

**Министерство науки и высшего образования Российской Федерации**  
федеральное государственное автономное образовательное учреждение  
высшего образования  
«Национальный исследовательский Томский политехнический университет»

Школа / филиал	ИЯТШ ТПУ
Обеспечивающее подразделение	ОЭФ
Направление подготовки / специальность	01.03.02 Прикладная математика и информатика
Образовательная программа (направленность (профиль))	<b>Прикладная математика в инженерии</b>

### ОТЧЕТ О ПРАКТИКЕ

Вид практики	учебная
Тип практики	практика по получению первичных профессиональных умений и навыков
Место практики	Томский политехнический университет

Выполнил обучающийся	Чепкасов Иван Юрьевич
Группа	0В32

\_\_\_\_\_  
(подпись обучающегося)

Руководитель практики ТПУ:

\_\_\_\_\_  
(степень, звание, должность)

\_\_\_\_\_  
(Ф. И. О.)

Дата проверки \_\_\_\_\_ 20\_\_ г.

Допустить / не допустить к защите  
Подпись \_\_\_\_\_

Итоговая оценка по практике \_\_\_\_\_  
(традиционная оценка, балл)

## Содержание

Введение.....	3
I. Теоретическое содержание работы.....	5
1. Методы работы с языковыми структурами и задачи обработки текстов.....	5
2. Алгоритм TF-IDF.....	8
3. Алгоритм LDA (Latent Dirichlet Allocation).....	9
II. Практическая часть.....	17
1. Предобработка текста.....	17
2. Использование TF-IDF.....	18
3. Реализация LDA.....	19
Заключение.....	26
Список источников.....	27
Приложения.....	28

## **Введение**

Трудно переоценить прогресс в машинном обучении в последние годы и практическую значимость этой области для современных информационных технологий и программных продуктов. Невозможно представить социальную сеть или маркетплейс, который не использовал бы рекомендательные системы, невозможно представить банковскую организацию, которая не использовала бы системы кредитного скоринга.

В последние десятилетия объем текстовой информации в цифровом формате стремительно растет: социальные сети, новостные порталы, научные публикации и корпоративные документы ежедневно порождают миллиарды текстов. Эффективная обработка и анализ таких данных позволяют извлекать важную информацию: выявлять ключевые темы, автоматизировать классификацию документов и улучшать системы поиска. В этом контексте методы обработки естественного языка (Natural Language Processing, NLP) и тематического моделирования обретают огромную значимость.

Важные составляющие современных программных продуктов – от рекомендательных систем до инструментов управления знаниями – требуют автоматизированных и надежных средств выявления скрытых тем и выделения ключевых терминов в больших объемах текстовой информации. Классические статистические подходы, такие как TF-IDF, и более сложные модели, например LDA, демонстрируют высокую эффективность при анализе текстовых данных различной тематики и объёма.

Среди целей практики следует выделить:

- 1) Изучение машинного обучения и его практического применения к решению задач регрессии, классификации, кластеризации и снижения размерности данных посредством прохождения курса “Практический Machine Learning” на платформе Stepik;

- 2) Освоение основ NLP и выполнение проекта в рамках индивидуального задания, посвященного сравнительному исследованию двух подходов к анализу текста на английском языке – TF-IDF и Latent Dirichlet Allocation (LDA) – в рамках единого пайплайна: от предварительной обработки текста до извлечения ключевых терминов и тем.

Задачи реализуемого проекта прямо вытекают из постановки индивидуального задания:

- 1) Описать методы работы с языковыми структурами и возникающие задачи при обработке текстов.
- 2) Реализовать пайплайн обработки выбранного текста на английском языке, включая всю необходимую предварительную обработку текста (приведение слов к нижнему регистру, удаление стоп-слов, цифр/неалфавитных символов, знаков пунктуации).
- 3) Разделить текст на главы и в каждой главе отобрать Топ-20 слов с помощью алгоритма TF-IDF.
- 4) Реализовать LDA алгоритм и сравнить результаты с полученными ранее с помощью TF-IDF. Сделать выводы о применимости реализованных подходов.

В результате исследования будут выявлены сильные и слабые стороны каждого метода, а также даны рекомендации по их применению в решении прикладных задач анализа больших текстовых массивов.

## I. Теоретическое содержание работы

Natural Language Processing – обработка естественного языка – подраздел информатики и ИИ, посвященный тому, как компьютеры анализируют естественные языки.

### 1. Методы работы с языковыми структурами и задачи обработки текстов

Для эффективной обработки и анализа текстовых данных в задачах NLP применяются разнообразные методы, каждый из которых фокусируется на определенном аспекте языка.

#### Морфологический анализ

– это анализ словоформ и частей речи.

Цель: привести слова к единообразному представлению и выделить их базовые лексические формы. Далее будут описаны основные составляющие морфологического анализа.

##### 1) Токенизация:

- **токенизация по предложениям** – это процесс разделения письменного языка на предложения-компоненты;
- **токенизация** (иногда – сегментация) **по словам** – это процесс разделения предложений на слова-компоненты;
- **подсловная токенизация** - разбиение на более мелкие единицы для работы с неизученными словами и морфологически богатым языком.

##### 2) Нормализация:

- приведение к нижнему регистру;
- удаление или замена диакритических знаков;
- удаление стоп-слов (артикли, междометия, союзы и т. д.), цифр/неалфавитных символов, знаков пунктуации.

##### 3) Стемминг и лемматизация

Тексты часто содержат одни и те же слова в разных формах, а также однокоренные слова. Целью стемминга и лемматизации является приведение слов к начальной форме (например, dog, dogs, dog's, dogs' => dog).

**Стемминг** – отрезание “лишнего” от корня слов, что часто приводит к потере словообразовательных суффиксов.

**Лемматизация** – это процесс, который использует словарь, чтобы в итоге привести слово к его канонической форме – лемме.

Стемминг - это более “грубый” метод. Его использование часто приводит к ошибкам (например, слово good – это лемма для слова better; стеммер не увидит эту связь). Тем не менее, стеммеры проще в реализации и работают быстрее, а ошибки, обусловленные особенностью метода, часто допустимы в практических задачах.

### **Синтаксический анализ (parsing)**

– это процесс выявления грамматической структуры предложения.

Цель: установить грамматические отношения между словами для понимания структуры предложений. Среди основных составляющих синтаксического анализа могут быть выделены аспекты ниже.

#### 1) Конституентный (иерархический) парсинг

– построение древовидной структуры (parse tree), где каждый узел — это синтаксический конституент (фраза):

- S (Sentence);
- NP (Noun Phrase) — именная группа;
- VP (Verb Phrase) — глагольная группа;
- ...

#### 2) Зависимый парсинг (dependency parsing)

– построение графа, где узлы — это слова, а ребра показывают синтаксико-семантические связи (“ядро → зависимое”).

#### 3) Распознавание грамматических отношений

– выявление функций словоформ в предложении:

- Подлежащее (nsubj / Nom);
- Сказуемое (root / Pred);
- ...

## Семантический анализ

– это смысловой анализ текста.

Цель: извлечь смысловые отношения и представления на уровне слов, предложений, документов. Опишем ниже основные компоненты семантического анализа.

### 1) Извлечение именованных сущностей (NER)

– поиск и классификация имен собственных: люди, организации, локации, даты и т. д.

### 2) Распознавание семантических ролей (SRL)

– установление отношений “агент–действие–объект”.

### 3) Корелационный анализ

– связывание местоимений и ссылок с соответствующими сущностями (“Alice ... she ... her”).

### 4) Построение семантических представлений:

- Векторные представления слов (Word2Vec, GloVe, fastText);
- Контекстные эмбединги (BERT, RoBERTa, GPT и др.) – модель кодирует смысл слова/фразы с учетом контекста.

## Задачи NLP

Задачи, решаемые NLP можно разбить по уровням.

Задача	Уровень
Распознавание речи, синтез речи	Сигнал
Морфологический анализ	Слово
POS-тэгирование, распознавание именованных сущностей, выделение слов	Словосочетание
Синтаксический разбор, токенизация предложений	Предложение
Извлечение отношений, определение языка, анализ эмоциональной окраски	Абзац
Аннотация документа, перевод, анализ	Документ

тематики, генерация текста	
Дедубликация, информационный поиск	Корпус

Таблица 1. Задачи NLP

## 2. Алгоритм TF-IDF

TF-IDF (Term Frequency – Inverse Document Frequency) переводит текст в вектор признаков, отражающих “важность” каждого слова в документе относительно всего корпуса.

### 1) Term Frequency (TF)

Частота термина  $t$  в документе  $d$ :

$$\text{TF}(t, d) = f_{t,d}$$

где  $f_{t,d}$  — число вхождений  $t$  в  $d$ .

Для нормализации часто используют:

$$\text{TF}(t, d) = \frac{f_{t,d}}{\max_{t'} f_{t',d}}$$

или

$$\text{TF}(t, d) = 1 + \log(f_{t,d}), \quad f_{t,d} > 0.$$

### 2) Inverse Document Frequency (IDF)

Обратная частота термина по всему корпусу:

$$\text{IDF}(t) = \log \left( \frac{N}{\text{DF}(t)} \right)$$

где

- $N$  — общее число документов,
- $\text{DF}(t)$  — число документов, содержащих термин  $t$ .



### 3) TF-IDF

Взвешенный признак:

$$TF-IDF(t, d) = TF(t, d) \times IDF(t)$$

- Высокий вес получают термины, часто встречающиеся в конкретном документе и редко во всём корпусе.
- Низкий вес – у часто встречающихся по всему корпусу или редких в данном документе.

Особенности TF-IDF:

- определяет важность слов;
- устраняет шум (стоп-слова имеют низкий IDF);
- отсутствие семантической информации;
- чувствителен к длине документа (для устранения этой проблемы используется нормализация).

Применение TF-IDF:

- векторизация текста для классификации, кластеризации, поиска;
- извлечение ключевых слов (слова с наибольшим TF-IDF);
- измерение сходства запроса и документов по косинусному расстоянию их TF-IDF векторов.

### 3. Алгоритм LDA (Latent Dirichlet Allocation)

Цель: автоматически выявить скрытые темы в корпусе документов и представить каждый документ как смесь этих тем.

Идея алгоритма:

- Каждый документ - смесь тем, он может быть представлен как распределение по темам.
- Каждая тема может быть определена распределением слов словаря.

Для полного понимания принципа работы данного алгоритма необходимо привести дополнительные сведения из теории вероятностей и байесовской статистики.

## Необходимые сведения из теории вероятностей

**DEF.** К-мерный симплекс – множество:

$$\left\{ x \in \mathbb{R}^K \mid x_i \geq 0, \sum_{i=1}^K x_i = 1 \right\}.$$

**DEF.** Распределение Дирихле – это многомерное непрерывное распределение с параметром  $\alpha = (\alpha_1, \dots, \alpha_K)$ , плотность которого на К-мерном симплексе задается как:

$$f(x_1, \dots, x_K; \alpha) = \frac{1}{B(\alpha)} \prod_{i=1}^K x_i^{\alpha_i - 1},$$

$$B(\alpha) = \frac{\prod_{i=1}^K \Gamma(\alpha_i)}{\Gamma\left(\sum_{i=1}^K \alpha_i\right)}.$$

Замечание: распределение Дирихле обобщает бета-распределение на многомерный случай.

Пусть

$$\xi \sim \text{Dir}(\alpha),$$

тогда

$$\mathbb{E}\xi = \frac{\alpha}{\alpha_0}, \quad \text{cov}(\xi_i, \xi_j) = \frac{\alpha_0 \delta_{ij} - \alpha_i \alpha_j}{\alpha_0^2 (\alpha_0 + 1)}, \quad \alpha_0 = \sum_{k=1}^K \alpha_k.$$

**DEF.** Мультиномиальное распределение – это обобщение биномиального распределения на  $k > 2$  исходов.

Пусть в  $n$  независимых испытаниях вероятность  $j$ -го исхода равна  $p_j$

$$\sum_{j=1}^k p_j = 1.$$

Тогда вектор частот  $(Y_1, \dots, Y_k)$  имеет мультиномиальное распределение.

$$\mathbb{P}(Y_1 = m_1, \dots, Y_k = m_k) = \frac{n!}{m_1! \dots m_k!} p_1^{m_1} \dots p_k^{m_k}, \quad \sum_{i=1}^k m_i = n.$$

Замечание: при  $n = 1$  мультиномиальное распределение превращается в категориальное.

### Некоторые понятия Байесовской статистики

**DEF.** Априорное распределение – это распределение неизвестных параметров до учета наблюдений. Обозначается:  $p(\Theta)$ .

**DEF.** Апостериорное распределение - это условное распределение параметров после учета наблюдений, вычисляемое по формуле Байеса:

$$p(\theta | x) = \frac{p(x | \theta) \cdot p(\theta)}{p(x)}$$

Здесь:

- $p(\Theta | x)$  – апостериорное распределение;
- $p(x | \Theta)$  – правдоподобие;
- $p(\Theta)$  – априорное распределение;
- $p(x)$  – нормализующая константа (маргинализированная вероятность).

$$p(x) = \int p(x | \theta) p(\theta) d\theta$$

**DEF.** Байесовская модель – вероятностная модель, в которой параметры  $\Theta$  считаются случайными величинами, обладающими априорным распределением  $p(\Theta)$ . После учета наблюдений получается апостериорное распределение  $p(\Theta|x)$ .

Полная Байесовская постановка:

- 1) Все параметры и скрытые переменные модели представляются как случайные величины (например:  $\theta, z$ );
- 2) Для каждой случайной величины задается априорное распределение;
- 3) Вывод о скрытых переменных и параметрах осуществляется путем полного применения формулы Байеса, т. е. нахождения апостериорного распределения:

$$p(\theta, z | x) \propto p(x | \theta, z) \cdot p(\theta) \cdot p(z).$$

Порождающая (генеративная) Байесовская модель:

- 1) Из априорного распределения выбирается  $\Theta \in p(\Theta)$ ;
- 2) По  $p(z | \Theta)$  выбираются скрытые переменные;
- 3) По  $p(x | z, \Theta)$  генерируются наблюдаемые данные.

## Генеративная модель LDA

Обозначения:

- $M$  – число документов в корпусе;
- $V$  – число различных слов в словаре;
- $K$  – число тем (гиперпараметр);
- $\alpha = (\alpha_1, \dots, \alpha_K)$  – гиперпараметр;
- $\beta = (\beta_1, \dots, \beta_V)$  – гиперпараметр;

$\alpha, \beta$  задают параметры априорных распределений Дирихле.

1) Априорные распределения тем и слов

- a) Для каждой темы  $t = 1, \dots, K$  выбираем вектор

$$\varphi_t = (\varphi_{t1}, \dots, \varphi_{tV}) \in Dir(\beta)$$

$\varphi_{tj}$  – вероятность слова  $j$  в теме  $t$

Таким образом, каждая тема задает распределение по словам.

- b) Для каждого документа  $d = 1, \dots, M$  выбираем вектор

$$\Theta_d = (\Theta_{d1}, \dots, \Theta_{dK}) \in Dir(\alpha)$$

$\Theta_{dt}$  – вероятность темы  $t$  в документе  $d$

2) Генерация слов документа

$N_d$  – число слов в документе  $d$ . Для каждого слова  $n = 1, \dots, N_d$

- a) Выбираем скрытую тему  $z_{d,n}$ :

$$z_{d,n} \in Multinomial(\Theta_d)$$

$$P(z_{d,n} = t) = \Theta_{dt}$$

- b) Выбираем слово  $\omega_{d,n}$  из распределения слов темы  $t = z_{d,n}$ :

$$\omega_{d,n} \in Multinomial(\varphi_t)$$

$$P(\omega_{d,n} = j \mid z_{d,n} = t) = \varphi_{t,j}$$

Таким образом,

$$p(\{\theta_d\}, \{\varphi_t\}, \{z_{d,n}\}, \{w_{d,n}\} \mid \alpha, \beta) = \left[ \prod_{t=1}^K p(\varphi_t \mid \beta) \right] \left[ \prod_{d=1}^M p(\theta_d \mid \alpha) \prod_{n=1}^{N_d} p(z_{d,n} \mid \theta_d) p(w_{d,n} \mid \varphi_{z_{d,n}}) \right],$$

$$p(\theta_d \mid \alpha) = \text{Dir}(\alpha),$$

$$p(\varphi_t \mid \beta) = \text{Dir}(\beta).$$

## Обучение модели

Даны слова  $\omega_{d,n}$ .

Необходимо оценить  $\{\theta_d\}$ ,  $\{\varphi_t\}$ ,  $\{z_{d,n}\}$ .

Для этого ищем апостериорное распределение:

$$p(\theta, \varphi, z \mid w, \alpha, \beta) = \frac{p(\theta, \varphi, z, w \mid \alpha, \beta)}{p(w \mid \alpha, \beta)}$$

Однако данная задача вычислительно нетривиальна, поэтому необходимо использовать приближенные методы.

Среди этих методов наиболее широко используются:

- Коллапсированное приближение и Гиббсовская выборка;
- Вариационный Байесовский метод.

Здесь будет подробно описан только первый алгоритм.

## Коллапсированное приближение и Гиббсовская выборка

Данный метод основывается на следующем утверждении:

$$p(z_{d,n} = t \mid z_{\setminus(d,n)}, w, \alpha, \beta) \propto (n_{d,t}^{-i} + \alpha_t) \frac{n_{t,w_{d,n}}^{-i} + \beta_{w_{d,n}}}{n_{t,\bullet}^{-i} + \sum_j \beta_j}.$$

Здесь:

- $z_{\setminus(d,n)}$  — все метки тем, кроме текущего слова (d,n);
- $n_{d,t}^{-i}$  — количество слов в документе d отнесенных к теме t (без текущего);
- $n_{t,w}^{-i}$  — количество слов w отнесенных к теме t (без текущего);

- $n_{t,\bullet}^{-i} = \sum_w n_{t,w}^{-i}$  — общее число слов в теме  $t$  (без текущего);
- $\alpha_t, \beta_w$  — гиперпараметры априорных распределений Дирихле.

Обоснование данного утверждения можно посмотреть в статье [\[10\]](#).

Теперь, имея данное утверждение, можно описать численный метод оценки параметров  $\{\Theta_d\}$  и  $\{\varphi_t\}$ .

## Алгоритм

### 1) Инициализация

Для каждого документа  $d$  и каждого слова  $n = 1, \dots, N_d$  присваиваем тему:

$$z_{d,n} \sim \text{Categorical} \left( \left\{ \frac{1}{K} \right\}_{t'=1}^K \right)$$

Обновляем счетчики:

$$n_{d,t} = \sum_{n: z_{d,n}=t} 1,$$

$$n_{t,w} = \sum_{d,n: w_{d,n}=w, z_{d,n}=t} 1$$

### 2) Основной цикл (итерации Гиббса)

Пока не выполнено условие сходимости, для каждого документа  $d = 1, \dots, M$  и каждого слова  $n = 1, \dots, N_d$  повторяются шаги:

#### 1. Удаление текущей метки

Пусть текущее значение темы  $t = z_{d,n}$

$$n_{d,t} -= 1, \quad n_{t,\omega_{d,n}} -= 1, \quad n_t -= 1.$$

#### 2. Вычисление условных вероятностей для новой темы

Для каждой темы  $t' = 1, \dots, K$  вычислить “ненормированную” вероятность того, что  $z_{d,n} = t'$ :

$$p(z_{d,n} = t' \mid z_{\setminus(d,n)}, w, \alpha, \beta) \propto (n_{d,t'} + \alpha_{t'}) \cdot \frac{n_{t',w_{d,n}} + \beta_{w_{d,n}}}{n_{t',\bullet} + \sum_{w'=1}^V \beta_{w'}}.$$

### 3. Нормировка

$$\sum_{t'=1}^K p(z_{d,n} = t') = 1$$

### 4. Ресемплирование темы

Сгенерировать новую тему для слова  $n$  документа  $d$ :

$$z_{d,n} \sim \text{Categorical}(\{p_{t'}\}_{t'=1}^K)$$

$$n_{d,z_{d,n}} + = 1, \quad n_{z_{d,n},w_{d,n}} + = 1, \quad n_{z_{d,n},\bullet} + = 1.$$

### 3) Оценка параметров после сходимости

$$\hat{\theta}_{d,t} = \frac{n_{d,t} + \alpha_t}{N_d + \sum_{t'} \alpha_{t'}},$$

$$\hat{\varphi}_{t,w} = \frac{n_{t,w} + \beta_w}{n_{t,\bullet} + \sum_{w'} \beta_{w'}}.$$

Критерии останова:

- 1) фиксированное число итераций  $T$ ;
- 2) мониторинг  $\ln(p(\omega|\alpha, \beta))$ ;
- 3) стабилизация оценок параметров.

Вычислительная сложность:  $O(T \times N \times K)$ .

## Метрики качества LDA

### 1) Перплексия

$$\text{Perplexity}(D_{\text{test}}) = \exp \left( -\frac{\sum_{d=1}^M \sum_{n=1}^{N_d} \ln p(w_{d,n})}{\sum_{d=1}^M N_d} \right)$$

- где  $p(w_{d,n})$  — предсказанная моделью вероятность  $n$ -го слова  $d$ -го документа;
- $N_d$  — длина документа.

Низкая перплексия означает, что модель хорошо предсказывает слова.

## 2) Когерентность тем (UMass)

Пусть тема  $t$  задана списком  $T = \{w_1, \dots, w_K\}$  топ- $K$  слов. Тогда:

$$C_{UMass}(T) = \sum_{i=2}^K \sum_{j=1}^{i-1} \log \frac{D(w_i, w_j) + \epsilon}{D(w_j)}$$

где

- $D(w_i, w_j)$  — число документов, содержащих оба слова  $w_i, w_j$ ;
- $D(w_j)$  — число документов с  $w_j$ ;
- $\epsilon$  — сглаживающий малый параметр.

Высокая когерентность означает, что каждая из тем согласована по смыслу.

Замечание: гиперпараметр  $K$  оптимизируется по когерентности, после чего проверяется перплексия.



## II. Практическая часть

Практическая часть реализована в среде [Google Colab](#). Существенно значимые фрагменты кода и результаты его работы будут также представлены здесь. Алгоритмы обучаются на тексте, приведенном в приложении (2).

### 1. Предобработка текста

Реализовать пайплайн обработки выбранного текста на английском языке, включая всю необходимую предварительную обработку текста (приведение слов к нижнему регистру, удаление стоп-слов, цифр/неалфавитных символов, знаков пунктуации).

Для этой задачи используется библиотека NLTK и регулярные выражения.

```
# Пайплайн обработки текста
def preprocess_text(text):
    # Приведение к нижнему регистру
    text = text.lower()

    # Удаление цифр, неалфавитных признаков и знаков пунктуации
    text = re.sub(r'^a-z\s]', '', text)

    # Токенизация
    tokens = word_tokenize(text)

    # Удаление стоп-слов
    stop_words = set(stopwords.words('english'))
    tokens = [word for word in tokens if word not in stop_words and len(word) >
2]

    return tokens

# Разделение текста на главы
def extract_chapters(text):
    # Формат заголовков: "CHAPTER X: TITLE"
    # Разбиваем текст на главы с сохранением разделителей
    chapters = re.split(r'(^CHAPTER\s+[^:]+:.*$)', text, flags=re.MULTILINE |
re.IGNORECASE)

    # Собираем абзацы из непустых частей между заголовками
    paragraphs = []
    for part in chapters:
        # Пропускаем заголовки глав и пустые строки
```

```

        if re.match(r'^CHAPTER\s+[:]+:.*$', part, flags=re.IGNORECASE) or not
part.strip():
            continue
        # Удаляем начальные/конечные пробелы и добавляем абзац
        paragraphs.append(part.strip())

return paragraphs

```

В результате применения данных функций, получаем предобработанный текст с разбиением на главы:



Токенизированные главы

```

Глава 1: machine learning subfield artificial intelligence focuses building systems learn data
Глава 2: natural language processing enables computers understand human language key technic
Глава 3: dimensionality reduction methods simplify complex datasets principal component anal
Глава 4: clustering groups similar data points together kmeans partitions data spherical clu
Глава 5: topic modeling discovers abstract themes document collections latent dirichlet allo

```

## 2. Использование TF-IDF

Разделить текст на главы и в каждой главе отобрать Топ-20 слов с помощью алгоритма TF-IDF.

```

from sklearn.feature_extraction.text import TfidfVectorizer

# Функция для нахождения топ-n слов по TF-IDF в каждой главе
def tf_idf_get_top_words(chapters, top_n=20):
    vectorizer = TfidfVectorizer()
    tf_idf_matrix = vectorizer.fit_transform(chapters)
    feature_names = vectorizer.get_feature_names_out()
    top_words_per_chapter = []
    for i in range(len(chapters)):
        tfidf_scores = zip(feature_names, tf_idf_matrix[i].toarray()[0])
        sorted_words = sorted(tfidf_scores, key=lambda x: x[1],
reverse=True)[:top_n]
        top_words_per_chapter.append([word for word, _ in sorted_words])
    return top_words_per_chapter

# Получение топ-20 слов для каждой главы
tfidf_results = tf_idf_get_top_words(chapters)

# Вывод результатов
print("Топ слов по TF-IDF")
for i, words in enumerate(tfidf_results):
    print(f"Глава {i+1}: {' '.join(words)}")

```

## Топ-20 слов по TF-IDF

Глава 1: learning, data, algorithms, artificial, building, critical, decision, engineering, feature, focuses, form, foundation, intelligence, learn, linear, main, modern, pipeline, preprocessing, regression.

Глава 2: language, architectures, bagofwords, computers, enables, key, lemmatization, models, natural, nlp, numerically, processing, range, recent, represent, revolutionized, sentiment, stemming, tokenization, transformer.

Глава 3: reduction, autoencoders, axes, complex, component, creates, datasets, help, highdimensional, improve, local, lowdimensional, maximum, nonlinear, orthogonal, pca, performance, preserving, principal, simplify.

Глава 4: clustering, clusters, include, data, anomaly, areas, based, builds, centroids, customer, daviesbouldin, dbscan, dendrograms, dense, detection, groups, hierarchical, index, kmeans, metrics.

Глава 5: topic, abstract, allocation, automated, coherence, collections, dirichlet, discovers, discovery, document, documents, enable, factorization, generative, interpretation, latent, lda, matrix, mixtures, modeling.

## 3. Реализация LDA

Реализовать LDA алгоритм и сравнить результаты с полученными ранее с помощью TF-IDF.

### 3.1 Реализация с нуля.

```
from sklearn.feature_extraction.text import CountVectorizer

def gibbs_lda(docs, n_topics, alpha, beta, n_iters):

    # Векторизация документов
    cv = CountVectorizer()
    X = cv.fit_transform(docs)
    vocab = cv.get_feature_names_out()
    D, V = X.shape

    # Преобразование документов в списки индексов слов
    docs_idx = [[idx for idx, count in zip(row.indices, row.data) for _ in
range(count) ]
```

```

        for row in X]

# Инициализация счетчиков
nw = np.zeros((n_topics, V), dtype=int)      # счетчик тема-слово
nd = np.zeros((D, n_topics), dtype=int)      # счетчик документ-тема
nwsum = np.zeros(n_topics, dtype=int)        # число слов в теме
z = []   # метки тем на словах

# Случайно первоначально присваиваем метки тем
for d, doc in enumerate(docs_idx):
    z_d = []
    for w in doc:
        topic = np.random.randint(n_topics)
        z_d.append(topic)
        nw[topic, w] += 1
        nd[d, topic] += 1
        nwsum[topic] += 1
    z.append(z_d)

alpha_vec = np.ones(n_topics) * alpha
beta_vec = np.ones(V) * beta

# Итерации Гиббса
for _ in range(n_iters):
    for d, doc in enumerate(docs_idx):
        for i, w in enumerate(doc):
            old_topic = z[d][i]
            # Удаление текущей метки
            nw[old_topic, w] -= 1
            nd[d, old_topic] -= 1
            nwsum[old_topic] -= 1

            # Вычисление условных вероятностей
            p = (nd[d] + alpha_vec) * (nw[:, w] + beta_vec[w]) / (nwsum +
beta_vec.sum())
            p = p / p.sum()

            # Присваивание новой темы
            new_topic = np.random.choice(n_topics, p=p)
            z[d][i] = new_topic

            # Обновление счетчиков
            nw[new_topic, w] += 1
            nd[d, new_topic] += 1

```

```

        nwsun[new_topic] += 1

# Оценки параметров LDA
phi = (nw + beta) / (nwsun[:, None] + beta * V)
theta = (nd + alpha) / (np.sum(nd, axis=1)[:, None] + alpha * n_topics)
return phi, theta, vocab

```

Число тем известно заранее в рассматриваемой задаче, однако другие гиперпараметры ( $\alpha$ ,  $\beta$ ) необходимо подобрать.

```

# Метрика - Перплексия
def lda_perplexity(docs, n_topics, alpha, beta, n_iters):

    phi, theta, _ = gibbs_lda(docs, n_topics, alpha, beta, n_iters)

    # Векторизация документов
    cv = CountVectorizer()
    X = cv.fit_transform(docs)

    # Общее число слов во всех тестовых документах
    nd_sum = int(X.sum())

    # Сумма лог-правдоподобий всех слов
    log_likelihood = 0

    D, V = X.shape
    for d in range(D):
        indices = X[d].nonzero()[1]
        counts = X[d, indices].toarray().ravel()
        probs = theta[d][:, np.newaxis] * phi[:, indices]
        word_probs = probs.sum(axis=0)
        log_likelihood += np.dot(counts, np.log(word_probs))

    perp = np.exp(- log_likelihood / nd_sum)
    return perp

# Множество значений параметров
param_grid = {'alpha': np.arange(0.005, 0.1, 0.005), 'beta': np.arange(0.025,
0.04, 0.005)}

# Полный перебор
params_values = (0.1, 0.01)

```

```

perplexity = lda_perplexity(docs=chapters, n_topics=5, alpha=params_values[0],
beta=params_values[1], n_iters=1000)

for alpha in param_grid['alpha']:
    for beta in param_grid['beta']:
        cur_perplexity = lda_perplexity(docs=chapters, n_topics=5, alpha=alpha,
beta=beta, n_iters=1000)
        if cur_perplexity < perplexity:
            perplexity = cur_perplexity
            params_values = (round(alpha, 2), round(beta, 2))

print(params_values)

(np.float64(0.01), np.float64(0.04))

[239] phi, theta, vocab = gibbs_lda(docs=chapters, n_topics=5, alpha=0.01, beta=0.04, n_iters=2000)

```

В результате, получаем следующие топ-20 слов для каждой главы при помощи реализованного алгоритма LDA:

	Глава 1	Глава 2	Глава 3	Глава 4	Глава 5
0	learning	applications	analysis	include	topic
1	data	years	reduction	clusters	uses
2	like	word	techniques	clustering	topics
3	language	wordvec	methods	data	text
4	machine	text	model	dense	nonnegative
5	systems	tokenization	human	daviesbouldin	nmf
6	three	transformer	dimensionality	centroids	modeling
7	subfield	supervised	embeddings	customer	collections
8	tsne	translation	themes	using	allocation
9	unsupervised	building	visualize	together	abstract
10	understand	bagofwords	use	identifies	factorization
11	trees	types	variance	hierarchical	coherence
12	neural	regression	scale	regions	enable
13	intelligence	key	principal	score	document
14	focuses	numerically	performance	segmentation	dirichlet
15	learn	main	preserving	separated	discovers
16	modern	lemmatization	mixtures	metrics	evaluation
17	networks	natural	networks	nested	documents
18	nlp	linear	maximum	partitions	latent
19	models	pipeline	nonlinear	points	lda

Для полноты картины также реализуем LDA при помощи библиотеки scikit-learn. Данная библиотека использует вариационный Байесовский метод для оценки параметров.

### 3.2 Использование scikit-learn для реализации LDA

Подбор гиперпараметров:

```
from sklearn.decomposition import LatentDirichletAllocation
from sklearn.model_selection import GridSearchCV


# Векторизация документов
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(chapters)
vocab = vectorizer.get_feature_names_out()

# Множество подбираемых параметров
param_grid = {'doc_topic_prior': np.arange(0.01, 1, 0.04), 'topic_word_prior':
np.arange(0.005, 1, 0.005)}

# Метрика - отрицательная Перплексия
def neg_perplexity(estimator, X_data):
    return -estimator.perplexity(X_data)

grid_search = GridSearchCV(
    estimator=LatentDirichletAllocation(n_components=5,
learning_method='batch'),
    param_grid=param_grid,
    scoring=neg_perplexity,
    cv=5)

grid_search.fit(X)
print("Лучшие параметры:", grid_search.best_params_)
```

 Лучшие параметры: {'doc\_topic\_prior': np.float64(0.93), 'topic\_word\_prior': np.float64(0.9600000000000001)}

Обучение модели:

```
lda = LatentDirichletAllocation(n_components=5,
                                max_iter=10,
                                learning_method='batch',
                                doc_topic_prior=0.93,
                                topic_word_prior=0.96)

lda.fit(X)
```

Результат работы LDA из scikit-learn:

	Глава 1	Глава 2	Глава 3	Глава 4	Глава 5
0	learning	data	reduction	include	topic
1	data	analysis	analysis	data	analysis
2	machine	applications	techniques	clusters	human
3	like	techniques	embeddings	clustering	text
4	applications	embeddings	identifies	applications	evaluation
5	networks	human	model	language	dimensionality
6	neural	text	methods	identifies	methods
7	feature	machine	dimensionality	evaluation	model
8	trees	like	neural	dbscan	uses
9	foundation	identifies	networks	spherical	automated
10	steps	reduction	data	customer	dirichlet
11	supervised	evaluation	preserving	detection	matrix
12	learn	methods	datasets	dendrograms	represents
13	building	dimensionality	local	hierarchical	generative
14	algorithms	model	maximum	dense	coherence
15	linear	networks	help	index	latent
16	systems	neural	performance	together	documents
17	subfield	language	variance	groups	nmf
18	regression	include	autoencoders	silhouette	document
19	pipeline	natural	tsne	nested	scale

### 3.3 Сравнение результатов работы TF-IDF и двух версий LDA

Далее будут сопоставлены наиболее значимые ключевые слова для каждой главы, определенных тремя алгоритмами.

Глава 1

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
TF-IDF	learning	data	algorithms	artificial	building	critical	decision	engineering	feature	focuses	form	foundation	intelligence	learn	linear	main	modern	pipeline	preprocessing	regression
LDA (Gibbs)	learning	data	like	language	machine	systems	three	subfield	tsne	unsupervised	understand	trees	neural	intelligence	focuses	learn	modern	networks	nlp	models
LDA (Bayes)	learning	data	machine	like	applications	networks	neural	feature	trees	foundation	steps	supervised	learn	building	algorithms	linear	systems	subfield	regression	pipeline

Глава 2

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
TF-IDF	language	architectures	bagofwords	computers	enables	key	lemmatization	models	natural	nlp	numerically	processing	range	recent	represent
LDA (Gibbs)	applications	years	word	wordvec	text	tokenization	transformer	supervised	translation	building	bagofwords	types	regression	key	numerically
LDA (Bayes)	data	analysis	applications	techniques	embeddings	human	text	machine	like	identifies	reduction	evaluation	methods	dimensionality	model



## Глава 3

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
TF-IDF	reduction	autoencoders	axes	complex	component	creates	datasets	help	highdimensional	improve	local	lowdimensional	maximum	nonlinear	orthogonal
LDA (Gibbs)	analysis	reduction	techniques	methods	model	human	dimensionality	embeddings	themes	visualize	use	variance	scale	principal	performance
LDA (Bayes)	reduction	analysis	techniques	embeddings	identifies	model	methods	dimensionality	neural	networks	data	preserving	datasets	local	maximum

## Глава 4

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
TF-IDF	clustering	clusters	include	data	anomaly	areas	based	builds	centroids	customer	daviesbouldin	dbscan	dendrograms	dense	detection
LDA (Gibbs)	include	clusters	clustering	data	dense	daviesbouldin	centroids	customer	using	together	identifies	hierarchical	regions	score	segmentation
LDA (Bayes)	include	data	clusters	clustering	applications	language	identifies	evaluation	dbscan	spherical	customer	detection	dendrograms	hierarchical	dense

## Глава 5

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
TF-IDF	topic	abstract	allocation	automated	coherence	collections	dirichlet	discovers	discovery	document	documents	enable	factorization	generative	interpretation
LDA (Gibbs)	topic	uses	topics	text	nonnegative	nmf	modeling	collections	allocation	abstract	factorization	coherence	enable	document	dirichlet
LDA (Bayes)	topic	analysis	human	text	evaluation	dimensionality	methods	model	uses	automated	dirichlet	matrix	represents	generative	coherence

Как мы видим из сопоставления ключевых слов, определенных каждым из алгоритмов, каждый из них показывает себя довольно хорошо в данной задаче. Из общей картины выбивается вторая глава, которая выделяется LDA-алгоритмами не совсем четко. Тем не менее, следует учитывать, что LDA также может быть использован для выделения любого числа скрытых тем при условии, что нам неизвестны темы, которые встречаются в корпусе документов. Также, вероятно, LDA алгоритмы лучше себя покажут на текстах большего объема.

Две реализации LDA, использующие разные методы оценки параметров, показывают схожие результаты в определении тем, но реализация библиотеки `scikit-learn`, использующая вариационный Байесовский метод, работает быстрее. Наиболее быстро ключевые слова выявляются при помощи TF-IDF.

Таким образом, TF-IDF подойдёт для быстрого выделения ключевых слов внутри отдельного документа, а LDA — для выявления скрытых тем в целом корпусе.

## Заключение

В ходе летней практики были изучены методы классического машинного обучения, применяемые к решению задач регрессии, классификации, кластеризации и снижения размерности. Это было достигнуто путем прохождения курса “Практический Machine Learning” на платформе Stepik.

Затем, в ходе второй части практики, были изучены основы NLP и выполнен индивидуальный проект, посвященный задаче выделения ключевых слов в тексте и тематическому моделированию. Данный проект включает изучение соответствующей темы и алгоритмов, решающих поставленную задачу — определение ключевых слов в тексте. Среди изученных алгоритмов выделяются подход TF-IDF и Latent Dirichlet Allocation (LDA). Впоследствии они были применены к корпусу документов на английском языке для выделения ключевых слов. Текст также был предобработан для последующей векторизации.

TF-IDF показал довольно высокое качество в задаче выделения ключевых слов, а также высокую скорость работы и оказался прост в реализации.

Алгоритм LDA был реализован в двух версиях: вручную, с использованием метода коллапсированного приближения и Гиббсовской выборки, а также с помощью решения из библиотеки scikit-learn, которое использует вариационный Байесовский метод. Оба алгоритма показали схожие результаты работы, с той поправкой, что второе решение оказалось быстрее в работе.

Таким образом, TF-IDF больше подходит для быстрого выделения ключевых слов внутри отдельного документа, а LDA — для выявления скрытых тем во всем корпусе.

Тем не менее, есть несколько точек роста. Во-первых, использование лемматизации может устранить шум в данных, что повысит качество применяемых алгоритмов. Во-вторых, LDA, как вероятностный метод, может себя показать лучше на больших объемах текста.

### Список источников

- 1) [Основы Natural Language Processing для текста](#) / Хабр;
- 2) [Обработка естественного языка](#) — Викиконспекты;
- 3) [Извлечение признаков из текстовых данных с использованием TF-IDF](#) / Хабр;
- 4) [Учебник по машинному обучению](#) — Яндекс Образование;
- 5) [Статистика Байеса в ML для самых маленьких](#) / Хабр;
- 6) [Вероятностные модели: от наивного Байеса к LDA, часть 1](#) / Хабр;
- 7) [Вероятностные модели: LDA, часть 2](#) / Хабр;
- 8) [Категоризация текстов и модель LDA](#) — Сергей Николенко, Казанский Федеральный Университет, 2014;
- 9) [Запускаем LDA в реальном мире. Подробное руководство](#) / Хабр;
- 10) [Probabilistic Topic Models](#), Griffiths & Steyvers (2004);
- 11) [Документация scikit-learn](#);
- 12) [Документация NLTK](#).

## Приложения

### 1. Сертификат о прохождении курса



### 2. Текст, на котором обучаются алгоритмы:

#### CHAPTER 1: INTRODUCTION TO MACHINE LEARNING

Machine learning is a subfield of artificial intelligence that focuses on building systems that learn from data. There are three main types: supervised learning, unsupervised learning, and reinforcement learning. Algorithms like linear regression, decision trees, and neural networks form the foundation of modern ML applications. Feature engineering and data preprocessing are critical steps in the ML pipeline.

#### CHAPTER 2: NATURAL LANGUAGE PROCESSING FUNDAMENTALS

Natural Language Processing enables computers to understand human language. Key techniques include tokenization, stemming, and lemmatization. Bag-of-words models and word embeddings like Word2Vec represent text numerically. Applications range from sentiment analysis to machine translation. Transformer architectures have revolutionized NLP in recent years.

#### CHAPTER 3: DIMENSIONALITY REDUCTION TECHNIQUES

Dimensionality reduction methods simplify complex datasets. Principal Component Analysis (PCA) identifies orthogonal axes of maximum variance. t-SNE creates low-dimensional embeddings preserving local structures. Autoencoders use neural networks for nonlinear reduction. These techniques help visualize high-dimensional data and improve model performance.

## CHAPTER 4: CLUSTERING ALGORITHMS

Clustering groups similar data points together. K-means partitions data into spherical clusters based on centroids. Hierarchical clustering builds nested clusters using dendrograms. DBSCAN identifies dense regions separated by sparse areas. Evaluation metrics include silhouette score and Davies-Bouldin index. Applications include customer segmentation and anomaly detection.

## CHAPTER 5: TOPIC MODELING APPROACHES

Topic modeling discovers abstract themes in document collections. Latent Dirichlet Allocation (LDA) is a generative probabilistic model that represents documents as mixtures of topics. Non-Negative Matrix Factorization (NMF) performs dimensionality reduction for topic discovery. Evaluation uses coherence scores and human interpretation. These methods enable automated text analysis at scale.