством разработки нейронных сетей различной архитектуры, которые обучаются связывать тексты с заданными классами (категориями) на тренировочных данных.

- **Нейронные сети** — архитектуры RNN, CNN, трансформеры и их комбинации обучаются на размеченных данных, связывая тексты с классами.

- **Классические алгоритмы** — SVM, Naive Bayes, решающие деревья, которые используют ручные или статистические признаки (n-grams, TF-IDF и др.).

Однако возможно решить эту задачу и без использования технологий искусственных нейронных сетей.

В работе [1] предложен метод классификации текстов, основанный на обучении без учителя с применением вычисления семантической близости текстов. Определение семантического сходства документов — это процесс оценки степени содержания, общего между текстами [1]. Задача определения семантической близости текстов также применяется в информационном поиске, оценке соответствия найденной информации поисковому запросу, классификации текстов и прочих областях.

К подходам решения этой задачи относятся:

- **Строковые методы**: сюда входят функции вычисления сходства на основе символов и на основе токенов [2].

- **Корпусные методы** (distributional): эти методы основаны на оценке сходства между текстами с использованием информации, извлечённой из большого корпуса данных (word2vec, fastText, GloVe).

- **Методы на основе знаний**: включают алгоритмы, использующие информацию, представленную в виде знаний. Эти знания хранятся в семантических сетях и представляют собой различные утверждения, которые система управления знаниями использует для вывода новых знаний, устранения противоречий и т.д.[2]. Примерами являются семантические сети, онтологии, где отношение между словами задаёт внешний источник.

- **Гибридные методы**: данный подход сочетает в себе комбинацию вышеперечисленных методов.

Строковые методы являются одними из самых популярных и простых в реализации. Как было отмечено, они подразделяются на:

- **Символьные методы**: семантическое сходство вычисляется через измерение редакционного расстояния. Редакционное расстояние относится к метрике, которая измеряет минимальное количество операций, необходимых для преобразования одной строки в другую. Эти операции обычно включают вставку, удаление или замену символов. Редакционное расстояние позволяет количественно оценить степень различия между двумя строками на уровне символов. Наиболее известным примером такой метрики являются:

- Расстояние Левенштейна — это минимальное число операций (включающих вставку, удаление и замену), необходимых для преобразования одной строки в другую, и оно особенно полезно для исправления опечаток.

Обычно в математике элементы строк нумеруются с единицы, в отличие от большинства языков программирования, где нумерация начинается с нуля. Допустим, у нас есть две строки S1 и S2 с длинами i и j соответственно. Редакционное расстояние (иначе говоря, расстояние Левенштейна) d(S1, S2) можно высчитать при помощи следующей рекуррентной формулы

$$d(S1, S2) = d(i, j),$$

где

$$d(i,j) = \begin{cases} 0, & \text{если } i = 0 \text{ и } j = 0, \\ i, & \text{если } j = 0 \text{ и } i > 0, \\ j, & \text{если } i = 0 \text{ и } j > 0, \\ \min\Big(d(i-1,j)+1, & \\ \quad d(i,j-1)+1, & \\ \quad d(i-1,j-1)+\mathbf{1}_{[s_i \neq t_j]}\Big), & \text{если } i > 0 \text{ и } j > 0, \end{cases}$$

где m(a,b) равна нулю, если a=b и единице в противном случае, min{a,b,c} возвращает наименьший из аргументов.

Если мы идём по индексу i, это означает удаление (D) из первой строки, по j — вставку (I) в первую строку, а движение по обоим индексам может указывать на замену символа (R) или отсутствие изменений (M).

- Расстояние Дамерау — Левенштейна — это улучшенная версия расстояния Левенштейна, где к стандартным операциям вставки, удаления и замены добавлена операция транспозиции (перестановка) символов.

Определение расстояния Дамерау — Левенштейна между строками a и b осуществляется с помощью функции $d_{a,b}(|a|, |b|)$, где

$$d_{a,b}(i,j) = \begin{cases} \max(i,j), & \text{если } \min(i,j) = 0, \\ \min\begin{cases} d_{a,b}(i-1,j)+1, & \\ d_{a,b}(i,j-1)+1, & \\ d_{a,b}(i-1,j-1)+\mathbf{1}_{(a_i \neq b_j)}, & \\ d_{a,b}(i-2,j-2)+1, & \text{если } i,j > 1,\ a_i = b_{j-1},\ a_{i-1} = b_j, \end{cases} & \end{cases}$$

где $\mathbf{1}_{(a_i \neq b_j)}$ это индикаторная функция, равная нулю при $a_i = b_j$ и 1 в противном случае.

В каждом рекурсивном вызове рассматриваются такие случаи:

- $d_{a,b}(i-1,j)+1$ соответствует удалению символа (из a в b),

- $d_{a,b}(i,j-1)+1$ соответствует вставке (из a в b),

- $d_{a,b}(i-1,j-1)+1$ $(a_i \neq b_j)$ соответствие или несоответствие, в зависимости от совпадения символов,

- $d_{a,b}(i-2,j-2)+1$ в случае перестановки двух последовательных символов.

Основной недостаток этого подхода заключается в высокой вычислительной сложности при работе с длинными строками.

- **Методы на основе токенов**: в отличие от symbols методов, входной текст представляется как набор токенов, которые могут быть словами, фразами или предложениями.

Сходство между двумя текстами в этом случае оценивается с помощью **функций подобия** [2]. Функции подобия — это математические функции, которые используются для оценки степени сходства между объектами. В нашем случае эти функции измеряют сходство между наборами токенов (слов, фраз или предложений) в текстах. Ниже приведены наиболее распространённые функции подобия:

1. **Косинусное сходство**: измеряет косинус угла между векторами, представляющими тексты в пространстве признаков.

$$\cos(\mathbf{A}, \mathbf{B}) = \frac{\mathbf{A} \cdot \mathbf{B}}{||\mathbf{A}|| \cdot ||\mathbf{B}||} = \frac{\sum_{i=1}^{n} A_i \, B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \, \sqrt{\sum_{i=1}^{n} B_i^2}}.$$

Оно не зависит от длины векторов — оценивает угловую близость.

2. **Коэффициент Жаккара**: вычисляет отношение размера пересечения токенов к размеру их объединения. Запишем формулу коэффициента Жаккара

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|}.$$

Где:

- A, B — множества токенов (например слов) в двух текстах,
- |A ∩ B| — количество общих элементов в множествах A и B,
- |A ∪ B| — общее количество элементов в объединении множеств A и B.

3. **Мера перекрытия**: оценивает долю общих токенов в одном тексте относительно другого.

$$\text{overlap}(A, B) = \frac{|A \cap B|}{\min(|A|, \ |B|)}.$$

Где A, B — множества токенов в двух текстах.

4. **Скалярное произведение** (Dot Product)

$$\mathbf{A} \cdot \mathbf{B} = \sum_{i=1}^{n} A_i B_i.$$

Учитывает не только направление, но и величину (модули) векторов. В задачах, где embeddings предложения вычисляется как сумма векторов слов $p = \sum_{w \in sentence} e_w$, длина предложения (число слов) влияет на сумму embeddings. Следовательно, $p_1 \cdot p_2 = \sum_{i=1}^{d} p_{1,i} p_{2,i}$ будет тем больше, чем ближе предложения по содержанию и чем больше их длины. Это свойство может быть полезно, когда длинные тексты считаются более информативными.

5. **Евклидово расстояние** (Euclidean Distance)

$$d_e(A, B) = ||\mathbf{A} - \mathbf{B}||_2 = \sqrt{\sum_{i=1}^{n} (A_i - B_i)^2}.$$

Используется реже для текстов напрямую, но полезно в некоторых визуализациях.

Однако подход на основе symbols методов имеет недостаток в виде высоких вычислительных затрат при обработке большого количества длинных строк. Методы на основе токенов помогают преодолеть этот недостаток.

Рассмотрим вновь задачу классификации. Алгоритм классификации можно разделить на несколько шагов:

1. Преобразование текстов в векторную форму;
2. Расчет матрицы сходства между текстами;
3. Определение порогового значения для оценки сходства текстов;
4. Сравнение текстов с использованием матрицы сходства.

1) Для векторного представления текстов предлагается использовать метод вычисления TF-IDF векторов, метод Word2Vec и трансформер Sentence-BERT.

a) **TF-IDF** (term frequency-inverse document frequency)

TF (частота терма) определяется как отношение количества вхождений конкретного слова к общему числу слов в исходном тексте. IDF (обратная документная частота) вычисляется как отношение общего количества документов к числу документов, в которых присутствует данное слово [3]. Алгоритм TF-IDF относится к методам на основе токенов и является одним из самых распространенных. Идея данного метода заключается в том, что слова, часто встречающиеся в одном документе, но редко в других, являются ключевыми.

- Формула для частоты терминов (Term Frequency – TF) выглядит так

$$\mathrm{tf}(t, d) = \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}},$$

где $f_{t,d}$ – количество вхождений термина t в документ d,

$\sum_{t' \in d} f_{t',d}$ – общее количество терминов (слов) в конкретном документе d.

- Запишем формулу для обратной частоты документа (Inverse Document Frequency – IDF)

$$\mathrm{idf}(t, D) = \log \frac{|D|}{|\{ d' \in D : t \in d' \}|},$$

где

- TF-IDF в итоге получается перемножением этих функций

$$\text{tf-idf}(t, d, D) = \mathrm{tf}(t, d) \times \mathrm{idf}(t, D).$$

Основные ограничения TF-IDF:

- Отсутствие учета порядка слов: TF-IDF игнорирует порядок слов, что приводит к потере информации о синтаксисе и структуре предложения.

- Не учитывает семантические связи: слова с похожим значением, но разным написанием (синонимы) рассматриваются как разные признаки.

- Высокая размерность: векторная репрезентация имеет размерность, равную размеру словаря, что может быть неэффективно для больших корпусов.

b) **Word2Vec**

Word2Vec – это алгоритм для обучения семантических векторных представлений слов (embeddings), где слова похожие по смыслу имеют близкие векторы в пространстве. Мы получаем представление каждого документа как среднее векторное представление входящих в него слов. Идея метода заключается в том, что слова, встречающиеся в похожих контекстах, имеют близкие векторы.

Архитектуры:

- Skip-gram: предсказывает контекстные слова по целевому слову
- CBOW (Continuous Bag of Words): предсказывает целевое слово по контексту.

На вход модели подаются токенизированные корпус текстов (к примеру предложения), а задачей является минимизация функции потерь

$$\mathcal{L} = -\sum_{t=1}^{T} \sum_{j=-c \le j \le c, j \neq 0} \log p(w_{t+j} \mid w_t)$$

где:

- T — количество слов,
- c — размер окна контекста,
- $p(w_{t+j} \mid w_t)$ вычисляется через softmax:

$$P(w_o \mid w_I) = \frac{\exp\left(\mathbf{v}'^{\top}_{w_o} \mathbf{v}_{w_I}\right)}{\sum_{w=1}^{|V|} \exp\left(\mathbf{v}'^{\top}_{w} \mathbf{v}_{w_I}\right)},$$

где

- V_{w_I} — вектор входного (центрального) слова,
- V'_{w_O} — вектор «выходного» (контекстного) слова,

- |V| — размер словаря.

В результате

- После обучения берут либо входные векторы v_w, либо среднее $(v_w+v_w')/2$ как итоговые представления слов.

- Embeddings отражают семантику:
v("король") − v ("мужчина") + v ("женщина") ≈ v ("королева").

- Размерность d обычно выбирают от 100 до 300.

Полученные векторы способны улавливать семантические связи между словами, однако:
- Недостаток контекста: каждое слово имеет свое фиксированное векторное представление, независимо от контекста, что приводит к потере значений в разных контекстах (проблема полисемии).
- Необходимость агрегации: для получения представления предложения или документа требуется агрегировать векторы слов, что не всегда эффективно отражает смысл всего текста.

c) **SentenceBERT** (SBERT)

SentenceBERT — это модифицированная трансформер-модель. Она базируется на архитектуре BERT, который сам является трансформером-энкодером (подробнее см. гл. 1.3).

2) В контексте задачи определения семантической близости текстов наиболее известной метрикой сходства считается косинусное сходство [4]. Мы можем оценить, насколько два текста близки по содержанию, вычисляя косинус угла между их векторными представлениями. Для двух векторов A и B косинусное сходство вычисляется с использованием их скалярного произведения и норм векторов следующим образом:

$$S = \cos(\theta) = \frac{A * B}{||A|| * ||B||}.$$

Матрица сходства S содержит значения сходства между векторными признаками всех пар текстов в корпусе. Таким образом, элемент S_{ij} представляет степень сходства между текстами i и j.

3) Пороговое значение определяется с использованием стандартного отклонения для матрицы сходства

$$\text{threshold} = \text{avg}(S) + \alpha \times \sigma(S),$$

где S — матрица сходства, σ — стандартное отклонение, α — настраиваемый параметр, к примеру 0.75.

На этапе сравнения текстов по матрице сходства происходит сопоставление величин сходства текстов с пороговым значением. Для каждого элемента матрицы сходства определяется, превышает ли он пороговое значение. Если это так, то два текста считаются похожими между собой и, следовательно, относятся к одному классу.

4) Тестирование проводится следующим образом. Из каждого набора данных извлекаются тысячи записей. Алгоритм классификации считается сработавшим верно, если значение сходства для двух текстов превышало порог, и оба текста принадлежали одному классу. Также верно и обратное: если значение сходства для двух текстов ниже порогового значения, то тексты относятся к разным классам. Во всех других случаях фиксируется ошибка работы алгоритма.

Точность работы алгоритма оценивается по формуле:

$$acc = \frac{N\_texts\_correctly\_classified}{N\_texts},$$

то есть как отношение количества правильно классифицированных текстов в тестовом наборе данных к N_texts — общему количеству текстов [1].

1.2 Метрики качества

Рассмотрим метрики, позволяющих автоматически оценивать качество перевода текстов и, соответственно, качество текстовых преобразований:

поиска перевода, перефразирования, summarization и других задач, в том числе нахождения схожих текстов

1. **BLEU** — метрика n-граммной точности, одна из первых автоматических мер для машинного перевода;

2. **ROUGE** — семейство метрик полноты и точности на уровне n-грамм и фрагментов текста, широко используемых в оценке summarization;

3. **METEOR** — учитывает синонимию, порядок слов и штрафы за разрывы, повышая корреляцию с человеческой оценкой;

4. **ChrF++** — символьно- и n-граммная метрика, хорошо работающая на морфологически богатых языках;

5. **BertScore** — современный подход на основе контекстных embeddings от моделей BERT, позволяющий учитывать семантическую близость.

Также обсудим человеческую оценку как золотой стандарт: шкалы, критерии и способы объединения результатов нескольких оценщиков. Понимание сильных и слабых сторон каждой метрики поможет выбрать оптимальный набор инструментов для вашей задачи и правильно интерпретировать полученные результаты.

1. **BLEU** (Bilingual Evaluation Understudy) – это отношение количества совпадающих слов к общему количеству слов в предложении гипотезы (traduction), ориентированная на precision. Эта метрика измеряет точность совпадения n-грамм перевода-кандидата с n-граммами эталона, с учётом штрафа за слишком короткий перевод.
Шаги вычисления BLEU:
1) Вычислим точность (precision) по n-граммам, т. е. отношение «сколько из n-грамм кандидата встречаются в reference» к «общему числу n-грамм в кандидате»

$$p_n = \frac{\sum_{n\text{-грамм} \in \text{cand}} \min(\text{count}_{\text{cand}}(n\text{-грамм}),\ \text{count}_{\text{ref}}(n\text{-грамм}))}{\sum_{n\text{-грамм} \in \text{cand}} \text{count}_{\text{cand}}(n\text{-грамм})},$$

где

- n — порядок n-грамм (например n = 1 — униграммы, n = 2 — биграммы и т. д.),
- cand — генерируемый текст (являющийся кандидатом),
- ref — эталонный (reference) текст,
- $\text{count}_{\text{cand}}$ (n-грамм) — количество вхождений данной n-граммы в сгенерированном тексте,
- $\text{count}_{\text{ref}}$ (n-грамм) — количество вхождений данной n-граммы в эталонном тексте.

2) BP (Brevity Penalty) – штраф за краткость который равен

$$\text{BP} = \begin{cases} 1, & \text{если } c > r, \\ e^{1-\frac{r}{c}}, & \text{если } c \leq r, \end{cases}$$

где |c| и |r| — длины сгенерированного и эталонного текстов.

3) Собственно BLEU

$$\text{BLEU} = \text{BP} \times \exp\Big(\sum_{n=1}^{N} w_n \ln(p_n)\Big),$$

где

- N — максимальный порядок n-грамм, обычно берут N=4,
- wn — весовая доля точности по n-граммам (часто wn=1/N).

Совпадения по юниграммам, как правило, измеряют адекватность, в то время как совпадения по более длинным n-граммам учитывают беглость речи.

Преимущества:

- Широко распространена: BLEU является одной из самых популярных и часто используемых метрик для оценки качества машинного перевода.

- Простота вычисления: Метрика основана на совпадениях n-грамм между переводом модели и эталонным переводом, что делает ее относительно простой для реализации.

- Объективность: Предоставляет стандартизированную числовую оценку, позволяющую сравнивать различные модели и алгоритмы.

Недостатки:

- Не учитывает синонимы и парафразы: BLEU полагается на точные совпадения слов, игнорируя возможность передачи смысла другими словами.

- Чувствительна к длине перевода: Метрика может штрафовать переводы, которые по длине отличаются от эталонного, что не всегда отражает качество перевода.

- Игнорирует порядок слов и грамматику: Не оценивает грамматическую правильность или стилистическую связность перевода.

- Может не коррелировать с человеческой оценкой: в некоторых случаях высокое значение BLEU не соответствует высокому качеству по мнению людей.

- Применение: BLEU измеряет точность совпадения n-грамм между сгенерированным текстом и эталоном. Это делает его пригодным для оценки степени поверхностного сходства между двумя текстами.
- Ограничения: Поскольку BLEU фокусируется на точных совпадениях слов и фраз, он может не улавливать семантическое сходство, если тексты выражены разными словами.

2. **ROUGE-N, ROUGE-L, ROUGE-W, ROUGE-S**

ROUGE основывается на подсчёте перекрытий N-грамм. Формула акцентирует внимание на пересечении N-грамм и униграмм между генерируемым и эталонным текстами, включая учёт порядка слов.

1) **ROUGE-N** – измеряет полноту n-грамм эталона в генерируемом тексте (recall-ориентированная)

$$\mathrm{ROUGE-N} = \frac{\sum_{n\text{-грамм} \in \mathrm{ref}} \min(\mathrm{count}_{\mathrm{cand}}(n\text{-грамм}),\ \mathrm{count}_{\mathrm{ref}}(n\text{-грамм}))}{\sum_{n\text{-грамм} \in \mathrm{ref}} \mathrm{count}_{\mathrm{ref}}(n\text{-грамм})}.$$

2) **ROUGE-L** - вычисляет самую длинную общую подпоследовательность (LCS - Longest Common Subsequence) между выводом (cand) и ref

где

- RLCS — recall-компонента: отношение длины LCS к длине эталона |r|,
- PLCS — precision-компонента: отношение длины LCS к длине кандидата |c|,
- β — параметр, задающий относительный вес recall и precision в итоговой F-мере; при β = 1 получается классическая F1-мера.

3) **ROUGE-W** (Weighted LCS) – базируется на взвешенной (Weighted) самой длинной общей подпоследовательности LCS (WLCS), где цепочки последовательных совпадений получают возрастающий вес

а) Взвешенная LCS

Пусть в LCS найдена подпоследовательность длины m, разбитая на t чанков подряд идущих совпадений длины $l_1, l_2, \ldots, l_t$. Тогда

$$\mathrm{WLCS}(c, r) = \sum_{i=1}^{t} \sum_{k=1}^{l_i} \mathrm{weight}(k) \quad \text{, где } \mathrm{weight}(k) = k^{\alpha}.$$

Здесь

- c, r — сгенерированный и reference тексты,
- l_i — длина i-го чанка подряд идущих совпавших символов/слов,
- α — гиперпараметр (обычно α=1.2 – 1.5),
- веса растут по закону k^{α}.

b) Precision и Recall по WLCS

$$P_W = \frac{\mathrm{WLCS}(c,r)}{|c|}, \quad R_W = \frac{\mathrm{WLCS}(c,r)}{|r|},$$

где |c|, |r| — длины (число токенов или символов) кандидата и reference.

3) **ROUGE-W** (F-мера)

$$\mathrm{ROUGE-W}_\beta = \frac{(1+\beta^2)\, P_W\, R_W}{R_W + \beta^2 P_W},$$

зачастую β = 1 (обычная F1-мера).

4) **ROUGE-S** (Skip-Bigrams) опирается на skip-bigrams — упорядоченные пары слов, допускающие промежутки.

1) Skip-биграммы (упорядоченные пары)

$$S(c) = \{\, (w_i, w_j) \mid 1 \le i < j \le |c| \,\}$$

– все пары слов из кандидата с сохранением порядка (может задаваться ограничение на максимальный «скачок»).

2) ROUGE-S (recall-ориентированная)

$$\mathrm{ROUGE-S} = \frac{|\, S(c) \,\cap\, S(r) \,|}{|\, S(r) \,|},$$

где

- |S(c)∩S(r)| — число совпавших skip-биграмм,
- |S(r)| — общее число skip-биграмм в reference.

5) **ROUGE-SU**

Расширение с учётом 1-грамм:

$$\mathrm{ROUGE-SU} = \frac{|S(c) \cap S(r)| + |U(c) \cap U(r)|}{|S(r)| + |U(r)|},$$

где U(c), U(r) — множества униграмм (по словам) в кандидате и reference.

Преимущества:

- Гибкость: Различные варианты метрики позволяют оценивать тексты с разных точек зрения.
- Хорошо подходит для summarization: Первоначально разработана для оценки автоматического реферирования (autosummarization) — это задача создания краткого содержания текста с сохранением ключевой информации.
- Учитывает полноту: ориентация на recall позволяет оценить, насколько полно сгенерированный текст отражает содержание эталона.

Недостатки:

- Не учитывает синонимы и перефразирования: Как и BLEU, ROUGE полагается на точные совпадения слов или фраз.
- Чувствительна к длине текста: Длинные тексты могут получать завышенные оценки из-за большего числа возможных совпадений.
- Меньшая корреляция с человеческой оценкой в машинном переводе: ROUGE лучше подходит для оценки summarization, чем для перевода.

- Применение: ROUGE, особенно ROUGE-L (Longest Common Subsequence), широко используется для оценки автоматической summarization, где важно измерить, насколько хорошо сгенерированный текст передает основное содержание исходного текста.
- Ограничения: Как и BLEU, ROUGE основывается на совпадениях n-грамм и может не учитывать перефразирование или семантическое сходство, выраженное различными словами.

3. **METEOR** (Metric for Evaluation of Translation with Explicit Ordering)
METEOR учитывает лексическую семантику и использование стемминга. Формула учитывает точные совпадения, синонимы и стемминг. Результаты повышаются благодаря учёту перестановок слов. Показатель основан на среднем гармоническом значении unigram precision и recall, при этом значение recall выше, чем precision. В METEOR также есть понятие "Штраф за фрагментацию", которое учитывает перекрытие не только униграмм, но и

фрагментов (последовательных слов) для установления определенного уровня упорядоченности.

Шаги расчёта METEOR:

1) Precision и Recall (с учётом выравнивания единиц — слов или их синонимов)

$$P = \frac{\text{количество совпавших юнитов}}{\text{число юнитов в cand}}, \quad R = \frac{\text{количество совпавших юнитов}}{\text{число юнитов в ref}}.$$

2) Гармоническое среднее (Harmonic mean) с весом 9:1 в пользу Recall

$$F_{\text{mean}} = \frac{10\,P\,R}{R + 9P}.$$

3) Penalty за фрагментацию (разрыв непрерывных совпадений)

$$p = 0.5\left(\frac{c}{u_m}\right)^3,$$

где c — число групп n-gram, а u_m — количество n-грамм, которые объединили в группы.

4) Итоговый показатель качества METEOR-скор

$$\text{METEOR} = F_{\text{mean}} \times (1 - \text{Penalty}).$$

Преимущества:

- Учитывает синонимы и стемминг: METEOR использует лексические ресурсы, такие как WordNet, для распознавания синонимов и разных форм одного слова.
- Сбалансированная оценка: Комбинирует точность (precision) и полноту (recall) с использованием гармонического среднего, придавая больший вес полноте.
- Более высокая корреляция с человеческой оценкой: в ряде исследований METEOR показывает лучшую согласованность с оценками людей по сравнению с BLEU.

Недостатки:

- Зависимость от лексических ресурсов: Требует наличие обширных словарей и баз синонимов, которые могут быть недоступны для некоторых языков.
- Большая вычислительная сложность: Более сложные алгоритмы сопоставления увеличивают время расчета.
- Может переоценивать совпадения: За счет учета синонимов и морфологических вариаций может увеличиваться вероятность ложных совпадений.

- Применение: METEOR учитывает синонимы и морфологически связанные слова, что делает его более чувствительным к семантическому сходству между текстами.
- Ограничения: Хотя METEOR лучше улавливает сходство, чем BLEU и ROUGE, он все же зависит от наличия лексических ресурсов и может не полностью отражать глубинное семантическое соответствие.

4. **ChrF++** (Character-level F-score)

ChrF++ сочетает F-меру по символьным n-граммам (ChrF) и по словным 1- граммам (WordF1). Chrf++ работает на уровне символов, а не на уровне слов, используемых в BLEU. Он рассматривает последовательности символов (символьные n-граммы) в сгенерированном тексте и сравнивает их с таковыми в справочном тексте. Этот подход позволяет ChrF++ фиксировать морфологические и структурные аспекты текста, делая его более устойчивым к изменениям порядка слов.

a) ChrF (для symbols n-грамм до порядка n)

$$\mathrm{ChrF}_\beta = \frac{(1+\beta^2)\, R_{\mathrm{chr}}\, P_{\mathrm{chr}}}{R_{\mathrm{chr}} + \beta^2 P_{\mathrm{chr}}},$$

где P_{chr} и R_{chr} — precision/recall symbols n-грамм, β обычно 2.

b) Word-F1 — обычная F1-мера по словным 1-граммам.

К примеру нужно оценить метрику между следующими reference и кандидатом:

Reference Text: "белка"

Machine Text: "блка"

Шаги вычисления ChrF:

1) N-граммы символов: вычисляем n-граммы на уровне символов для эталонного и сгенерированного текста. В этом примере мы рассмотрим униграммы (отдельные символы), биграммы (пары символов) и триграммы (тройки символов).

- Reference Unigrams: [“б”, “е”, “л”, “к”, “а”] / Machine-Unigrams: [“б”, “л”, “к”, “а”]
- Reference Bigrams: [“бе”, “ел”, “лк”, “ка”] / Machine-Generated Bigrams: [“бл”, “лк”, “ка”]
- Reference Trigrams: [“бел”, “елк”, “лка”] / Machine-Generated Trigrams: [“блк”, “лка”]

2) Перекрытие symbols N-грамм: Подсчитывает количество перекрывающихся symbols n-грамм между эталонным и машинным текстом.

- Overlapping Unigrams: [“б”, “л”, “к”, “а”]
- Overlapping Bigrams: [“лк”, “ка”]
- Overlapping Trigrams: [“лка”]

3) Вычисление ChrF++: Используем эти перекрывающиеся n-граммы для расчета F-score для каждой длины n-граммы

F-score for Unigrams: Precision (P1) = 4 / 4 = 1.0; Recall (R1) = 4 / 5 = 0.8

F1−Score (F1) = 2 * (P1 * R1)/(P1 + R1) = 0.89

F-score for Bigrams (ChrF++-2): Precision (P2) = 2 / 3 = 0.66;

Recall (R2) = 0.5, F2−Score (F2) = 2 * (P2 * R2)/(P2 + R2) = 0.57

F-score for Trigrams (ChrF++-3): Precision (P3) = 1 / 1 = 1.0;

Recall (R3) = 1 / 3 = 0.3333, F3−Score (F3) = 2 * (P3 * R3)/(P3 + R3) = 0.89

Преимущества:

- Подходит для морфологически богатых языков: символьные n-граммы эффективно улавливают сходства в языках со сложной морфологией.

- Чувствительность к небольшим вариациям: учитывает частичные совпадения слов, что может быть полезно при оценке опечаток или незначительных разночтений.
- Хорошая корреляция с человеческой оценкой: в исследованиях ChrF и ChrF++ показывают высокую согласованность с человеческими суждениями.

Недостатки:
- Менее интерпретируемы: Результаты на уровне символов могут быть сложнее для интуитивного понимания.
- Возможность переоценки незначительных совпадений: символьные совпадения могут не всегда отражать реальное качество перевода или генерации текста.

- Применение: эти метрики основаны на совпадениях symbols n-грамм и могут быть полезны для сравнения текстов на уровнях, чувствительных к морфологии и орфографии.
- Ограничения: они менее эффективны для улавливания семантического сходства на уровне предложений или абзацев.

5. **BERT-score** (Bilingual evaluation understudy)

BERTScore использует предварительно обученные контекстные embeddings (векторные представления) из трансформера BERT и сопоставляет слова в предложениях-кандидатах и reference по косинусному сходству. Обычно берутся embeddings из последних слоёв модели. Было показано, что он коррелирует с человеческим суждением при оценке на уровне предложения и системы.

Шаги:

1) Пусть e_i^{cand} и e_j^{ref} — контекстные embeddings токенов.

2) Для каждого токена в кандидате ищется наиболее семантически близкий токен в эталоне (и наоборот) через косинусное сходство

$$cos(v_c, v_r) = \frac{v_c \cdot e_r}{||\, e_c \,|| \cdot ||\, e_r \,||}$$

3) Precision (P) - усреднённое максимальное сходство токенов кандидата с токенами эталона. Оценивает, насколько точно кандидат передаёт смысл эталона

$$P = \frac{1}{|\text{cand}|} \sum_{i=1}^{|\text{cand}|} \max_j \cos\left(\mathbf{e}_i^{\text{cand}}, \mathbf{e}_j^{\text{ref}}\right).$$

4) Recall (R) – усреднённое максимальное сходство токенов эталона с токенами кандидата. Оценивает, насколько полно кандидат покрывает смысл эталона

$$R = \frac{1}{|\text{ref}|} \sum_{j=1}^{|\text{ref}|} \max_i \cos\left(\mathbf{e}_i^{\text{cand}}, \mathbf{e}_j^{\text{ref}}\right).$$

5) F-score (β = 1) – Гармоническое среднее между P и R

$$\text{BertScore} = \frac{2PR}{P+R}.$$

Ключевые улучшения:

1. Взвешивание через IDF

Чтобы снизить влияние частых слов (например, артиклей), токены взвешиваются с помощью Inverse Document Frequency (IDF).
Например, слова "the" получают низкий вес, а редкие слова (ключевые термины) — высокий.

2. Базовая нормализация

BERTScore калибруется на корпусе случайных предложений, чтобы диапазон значений был ближе к 0–100 (как BLEU). Это улучшает интерпретируемость.

Преимущества:
- Использует контекстуальные embeddings: основан на предобученных моделях трансформеров (например, BERT), что позволяет учитывать контекст и семантическое значение слов.

- Улавливает синонимы и перефразирования: благодаря embeddings, метрика может распознавать различные формулировки одного и того же смысла.
- Высокая корреляция с человеческой оценкой: в ряде исследований демонстрирует превосходство над традиционными метриками в соответствии с оценками людей.

Недостатки:
- Высокие вычислительные затраты: Требует значительных ресурсов для вычисления embeddings, особенно на больших наборах данных.
- Зависимость от выбранной модели: Качество оценки может варьироваться в зависимости от используемой предобученной модели и ее актуальности для конкретного языка или домена.
- Ограниченная интерпретируемость: Результаты сложно анализировать и объяснять без глубокого понимания работы моделей трансформеров.

- Применение: BERTScore использует контекстуальные embeddings слов из моделей типа BERT для оценки сходства между текстами на семантическом уровне. Это делает его особенно пригодным для задач оценки сходства текстов.
- Преимущества: Благодаря учету контекста и семантики, BERTScore может улавливать сходство между текстами, даже если они выражены разными словами или фразами.

6. **Человеческая оценка**

Преимущества:
- Глубокое понимание текста: люди способны оценивать смысл, тон, стилистическую соответствие и другие аспекты, недоступные автоматическим метрикам.
– Гибкость: могут адаптировать критерии оценки под конкретные задачи или требования проекта.
- Эталон оценки: человеческое восприятие является окончательным арбитром качества в задачах естественного языка.

Недостатки:

- Ресурсоемкость: процесс требует времени и финансовых затрат на привлечение и обучение оценщиков.
- Субъективность: разные люди могут по-разному оценивать один и тот же текст, что приводит к вариативности результатов.
- Сложности масштабирования: трудно применять для оценки больших объемов данных или при необходимости быстрой обратной связи.

Вывод: таким образом не все метрики из перечисленных, ограничены только оценкой качества машинного перевода. Некоторые из них, особенно BERTScore, могут быть эффективно применены для задачи оценки сходства текстов.

Проиллюстрируем статистику применения метрик машинного обучения с 2010 по 2020 год (рис. 1). Видно, что не самая лучшая, но самая известная метрика BLUE используется в подавляющем количестве случаев.

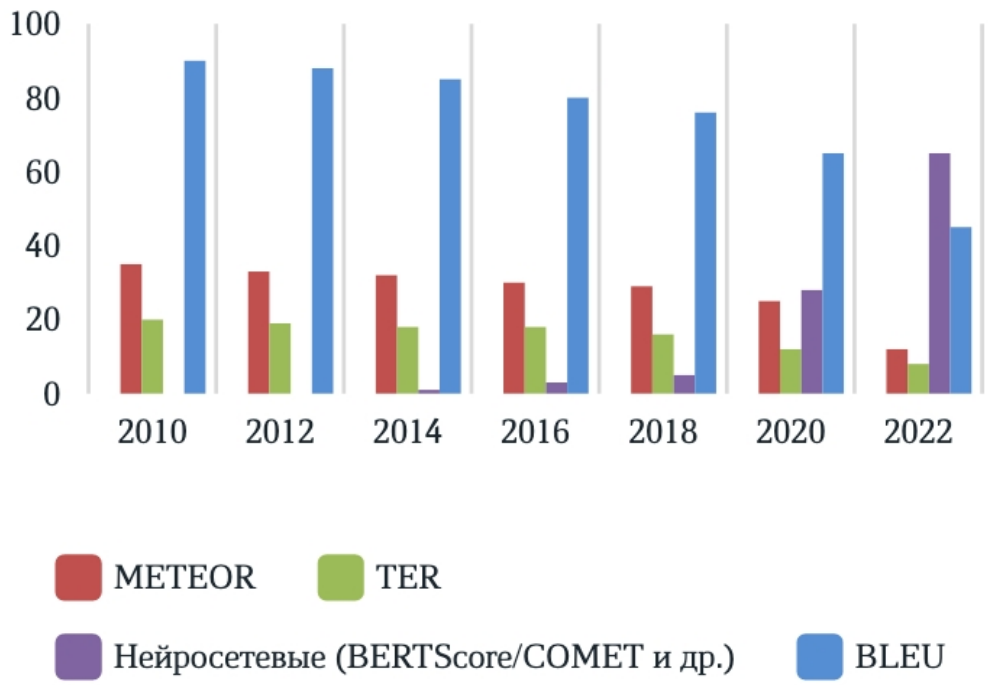

Рис. 1. Популярность метрик

1.3 Устройство и особенности трансформеров

1. **Введение**

Трансформеры представляют собой революционную архитектуру нейронных сетей, которая изменила подходы к обработке последовательностей данных, особенно в области обработки естественного языка (NLP). Введенные в 2017 году в работе "Attention is All You Need" (Vaswani et al.) [5], трансформеры показали превосходные результаты в задачах машинного перевода, summarization текста, ответа на вопросы и многих других.

Основным новшеством трансформеров является использование механизма внимания (Attention) без необходимости в рекуррентных или сверточных слоях для моделирования зависимостей в последовательностях. Это позволяет эффективно обрабатывать долгие зависимости и значительно ускоряет обучение благодаря параллельным вычислениям.

2. **Архитектура трансформера**

Архитектура трансформера состоит из двух основных компонентов:

- Кодер (Encoder)
- Декодер (Decoder)

Оба компонента состоят из нескольких однотипных слоев, снабженных механизмами внимания и feed-forward нейронными сетями (3 из [5]).

Общий «путь» данных выглядит следующим образом:

Токенизация → эмбединги + позиционное кодирование →

Энкодер (N× self-attention + FFN) →

Декодер (N×: masked self-attention → cross-attention → FFN) →

Линейная проекция → softmax → авторегрессивный вывод

1) Кодер и декодер

- Кодер принимает входную последовательность и преобразует ее в скрытое представление.

- Декодер использует это скрытое представление и предыдущие выходы для генерации выходной последовательности.

2) Слои кодера

Каждый слой кодера состоит из двух основных :

a) Multi-Head Self-Attention

b) Полносвязная feed-forward нейронная сеть

Между sublayers используется механизм добавления остаточных связей (Residual Connections) и слой нормализации (стандартом является Layer Norm).

3) Слои декодера

Слои декодера аналогичны слоям кодера, но содержат дополнительный sublayer внимания к выходу кодера:

a) Masked Multi-Head Self-Attention

b) Cross Attention

c) Полносвязная feed-forward нейронная сеть

3. **Механизм внимания (Attention)**

Механизм внимания позволяет модели фокусироваться на различных частях входной последовательности при обработке каждого элемента. Это особенно важно для понимания контекста и взаимосвязей в последовательности.

a) Q, K, V Self-Attention

Self-Attention вычисляет внимание по отношению к самой последовательности, позволяя модели учитывать взаимосвязи между разными позициями в последовательности.

Начнем с входных данных (например, слов), которые преобразуются в векторы.

Пусть $X = \{x_1, x_2, ..., x_n\} \in R^{T \times dmodel}$ – входная матрица, состоящая из последовательности векторов, где T — длина последовательности (число токенов), а dmodel — размер скрытого состояния.

Ключи, запросы и значения: каждый вектор проецируется тремя различными линейными слоями в три представления:

- Ключи $K = XW_K$,
- Запросы $Q = XW_Q$,
- Значения $V = XW_V$,

где

- $WQ,WK,WV \in R^{dmodel \times dk}$ (обычно $d_k = d_{model}/h$ для h голов),
- результат Q, K, V $\in R^{T \times dk}$.

Для каждого токена i (строка Qi) и каждого токена j (строка K_j) считаем скалярное произведение, масштабируемое на d_k

$$e_{ij} = \frac{Q_i \cdot K_j}{\sqrt{d_k}} = \frac{1}{\sqrt{d_k}} \sum_{m=1}^{d_k} Q_{i,m} K_{j,m}.$$

Это даёт матрицу скорингов $E=QK^T \in R^{T \times T}$.

К «сырым» скорингам e_{ij} применяется по-строчный softmax, чтобы получить неотрицательные веса, сумма которых по j равна 1

$$w_{ij} = \mathrm{softmax}_j(e_{ij}) = \frac{\exp(e_{ij})}{\sum_{k=1}^{T} \exp(e_{ik})}.$$

В матричной форме

$$W = \mathrm{softmax}(E) \quad \text{где } W_{i,:} = \mathrm{softmax}(E_{i,:}).$$

В итоге для каждого токена i берётся взвешенная сумма значений Vj с весами wij

$$\mathrm{Attention}(Q,K,V)_i = \sum_{j=1}^{T} w_{ij} V_j.$$

Или в матричной форме

$$\mathrm{Attention}(Q,K,V) = W\,V = \mathrm{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right) V,$$

что и даёт выходную матрицу из множества $R^{T \times dk}$.

Без деления на $\sqrt{d_k}$ величины скалярных произведений при большой размерности dkdk могли бы «взорваться», дав очень большие аргументы

softmax и приводя к «забиванию» распределения (почти единичный вес для одного элемента и ноль для остальных). Деление стабилизирует градиенты и делает обучение более устойчивым.

b) Multi-Head Attention

Multi-Head Attention расширяет способность модели фокусироваться на разных позициях путем использования h "голов" внимания. Каждая голова самостоятельно (параллельно) выполняет операцию внимания, после чего результаты объединяются (рис. 2).

$$\text{head}_\ell = \text{Attention}(XW_Q^\ell,\ XW_K^\ell,\ XW_V^\ell), \quad \ell = 1 \dots h.$$

$$\text{MultiHead}(X) = \text{Concat}(\text{head}_1, \dots, \text{head}_h)\, W_O,$$

где W_O є Rh $d^{k \times dmodel}$

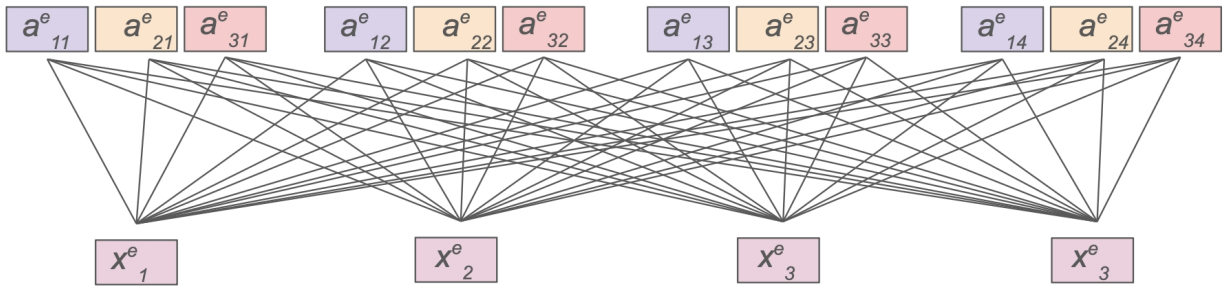

Рис. 2. Multi-Head Attention с 3-мя головами

c) Cross-Attention (Encoder-Decoder Attention)

Cross-Attention используется в каждом слое декодера, чтобы выходные представления декодера могли получать ("спрашивать") информацию у выходов энкодера. В отличие от Self-Attention (где Q, K и V приходят из одного и того же источника), здесь

Определения:

$X_{dec} \in \mathbb{R}^{T_{dec} \times d_{model}}$ – матрица скрытых состояний декодера,

$X_{enc} \in \mathbb{R}^{T_{enc} \times d_{model}}$ – матрица выходов энкодера,

$W_Q^C, W_K^C, W_V^C \in \mathbb{R}^{d_{model} \times d_k}$ – проекционные матрицы.

Вычисление проекций:

$$Q^C = X_{dec} W_Q^C,\ K^C = X_{enc} W_K^C,\ V^C = X_{enc} W_V^C$$

Дальше точно так же, как в Self-Attention, считаем «сырые» скоринги и масштабируем их (получаем веса):

$$E^C = \frac{Q^C (K^C)^{top}}{\sqrt{d_k}},\ W^C = softmax\left(E^C\right)$$

И финальный выход cross-attention:

$$CrossAttn(Q^C, K^C, V^C) = W^C V^C \in \mathbb{R}^{T_{dec} \times d_k}$$

В многоголовом варианте для каждой головы i=1, ..., h свои проекции:

$$head_i^C = Attention\left(X_{dec} W_Q^{C,i},\ X_{enc} W_K^{C,i},\ X_{enc} W_V^{C,i}\right)$$

Объединение голов:

$$MultiHead^C = Concat\left(head_1^C, \ldots, head_h^C\right) W_O^C$$

4. Позиционное кодирование (Positional Encoding)

Так как трансформеры не используют рекуррентность или сверточные слои, они не имеют встроенного механизма для учета позиции элементов в последовательности. Для решения этой проблемы используют позиционное кодирование , которое добавляется к входным embeddings.

Позиционное кодирование может быть задано синусоидальными функциями:

$$p_{ti}^e = \begin{cases} p_{2i}^e = \sin\left(\frac{t}{10000^{\frac{2i}{emb\_dim}}}\right), \\ p_{2i+1}^e = \cos\left(\frac{t}{10000^{\frac{2i}{emb\_dim}}}\right), \end{cases}$$

где

- t – позиция слова в последовательности
- i – индекс размерности embedding
- emb_dim – размерность embedding

5. **Механизм маскирования**

В декодере используется маскирование последовательности для предотвращения "заглядывания вперед" (т.е. модель не должна учитывать будущие слова при генерации текущего). Маска является матрицей, у

которой все элементы выше главной диагонали равны нулю, которая накладывается на матрицу внимания (Decoder). Покажем механизм работы двух видов маскирования: Padding-mask и Causal-mask

1) Padding-mask

Используется для игнорирования токенов-паддингов (нулевых или пустых позиций) в последовательностях разной длины. Паддинговые токены маскируются таким образом, чтобы они не оказывали влияния на финальные вычисления внимания. Это позволяет эффективнее обрабатывать пакеты (батчи) данных с последовательностями разных длин.

Механизм внимания обычно реализуется с использованием скалярного произведения между векторами запроса (Query, Q) и ключа (Key, K), нормализованного с помощью softmax. Формула для вычисления внимания без маски:

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V,$$

где V — вектор значений (Value), а d_k — размерность ключа.

Чтобы обеспечить игнорирование паддинговых токенов, мы добавляем матрицу маскирования M, где элементы, соответствующие паддинговым позициям, принимают значение -∞. Таким образом, механизм внимания не будет учитывать эти позиции. Формула с учетом маски становится

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k} + M}\right)V,$$

где элементы матрицы M определены как:

$$M_{i,j} = \begin{cases} 0, & \text{если позиция j не является паддингом,} \\ -\infty, & \text{если позиция j является паддингом.} \end{cases}$$

При добавлении матрицы M, значения, соответствующие паддинговым токенам, становятся настолько малыми после применения softmax, что их влияние на итоговый результат становится незначительным. Таким образом, внимание сосредотачивается только на значимых токенах последовательности.

Эта техника помогает трансформеру обрабатывать последовательности разной длины, не искажая результат ненужной информацией.

2) Causal-mask

Для декодера мы хотим, чтобы при предсказании позиции i модель не «видела» токены j>i. Так для предотвращения утечек информации из будущих токенов при генерации последовательностей на этапе обучения модели используется Causal-mask

Построим матрицу

$$M_{i,j} = \begin{cases} 0, & j \le i \\ -\infty & j > i \end{cases}.$$

Тогда вместо обычных скорингов

$$E = \frac{Q(K)^{top}}{\sqrt{d_k}}$$

мы берём

$$E' = E + M,\ W = softmax\left(E'\right).$$

За счёт $-\infty$ для запрещённых позиций softmax присвоит им нулевой вес.

Где и когда применяется:

- Masked Multi-Head Self-Attention только в декодере, в первом sub-layer каждого слоя.
- Cross-Attention и обычный self-attention в энкодере маски не требуют (используется full self-attention).

Отличия между данными видами маскирования заключается в том, что

- Causal-mask используется для управления направлением потока информации, предотвращая утечку из будущего.

- Padding-mask используется для игнорирования бесполезной информации в фиктивных токенах, созданных для выравнивания последовательностей.

6. **Нормализация слоя**

Слой нормализации (Layer Normalization) применяется для стабилизации и ускорения обучения. Он нормализует выходы sublayers по среднему и

стандартному отклонению элементов каждого embeddings, а не элементов на одних и тех же позициях всех batch embeddings (группе примеров), как это происходит в Batch Normalization.

$$\mu_i = \frac{1}{emb\_size} \sum_{i=1}^{emb\_size} x_i, \quad \sigma_i^2 = \frac{1}{m} \sum_{i=1}^{emb\_size} (x_i - \mu_i)^2, \quad \text{где } 1 \leq i \leq seq\_len$$

$$normalized\_x = \frac{(x - \mu)}{\sigma + \varepsilon} * \gamma + \beta$$

Здесь x — входной вектор, μ и σ — среднее и стандартное отклонение по элементам этого вектора, а γ и β — параметры, которые обучаются вместе с остальной моделью. γ отвечает за масштабирование, а β за сдвиг.

В оригинальной архитектуре Transformer, предложенной в статье "Attention is All You Need", использовалась пост-нормализация (post-norm) для слоя LayerNorm. Это означает, что LayerNorm добавлялся после слоя self-attention и feed-forward, то есть

x = LayerNorm(x + Sublayer(x))

Однако в более современных реализациях, таких как BERT и GPT-3, часто применяется пред-нормализация (pre-norm). В этом случае LayerNorm используется до слоя self-attention и feed-forward, обеспечивая лучшую стабильность при обучении более глубоких моделей:

x = x + Sublayer(LayerNorm(x))

Пред-нормализация помогает улучшить качество эмбеддингов предложений и стабилизировать обучение глубоких моделей, поскольку нормализация входов слоя происходит заранее, нивелируя некоторые эффекты, которые могут привести к нестабильности градиентов и их взрывам
или исчезновению.

Помимо повышения устойчивости и скорости обучения, Layer Norm снижает чувствительность к выбору начальных параметров, а также предотвращает исчезновение или взрыв градиентов в очень глубоких моделях.

7. **Остаточные связи (Residual Connections)**

Остаточные связи позволяют избежать проблемы затухающих градиентов, облегчая обучение глубоких сетей. Они позволяют сигналу, то есть результату обработки информации, которая передаётся между нейронами, обходить один или несколько слоев без изменений.

Математически это можно выразить как:

$$Output = LayerNorm(X + Sublayer(X)),$$

где X – вход в слой, Sublayer(X) – выход из sublayer (например внимания или feed-forward сети).

8. **Feed-Forward слои**

Каждый слой трансформера содержит полносвязную feed-forward нейронную сеть, которая применяется по каждому позиционному embedding независимо и одинаково

$$FFN(x) = ReLU(xW_1 + b_1)W_2 + b_2,$$

где

- W_1, W_2 – матрицы весов,
- b_1, b_2 – смещения,
- ReLU – функция активации Rectified Linear Unit.

Иллюстративно полный декодерный слой записывается так:

$$\begin{aligned} x_1 &= LayerNorm(x + MaskedSelfAttn(x)) \\ x_2 &= LayerNorm(x_1 + MultiHeadCrossAttn(x_1, X_{enc})) \\ y &= LayerNorm(x_2 + FeedForward(x_2)) \end{aligned}$$

2 ПРАКТИЧЕСКАЯ ЧАСТЬ

2.1 Используемые данные

В данной работе использовали небольшие тексты для первоначального тестирования работы алгоритма нахождения схожих текстов.

Используем данные тексты для анализа

```
documents = [
    "Мама мыла раму и окно.",
    "Отец читал газету в кресле.",
    "Собака лает на прохожих.",
    "Кошка спит на подоконнике.",
    "Дети играют в футбол на стадионе.",
    "Мама готовит вкусный ужин.",
    "Отец ремонтирует автомобиль в гараже.",
    "Птицы поют на рассвете.",
    "Дети смотрят мультфильмы по телевизору.",
    "Собака и кошка дружат между собой."
]
```

Мы будем искать смысловые копии и сходства между этими текстами.

2.2 Предобработка текста

Для обработки русского текста мы можем использовать библиотеку spaCy с поддержкой русского языка или pymorphy2 для морфологического анализа. В данной задаче мы воспользуемся spaCy.

1) Лемматизация, удаление стоп-слов и нормализацию текста

- Лемматизация: приводим слова к начальной форме.
- Удаление стоп-слов: исключаем часто встречающиеся слова, которые не несут особого смысла (предлоги, союзы и т.д.).
- Нормализация текста: приводим текст к нижнему регистру.

```
[
    "мама мыла рама окно",
    "отец читать газета кресло",
    "собака лаять прохожий",
    "кошка спать подоконник",
    "ребёнок играть футбол стадион",
    "мама готовить вкусный ужин",
    "отец ремонтировать автомобиль гараж",
    "птица петь рассвет",
    "ребёнок смотреть мультфильм телевизор",
    "собака кошка дружат"
]
```

2) Векторизация текста

Применим TF-IDF векторизацию для преобразования текстов в числовой формат. TF-IDF (Term Frequency-Inverse Document Frequency): статистическая мера, отражающая важность слова в документе относительно корпуса документов.

Используем предобученные модели Word2Vec для русского языка из библиотеки gensim. Видим, что предобученная модель "word2vec-ruscorpora-300" для русского языка не может найти большинство слов из представленных текстов, поэтому данную модель можем считать неподходящей для нашей задачи с учётом рассматриваемых данных.

Используем модели глубокого обучения (трансформеры) Sentence Transformers (BERT). для получения embeddings предложений. Sentence Transformers: библиотека для получения embeddings предложений с использованием моделей BERT и их производных. Модель paraphrase-multilingual-MiniLM-L12-v2 подходит для многих языков, в том числе для русского.

2.3 Сравнение текстов и поиск похожих

Для каждой пары документов вычислим косинусное сходство. Косинусное сходство: метрика, измеряющая косинус угла между двумя векторами в пространстве; значения от -1 до 1. Матрица сходства содержит значения сходства между каждым парой документов. Написали функцию для вывода наиболее похожих документов для каждого подхода. Для каждого документа выводим два самых похожих документа по выбранной метрике.

2.4 Произвели алгоритм K-Means для кластеризации документов

Используем метод снижения размерности t-SNE для визуализации результатов кластеризации. t-SNE (t-distributed Stochastic Neighbor Embedding): метод нелинейного снижения размерности, часто используемый для визуализации высокомерных данных. Мы визуализируем кластеры в 2D пространстве.

Для модели Word2Vec кластеризацию не визуализируем по причине того, что слов из нашего датасета не нашлось в модели Word2Vec и все слова соответственно относились бы к одному и тому же кластеру, в чём нет смысла.

2.5 Произвели оценку качества кластеризации

Используем метрику силуэта для оценки качества кластеризации. Его формула

$$S(x_i) = \frac{B(x_i) - A(x_i)}{\max(B(x_i), A(x_i))}$$

где

$S(x_i)$ – коэффициент силуэта,

$A(x_i)$, — это среднее расстояние между x_i и объектами того же кластера,

B(xi) — это среднее расстояние между x_i и объектами следующего ближайшего кластера.

Коэффициент силуэта измеряет, насколько объект похож на свой кластер по сравнению с другими кластерами. Значения близкие к 1 указывают на хорошую кластеризацию.

В ходе вычисления коэффицинта Силуэта получили следующие значения:

Силует для TF-IDF = 0.06,

Силует для Sentence-BERT = 0.07.

Таким образом, получили более качествунную кластеризацию моделью Sentence-BERT.

Альтернатива:

• DBSCAN (сокращение от Density-Based Spatial Clustering of Applications with Noise) – это алгоритм кластеризации на основе плотности. В отличие от иерархических или «центроидных» (например, k-means) подходов, DBSCAN группирует объекты, «обнаруживая» области высокой плотности, отделённые «пробелами» (областями с малой плотностью).

Параметры: eps и min_samples:

– eps (эпсилон) – радиус окрестности, в которой мы ищем «соседей» для каждого объекта (точки).

– min_samples – минимальное количество точек, которое должно находиться внутри eps-окружности точки, чтобы считать эту точку «ключевой» (core point).

– Не требует заранее задавать число кластеров (в отличие от k-means).

– Умеет «выявлять» кластеры произвольной формы и хорошо работает с шумовыми точками.

– Устойчив к выбросам (outliers), так как шум автоматически не попадает в кластеры.

– При высокой размерности данных и выборе неудачных параметров eps и min_samples результат может быть некачественным или алгоритм становится менее эффективным.

– DBSCAN может плохо работать, если плотность данных слишком меняется в разных областях: одна и та же пара (eps, min_samples) может не подходить сразу для всех регионов.

• HDBSCAN (Hierarchical Density-Based Spatial Clustering of Applications with Noise)

– Это иерархический алгоритм кластеризации, основанный на плотности, «продолжение» классического DBSCAN.

– Подходит, когда кластеры могут иметь сложную форму (не обязательно сферическую, как у K-Means), а данные содержат выбросы или шум.

– Позволяет выделять «основные» кластеры и игнорировать разреженные участки данных.

– Может быть полезен при анализе текстовых данных, если они образуют неравномерные, «рваные» кластеры и при этом требуется автоматически определять количество кластеров.

• Faiss (Facebook AI Similarity Search)

– Это высокопроизводительная библиотека от Meta (Facebook) для поиска ближайших соседей («approximate nearest neighbor search») в больших наборах векторных представлений.

– Обычно Faiss не рассматривают как самостоятельный алгоритм кластеризации; однако его можно использовать для ускорения многих операций, связанных со сравнениями векторов (например, при поиске похожих документов или при построении собственных методов кластеризации, где часто требуется быстрый поиск ближайших соседей).

– Faiss лучше подходит для случаев, когда нужно работать с действительно большими наборами данных (миллионы и миллиарды объектов) и важна оптимизация по времени и памяти.

Помимо коэффициента силуэта, который мы уже упомянули ($S(x_i) = B(x_i) – A(x_i) / \max\{A(x_i), B(x_i)\}$), есть и другие метрики оценки качества кластеризации:

1) Davies–Bouldin Index (DBI)

– Отражает степень «похожести» кластеров друг на друга.

– По сути, индекс представляет собой среднюю степень близости (или «перекрытия») каждого кластера с ближайшим к нему кластером.

– Формально DBI определяется как

$$DBI = (1 / K) \sum(i=1..K) \max (j \neq i) ((\sigma_i + \sigma_j) / d(c_i, c_j)) ,$$

где K – число кластеров, c_i – центр i-го кластера, σ_i – среднее расстояние объектов i-го кластера до его центра, $d(c_i, c_j)$ – расстояние между центрами кластеров i и j.

– Чем меньше значение Davies–Bouldin Index, тем лучше качество кластеризации (меньше взаимного «перекрытия» кластеров).

2) Calinski–Harabasz Score (CH Score)

– Называется также «Variance Ratio Criterion».

– Рассчитывается как отношение межкластерной дисперсии к внутрикластерной (аналогично F-статистике в дисперсионном анализе):

$$CH = (B/W) \times (n – K) / (K – 1),$$

где n – количество объектов, K – количество кластеров, B – межкластерная дисперсия, W – внутрикластерная дисперсия.

– Чем выше это значение, тем более «правильным» считается разделение на кластеры (сильнее разделены центры кластеров и меньше внутренняя вариация в рамках каждого кластера).

Таким образом, для более всесторонней оценки кластеризации стоит использовать несколько показателей одновременно:

• Коэффициент силуэта (Silhouette Score) – насколько объекты «плотно» собираются по сравнению с ближайшим кластером.

• Davies–Bouldin Index – насколько кластеры перекрываются друг с другом.

• Calinski–Harabasz Score – соотношение между межкластерной и внутрикластерной вариацией.

2.6 Классификация текстов

Предположим, у нас есть метки классов для наших документов. Научим модель наивного байесовского классификатора [6] предсказывать эти классы и вычислим метрики (рис. 3), отражающие предсказательную способность нашей модели [7 – 11]

	precision	recall	f1-score	support
Семья	0.50	1.00	0.67	2
Животные	0.00	0.00	0.00	2
Дети	0.50	0.50	0.50	2
accuracy			0.50	6
macro avg	0.33	0.50	0.39	6
weighted avg	0.33	0.50	0.39	6

Рис. 3. Метрики классификации (иллюстрация вывода программы)

Где

– Precision – точность предсказаний положительного класса,

– Recall – полнота; сколько объектов положительного класса было найдено,

– F1-score – гармоническое среднее точности и полноты.

2.7 Обучение без учителя

Используем модель латентного размещения Дирихле (LDA) для выявления скрытых тем. LDA – это вероятностная модель [9, 10, 12], разработанная для моделирования тем в коллекциях документов. Она основана на предположении, что каждый документ может быть представлен как смесь различных тем, а каждая тема связана с распределением слов. Его принцип работы можно описать следующим образом:

1. **Подготовка данных**: Сначала тексты преобразуются в числовое представление. Это может быть, например, "мешок слов" (bag of words) или TF-IDF матрица, которая показывает, сколько раз каждое слово встречается в каждом документе.

2. **Определение количества тем**: Для применения LDA необходимо заранее определить количество тем, которые вы хотите выделить. Это может

быть достаточно сложной задачей, и выбор зависит от конкретных характеристик данных.

3. **Обучение модели**: LDA итеративно обрабатывает тексты, чтобы определить, какие темы присутствуют в документах и какие слова связаны с каждой темой. Алгоритм старается максимизировать вероятность появления слов в каждой теме и вероятность наличия тем в каждом документе.

Алгоритм LDA можно разбить на несколько шагов:

Шаг 1: Инициализация параметров.

Задается количество тем (гиперпараметр K) и другие параметры модели.

Шаг 2: Подготовка данных.

Документы преобразуются в числовое представление, например, с помощью "мешка слов" или TF-IDF матриц.

Шаг 3: Инициализация распределений.

Инициализируются распределения тем для документов и распределения слов для тем.

Шаг 4: Итеративный процесс.

Повторяется следующий процесс для нескольких итераций:

а) Для каждого слова в каждом документе вычисляется вероятность принадлежности к каждой теме, используя текущие распределения тем и слов.

б) На основе вероятностей слов в темах и вероятностей тем в документах пересчитываются распределения тем и слов.

Шаг 5: вывод результатов.

- По окончании итераций можно получить распределения тем для каждого документа и распределения слов для каждой темы.

Применив данную модель для наших текстов, получили вывод топ-слов для каждой темы (рис. 4):

```
Тема 1 Семья:
семейный, на, всех, ужин, лает, собака, парке, незнакомцев, саду, детском

Тема 2 Животные:
кататься, каруселях, семье, сказку, рассказывает, любят, дети, мама, всей, поют

Тема 3 Дети:
папа, орехи, белка, лесу, ходить, учится, говорить, малыш, играть, футбол
```

Рис. 4. Тематика

ЗАКЛЮЧЕНИЕ

- TF-IDF + Косинусное сходство: простой и быстрый метод, однако не учитывает семантическую близость слов.
- Word2Vec: учитывает семантическое сходство слов, однако при усреднении embeddings теряется контекст предложения.
- Sentence-BERT: предоставляет embeddings на уровне предложений, учитывая контекст и семантику, показывая лучшие результаты в измерении сходства.
- Метрики силуэта показывают, что модель Sentence-BERT лучше разделяет данные на кластеры.
- В классификации, использовав TF-IDF и наивный байесовский классификатор, получили удовлетворительные результаты для данной небольшой выборки.

В данной работе мы рассмотрели несколько подходов для поиска и группировки схожих текстов на русском языке без использования NLTK. Мы применили различные методы векторизации текстов и сравнили их эффективность. Результаты показывают, что современные модели глубокого обучения, такие как Sentence-BERT, превосходят традиционные методы в задаче определения семантического сходства текстов.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Sitikhu P. A Comparison of Semantic Similarity Methods for Maximum Human Interpretability // arXiv:1910.09129v2 [cs.IR]. – 2019. – Oct. – Access Mode: https://arxiv.org/pdf/1910.09129.pdf.

2. Prasetya D.D., Wibawa A.P., Hirashima T. The performance of text similarity algorithms. // International Journal of Advances in Intelligent Informatics vol. 4(1). – 2018. – P.63–69.

3. Jalilifard A. Semantic Sensitive TF-IDF to Determine Word Relevance in Documents. // arXiv:2001.09896v2 [cs.IR]. – 2021. – Jan. – Access Mode: https://arxiv.org/pdf/2001.09896.pdf.

4. Kaggle: News dataset from Lenta.Ru. – 2021. – Access Mode: https://www.kaggle.com/yutkin/corpus-of-russian-news-articles-from-lenta/.

5. Vaswani A. et al. Attention Is All You Need // Proceedings of the 31st International Conference on Neural Information Processing Systems (NeurIPS 2017). — 2017. — P. 6000–6010. — Access Mode: https://papers.nips.cc/paper/7181-attention-is-all-you-need

6. Kaggle: 7 topic data for text classification. – 2021. – Access Mode: https://www.kaggle.com/deepak711/4-subject-data-text-classification?select=Physics_Biology_Geography_Accounts+subject+training+data+ for+text+classification.

7. Raschka S. Naive Bayes and Text Classification I – Introduction and Theory // arXiv:1410.5329 [cs.LG]. – 2014.

8. Li Y., Khan M.A., Zhang X. A Comprehensive Review of Text Classification Algorithms // Journal of AI. – 2024. – Vol. 12, no. 3. – P. 45–62.

9. Jelodar H., Wang Y., Yuan C., Feng X., Jiang X., Li Y., Lv Y., Wang W., Huaiqiu W. Latent Dirichlet Allocation (LDA) and Topic Modeling: Models, Applications, a Survey // arXiv:1711.04305 [cs.LG]. – 2017.

10. Baeldung. Topic Modeling with Latent Dirichlet Allocation. Baeldung.com, 2020. URL: https://www.baeldung.com/lda.

11. Escalante H.J., Morales E.F., Vilalta R. Early Text Classification: a Naive Solution // arXiv:1509.06053 [cs.LG]. – 2015.

12. GeeksforGeeks. Topic Modeling Using Latent Dirichlet Allocation (LDA). GeeksforGeeks.org, 11 Jun 2024. URL: https://www.geeksforgeeks.org/nlp/topic-modeling-using-latent-dirichlet-allocation-lda/.

ПРИЛОЖЕНИЕ А

Ниже приведём код программы, результаты которой описаны в практической части:

```
!pip install -U spacy
!python -m spacy download ru_core_news_sm
!pip install gensim
!pip install -U sentence-transformers

# Наши тексты для анализа
documents = [
"Мама мыла раму и окно.",
"Отец читал газету в кресле.",
"Собака лает на прохожих.",
"Кошка спит на подоконнике.",
"Дети играют в футбол на стадионе.",
"Мама готовит вкусный ужин.",
"Отец ремонтирует автомобиль в гараже.",
"Птицы поют на рассвете.",
"Дети смотрят мультфильмы по телевизору.",
"Собака и кошка дружат между собой."
]

import spacy
# Загружаем модель русского языка
nlp = spacy.load('ru_core_news_sm')

# Функция для лемматизации текста
def preprocess(text):
```

```
doc = nlp(text.lower())
tokens = [token.lemma_ for token in doc if not token.is_punct and not token.is_stop]
return ' '.join(tokens)

# Применяем предобработку к нашим текстам
processed_documents = [preprocess(doc) for doc in documents]

# Применим TF-IDF векторизацию для преобразования текстов в числовой формат
from sklearn.feature_extraction.text import TfidfVectorizer

vectorizer = TfidfVectorizer()
tfidf_matrix = vectorizer.fit_transform(processed_documents)
# Используем предобученные модели Word2Vec для преобразования слов в embeddings

import gensim.downloader as api
from gensim.models import Word2Vec

# Загружаем предобученную модель для русского языка
model = api.load("word2vec-ruscorpora-300") # может занять некоторое время
# Функция для получения среднего вектора документа
import numpy as np

def preprocess_for_w2v(text):
doc = nlp(text.lower())
tokens = [token.text for token in doc if not token.is_punct and not token.is_stop]
return tokens
```

```
wv_documents = [preprocess_for_w2v(doc) for doc in documents]

def document_vector(doc):
unknown_words = [word for word in doc if word not in model.key_to_index]
if unknown_words:
print(f"Неизвестные слова: {unknown_words}")
doc = [word for word in doc if word in model.key_to_index]
if len(doc) > 0:
return np.mean([model[word] for word in doc], axis=0)
else:
return np.zeros(model.vector_size)

# Преобразуем наши документы в векторы
wv_vectors = np.array([document_vector(doc) for doc in wv_documents])
# Используем трансформеры для преобразования слов в embeddings
from sentence_transformers import SentenceTransformer
# Загружаем модель для русского языка
model_name = 'sentence-transformers/paraphrase-multilingual-MiniLM-L12-v2'
sbert_model = SentenceTransformer(model_name)
# Преобразуем документы в embeddings
sbert_embeddings = sbert_model.encode(documents)
# Для каждой пары документов вычислим косинусное сходство.

from sklearn.metrics.pairwise import cosine_similarity
# Для TF-IDF
tfidf_cosine_sim = cosine_similarity(tfidf_matrix)
# Для Word2Vec
wv_cosine_sim = cosine_similarity(wv_vectors)
# Для Sentence BERT
sbert_cosine_sim = cosine_similarity(sbert_embeddings)
```

```
import pandas as pd

def print_similar_docs(similarity_matrix, method_name):
print(f"==========================================================")
print(f"Наиболее похожие документы по методу {method_name}:\n")
print(f"==========================================================")
for idx, row in enumerate(similarity_matrix):
similar_indices = row.argsort()[::-1][1:3] # Индексы двух самых похожих (кроме самого себя)
print(f"Документ: {documents[idx]}")
print(f"Похож на: {documents[similar_indices[0]]} (сходство: {row[similar_indices[0]]:.2f})")
print(f"И на: {documents[similar_indices[1]]} (сходство: {row[similar_indices[1]]:.2f})\n")

print_similar_docs(tfidf_cosine_sim, "TF-IDF")
print_similar_docs(wv_cosine_sim, "Word2Vec")
print_similar_docs(sbert_cosine_sim, "Sentence-BERT")
from sklearn.cluster import KMeans
# Выберем число кластеров (например, 3)
n_clusters = 3
# Для TF-IDF
kmeans_tfidf = KMeans(n_clusters=n_clusters, random_state=0).fit(tfidf_matrix)
# Для Word2Vec
kmeans_wv = KMeans(n_clusters=n_clusters, random_state=0).fit(wv_vectors)
# Для Sentence-BERT
kmeans_sbert = KMeans(n_clusters=n_clusters, random_state=0).fit(sbert_embeddings)
import matplotlib.pyplot as plt
from sklearn.manifold import TSNE
```

```
def plot_clusters(embeddings, labels, title, perplexity=5):
# Установка init='random' для поддержки разреженных матриц
tsne = TSNE(n_components=2, random_state=0, perplexity=perplexity, init='random')
reduced_embeddings = tsne.fit_transform(embeddings.toarray() if hasattr(embeddings, "toarray") else embeddings)
plt.figure(figsize=(8, 6))
unique_labels = set(labels)
n_clusters = len(unique_labels)
for i in unique_labels:
cluster_points = reduced_embeddings[labels == i]
plt.scatter(cluster_points[:, 0], cluster_points[:, 1], label=f"Кластер {i}")
plt.legend()
plt.title(title)
plt.xlabel('TSNE Component 1')
plt.ylabel('TSNE Component 2')
plt.show()

# Визуализация для каждого подхода
plot_clusters(tfidf_matrix, kmeans_tfidf.labels_, "Кластеры TF-IDF")
plot_clusters(wv_vectors, kmeans_wv.labels_, "Кластеры Word2Vec", perplexity=5)
plot_clusters(sbert_embeddings, kmeans_sbert.labels_, "Кластеры Sentence-BERT", perplexity=5)

# интерактивная визуализация
import plotly.express as px
import pandas as pd
import scipy
```

```
from sklearn.manifold import TSNE

def plot_clusters_interactive(embeddings, labels, title, perplexity=None):
if scipy.sparse.issparse(embeddings):
embeddings = embeddings.toarray()
n_samples = embeddings.shape[0]
# Автоматический выбор perplexity, если не задан
if perplexity is None:
# Обычно perplexity должно быть < n_samples, минимум 5
perplexity = min(30, n_samples - 1) if n_samples > 1 else 1
else:
# Убедимся, что perplexity меньше n_samples
if perplexity >= n_samples:
raise ValueError(f"perplexity ({perplexity}) must be less than n_samples ({n_samples})")
if perplexity < 1:
raise ValueError("perplexity must be at least 1")
print(f"Используем perplexity={perplexity} при количестве образцов={n_samples}")
tsne = TSNE(n_components=2, random_state=0, perplexity=perplexity)
reduced_embeddings = tsne.fit_transform(embeddings)
df = pd.DataFrame({
'TSNE1': reduced_embeddings[:, 0],
'TSNE2': reduced_embeddings[:, 1],
'Cluster': labels
})
fig = px.scatter(df, x='TSNE1', y='TSNE2', color='Cluster', title=title)
fig.show()

# Визуализация для каждого подхода
```

```
plot_clusters_interactive(tfidf_matrix, kmeans_tfidf.labels_, "Кластеры TF-IDF", perplexity=5) # странная визуализация, подумать, как исправить
#plot_clusters_interactive(wv_vectors, kmeans_wv.labels_, "Кластеры Word2Vec", perplexity=5)
plot_clusters_interactive(sbert_embeddings, kmeans_sbert.labels_, "Кластеры Sentence-BERT", perplexity=5)

# print(f"Количество образцов в sbert_embeddings: {sbert_embeddings.shape[0]}")
# print(f"Количество меток в kmeans_sbert.labels_: {len(kmeans_sbert.labels_)}")
from sklearn.metrics import silhouette_score

# Расчёт коэффициента силуэта для каждого подхода
silhouette_tfidf = silhouette_score(tfidf_matrix, kmeans_tfidf.labels_)
#silhouette_wv = silhouette_score(wv_vectors, kmeans_wv.labels_)
silhouette_sbert = silhouette_score(sbert_embeddings, kmeans_sbert.labels_)
print(f"Силуэт для TF-IDF: {silhouette_tfidf:.2f}")
#print(f"Силуэт для Word2Vec: {silhouette_wv:.2f}")
print(f"Силуэт для Sentence-BERT: {silhouette_sbert:.2f}")

# 7. Классификация текстов


## 7.0. Подготовка данных


# Расширенный корпус текстов с равным количеством примеров для каждого класса
texts = [
```

```
# Семья
"Мама и папа любят проводить время с семьей",
"Бабушка печет вкусные пироги для всей семьи",
"Семейный ужин собирает всех вместе",
"Папа учит сына играть в футбол",
"Мама рассказывает сказку всей семье",
"Родители планируют семейный отпуск на море",
# Животные
"Собака лает на незнакомцев в парке",
"Кошка спит на солнечном подоконнике весь день",
"Львы и тигры являются хищными животными",
"Птицы поют красивые песни по утрам",
"Белка собирает орехи в лесу",
"Дельфины прыгают над волнами в океане",
# Дети
"Дети играют в детском саду с друзьями",
"Малыш учится ходить и говорить",
"Ребенок рисует картину для мамы",
"Школьники делают домашнее задание вместе",
"Дети любят кататься на каруселях",
"Мальчик читает интересную книгу перед сном",
]

# Соответствующие метки классов (0: Семья, 1: Животные, 2: Дети)
labels = [
0, 0, 0, 0, 0, 0, # Семья
1, 1, 1, 1, 1, 1, # Животные
2, 2, 2, 2, 2, 2 # Дети
]
label_names = {0: "Семья", 1: "Животные", 2: "Дети"}
```

```
## 7.0.1 Преобразование текстов в числовое представление
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer()
tfidf_matrix = vectorizer.fit_transform(texts)

## 7.1. Разделение данных на обучение и тестирование
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
tfidf_matrix, labels, test_size=0.3, random_state=42, stratify=labels
)
## 7.2. Обучение классификатора
from sklearn.naive_bayes import MultinomialNB
classifier = MultinomialNB()
classifier.fit(X_train, y_train)

## 7.3. Оценка модели
from sklearn.metrics import classification_report

y_pred = classifier.predict(X_test)
print(classification_report(y_test, y_pred, target_names=label_names.values()))

# 8. Обучение без учителя

## 8.1. Тематическое моделирование с LDA
from sklearn.decomposition import LatentDirichletAllocation

n_topics = 3 # Число тем равно количеству классов
lda_model = LatentDirichletAllocation(n_components=n_topics, random_state=0)
lda_matrix = lda_model.fit_transform(tfidf_matrix)
# Вывод топ-слов для каждой темы
```

```
feature_names = vectorizer.get_feature_names_out()
for topic_idx, topic in enumerate(lda_model.components_):
print(f"\nТема {topic_idx + 1} {list(label_names.values())[topic_idx]}:")
print(", ".join([feature_names[i] for i in topic.argsort()[:-11:-1]]))
```