САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Направление: 01.03.02 Прикладная математика и информатика
ООП: Прикладная математика, фундаментальная информатика и программирование

Применение методов теории игр и машинного обучения в анализе текстов.

Отчет по учебной практике
(дипломная работа)
Версия №1

Научный руководитель:
Буре Владимир Мансурович
Выполнил:
Сыдыгалиева Бегаим Нурбековна
студент группы 21.Б03

Санкт-Петербург
2024 г.

Содержание

ПОСТАНОВКА ЗАДАЧИ

- Тема: Применение методов теории игр и машинного обучения в анализе текстов.
- Объект исследования: учебные тексты
- Цель работы: Разработка и сравнительный анализ методов оценки сложности учебных текстов с использованием подходов теории игр и машинного обучения, а также выявление их эффективности в контексте задач анализа текстовой информации.
- Актуальность:
  - Увеличение объема информации требует эффективных методов анализа.
  - Анализ сложности позволяет адаптировать материалы под уровень подготовки аудитории.

Текст как средство передачи знаний и идей играет ключевую роль в различных сферах человеческой деятельности. В условиях стремительного роста объема информации возникает необходимость в эффективных методах анализа и обработки текстовых данных. Одним из важных аспектов в этой области является оценка сложности текстов, что особенно актуально в лингводидактике. Точная оценка сложности позволяет адаптировать учебные материалы под уровень подготовки целевой аудитории, что способствует повышению эффективности образовательного процесса.

В последние годы методы теории игр и машинного обучения стали активно применяться для анализа текстов, что нашло отражение в развитии области Natural Language Processing (NLP). Теория игр предоставляет математический инструментарий для моделирования задач, таких как анализ игр голосования, что может быть полезно для оценки сложности текстов. Например, подходы теории игр позволяют формализовать задачу выбора оптимального текста для обучения, рассматривая её как игру с несколькими участниками (например, учителями, учениками и текстами).

Машинное обучение, в свою очередь, предоставляет возможности для автоматизации процесса обработки больших объемов текстовых данных и выявления скрытых закономерностей. В частности, NLP-технологии, такие как анализ синтаксических структур, семантическое моделирование и

векторное представление текстов, позволяют создавать модели, способные оценивать сложность текстов на основе их лексических, синтаксических и семантических характеристик. Таким образом, сочетание методов теории игр и машинного обучения открывает новые перспективы для решения задач анализа текстов, включая оценку их сложности.

Таким образом, разработка и сравнительный анализ методов оценки сложности текстов с использованием подходов теории игр и машинного обучения является актуальной задачей. В рамках данной работы будут рассмотрены существующие подходы к оценке сложности, реализованы и протестированы методы на основе теории игр и машинного обучения, а также проведено их сравнение для выявления наиболее эффективных решений.

Глава 1. ТЕОРЕТИКО-МЕТОДОЛОГИЧЕСКИЕ ОСНОВЫ ИССЛЕДОВАНИЯ

1.1 Критерии

В качестве критериев рассматриваются количественные параметры оценки сложности текста на разных уровнях языка: фонологическом, морфологическом, синтаксическом и лексическом. Кроме того, рассмотрены статистические показатели, такие как индексы удобочитаемости (тест Флеша-Кинкейда, LIX, FRE) и индекс SMOG.

Фонологический уровень
Оценка сложности на фонологическом уровне включает анализ структуры слов и предложений:

- Количество слогов.
- Среднее количество слогов на слово.
- Средняя длина предложения.
- Фонетические особенности: сложные фонемы, сочетания, ритмика и интонация.

Синтаксический уровень
На синтаксическом уровне анализируются:

- Связи слов в предложении.
- Наличие сложных синтаксических конструкций.
- Общее количество слов и предложений.
- Средняя длина слов и предложений.
- Количество сложных конструкций (например, придаточных предложений).

Лексический уровень
Основным показателем сложности на лексическом уровне является:

- Коэффициент лексического разнообразия (Type Token Ratio, TTR).

$$TTR = \frac{\text{Количество уникальных слов (Types)}}{\text{Общее количество слов (Tokens)}}$$

Семантический уровень

Темы текстов обуславливаются исходя из возрастных особенностей обучающихся и их уровня владения иностранным языком.

Статистические метрики

- **Индекс удобочитаемости Флеша (FRE)**: Метрика, показывающая сложность восприятия текста по 100-балльной шкале (0 – сложный текст для выпускников профильных вузов, 100 – легкий текст для обучающихся начальной школы).

$$FRE = 206.835 - 1.015 \left(\frac{\text{total words}}{\text{total sentences}} \right) - 84.6 \left(\frac{\text{total syllables}}{\text{total words}} \right)$$

  Где:

  - `total words` – общее количество слов в тексте.
  - `total sentences` – общее количество предложений в тексте.
  - `total syllables` – общее количество слогов в тексте.

- **Тест Флеша-Кинкейда (FKGL)**: Метрика, определяющая потенциальный уровень подготовки обучающихся, для которых предназначен текст.

$$FKGL = 0.39 \left(\frac{\text{total words}}{\text{total sentences}} \right) + 11.8 \left(\frac{\text{total syllables}}{\text{total words}} \right) - 15.59$$

1.2 Методы

1.2.1 Теоретико-игровой подход

Метод оценки сложности текстов основан на методах теории кооперативных игр. В рамках этого подхода текст рассматривается как кооперативная игра, где игроками являются:

- Длины слов в тексте.
- Лексическая разнообразность.

Игра моделируется как **игра голосования**, где ценность каждого игрока (слова или лексической единицы) определяется количеством коалиций, в которых он является ключевым. Для оценки ранга игроков используются два подхода:

1. **Значение Шепли-Шубика**
   Значение Шепли-Шубика — это концепция из теории кооперативных игр, которая позволяет распределить выигрыш между игроками на основе их вклада в коалиции. В контексте текстового анализа значение Шепли-Шубика рассчитывается для каждого игрока (слова или лексической единицы) и отражает его важность в формировании коалиций. Чем выше значение, тем более значимым является игрок для текста.

2. **Индекс Банцафа**
   Индекс Банцафа — это альтернативный метод оценки влияния игроков в игре голосования. Он основан на количестве минимальных выигрывающих коалиций, в которых участвует данный игрок. В контексте текстового анализа индекс Банцафа позволяет определить, насколько часто слово или лексическая единица встречается в ключевых коалициях, что отражает его роль в структуре текста.

Каждому тексту сопоставляется вектор значений Шепли-Шубика или индексов Банцафа. В пространстве векторов проводится ранжирование текстов по сложности на основе экспертных оценок, полученных в данной области. Этот подход позволяет формализовать задачу оценки сложности текста и выявить ключевые факторы, влияющие на его восприятие.

1.2.2 Методы машинного обучения

Обработка естественного языка (NLP) Обработка естественного языка (Natural Language Processing, NLP) — это область искусственного интеллекта, которая фокусируется на возможности машин читать, понимать и извлекать смысл из человеческих языков.

Предобработка данных

- **Приведение символов к одному регистру**: для унификации текста.

- **Токенизация**: разбиение текста на токены (слова, знаки препинания и т.д.).

- **Тегирование частей речи**: определение частей речи в каждом предложении для применения грамматических правил.
- **Лемматизация и стемминг**:
  - **Стемминг**: грубое приведение слов к корням путем обрезания суффиксов.
  - **Лемматизация**: приведение слов к изначальным словоформам с учетом контекста.
- **Удаление стоп-слов**: удаление артиклей, междометий и других ненужных слов.
- **Спелл-чекинг**: автокоррекция слов, написанных неправильно.

Преобразование текста в числовые представления Для анализа текста используются следующие методы:

- **Bag of Words (BoW)**: представление текста в виде вектора частот слов.
- **TF-IDF (Term Frequency-Inverse Document Frequency)**: мера важности слова в контексте документа.

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

где:

- $TF(t, d)$ — частота слова t в документе d.
- $IDF(t)$ — обратная частота документа для слова t, рассчитываемая как:
$$IDF(t) = \log\left(\frac{N}{DF(t)}\right)$$
где:
  - N — общее количество документов.
  - $DF(t)$ — количество документов, содержащих слово t.

Word Embeddings:

- **Word2Vec**: генерация векторных представлений слов на основе контекста.
- **GloVe (Global Vectors for Word Representation)**: метод обучения векторных представлений слов на основе глобальной статистики корпуса.

Глава 2. Классификация текстов относительно сложности: классические методы машинного обучения и глубокое обучение

Классификация текстов является одной из ключевых задач в области обработки естественного языка (NLP). Она заключается в автоматическом отнесении текста к определенной категории или классу на основе его содержания..

2.1 Классические методы машинного обучения

Классические методы машинного обучения остаются актуальными благодаря их простоте, интерпретируемости и эффективности в задачах классификации текстов. Основные подходы включают:

2.1.1 Логистическая регрессия

Логистическая регрессия представляет собой вероятностный метод, который строит линейную модель для предсказания вероятности принадлежности текста к определенному классу. Логистическая регрессия характеризуется высокой интерпретируемостью, что позволяет анализировать веса признаков (например, важность слов в тексте).

$$\mathcal{L}(w, X, y) = \sum_{i=1}^{N} \log(1 + e^{-y_i \langle w, x_i \rangle})$$

$$p = \sigma(\langle w, x_i \rangle)$$

$$V_w L(y, X, w) = -\sum_i x_i (y_i - \sigma(\langle w, x_i \rangle))$$

Преимущества:

- Простота реализации и быстрота обучения.
- Высокая интерпретируемость модели.

2.1.2 Метод опорных векторов (SVM)

Метод опорных векторов (Support Vector Machine, SVM) представляет собой алгоритм, который ищет оптимальную гиперплоскость для разделения классов в многомерном пространстве. SVM эффективно работает с высокоразмерными данными, что делает его подходящим для текстовой

классификации, где каждое слово может быть представлено как признак.

$$F(M) = \max(0, 1 - M)$$

$$L(w, x, y) = \lambda \|w\|_2^2 + \sum_i \max(0, 1 - y_i \langle w, x_i \rangle)$$

$$\nabla_w L(w, x, y) = 2\lambda w + \sum_i \begin{cases} 0, & 1 - y_i \langle w, x_i \rangle \leq 0 \\ -y_i x_i, & 1 - y_i \langle w, x_i \rangle > 0 \end{cases}$$

Преимущества:

- Высокая точность на небольших и средних наборах данных.
- Эффективность в задачах с линейно и нелинейно разделимыми классами (с использованием ядерных функций).

2.1.3 Деревья решений и случайные леса

Деревья решений представляют собой иерархические модели, которые разбивают данные на подмножества на основе значений признаков. Случайные леса — это ансамблевый метод, который объединяет несколько деревьев решений для повышения точности и устойчивости модели.

Преимущества:

- Легкость интерпретации и визуализации.
- Устойчивость к выбросам и пропущенным данным.

Применение:

- Классификация текстов по жанрам.
- Анализ сложности текстов в образовательных целях.

2.2 Глубокое обучение

Глубокое обучение представляет собой современный подход, который использует нейронные сети для анализа текстов. Оно позволяет автоматически извлекать сложные признаки из данных, что делает его особенно эффективным для задач классификации текстов высокой сложности.

2.2.1 Рекуррентные нейронные сети (RNN)

Рекуррентные нейронные сети (Recurrent Neural Networks, RNN) читывают контекст и порядок слов, что позволяет моделировать зависимости между элементами последовательности.

Преимущества:

- Возможность работы с длинными последовательностями.
- Учет контекста при классификации.

2.2.2 Долгая краткосрочная память (LSTM) и GRU

LSTM (Long Short-Term Memory) и GRU (Gated Recurrent Unit) — это улучшенные варианты RNN, которые решают проблему исчезающего градиента и позволяют эффективно обрабатывать длинные последовательности.

Преимущества:

- Устойчивость к длинным последовательностям.
- Высокая точность в задачах, требующих учета контекста.

2.2.3 Transformers

Архитектура Transformers использует механизм self-attention для анализа взаимосвязей между словами в тексте, что позволяет учитывать контекст на больших расстояниях.

Преимущества:

- Высокая точность в задачах классификации.
- Параллелизуемость вычислений.

Attention

Формула механизма внимания для пары запроса (Q), ключа (K) и значения (V):

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

- Q (Query) — вектор запроса, представляющий интересующий нас элемент.

- K (Key) — вектор ключа, с которым сравнивается запрос.
- V (Value) — вектор значения, содержащий информацию, которую мы хотим извлечь.
- QK^T — матричное произведение запросов и ключей, вычисляющее сходство между ними.
- $\frac{QK^T}{\sqrt{d_k}}$ — нормализация сходства с помощью корня из размерности ключей d_k, чтобы избежать переполнения.
- $\text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)$ — функция softmax, которая преобразует сходство в вероятности (веса внимания).
- V — вектор значений, умноженный на веса внимания для получения выходного вектора.

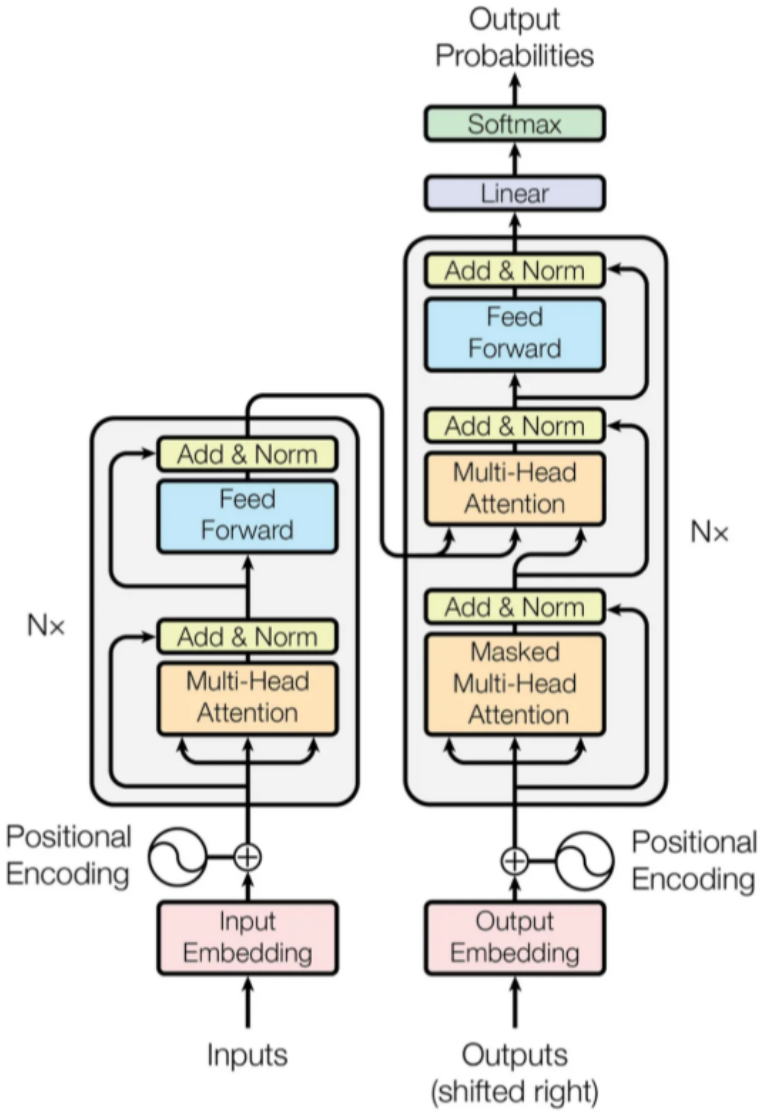

Figure 1: The Transformer - model architecture.

Рис. 1: Архитектура

```
import os
import math
import pandas as pd
import torch
from torch import nn
from torch.utils.data import Dataset, DataLoader
from d2l import torch as d2l
from transformers import BertTokenizer

# Класс для многоголового внимания
class MultiHeadAttention(d2l.Module):
```

```
    def __init__(self, num_hiddens, num_heads, dropout, bias=False, **
        super().__init__()
        self.num_heads = num_heads
        self.attention = d2l.DotProductAttention(dropout)
        self.W_q = nn.LazyLinear(num_hiddens, bias=bias)
        self.W_k = nn.LazyLinear(num_hiddens, bias=bias)
        self.W_v = nn.LazyLinear(num_hiddens, bias=bias)
        self.W_o = nn.LazyLinear(num_hiddens, bias=bias)

    def forward(self, queries, keys, values, valid_lens):
        queries = self.transpose_qkv(self.W_q(queries))
        keys = self.transpose_qkv(self.W_k(keys))
        values = self.transpose_qkv(self.W_v(values))

        if valid_lens is not None:
            valid_lens = torch.repeat_interleave(
                valid_lens, repeats=self.num_heads, dim=0)

        output = self.attention(queries, keys, values, valid_lens)
        output_concat = self.transpose_output(output)
        return self.W_o(output_concat)

    def transpose_qkv(self, X):
        X = X.reshape(X.shape[0], X.shape[1], self.num_heads, -1)
        X = X.permute(0, 2, 1, 3)
        return X.reshape(-1, X.shape[2], X.shape[3])

    def transpose_output(self, X):
        X = X.reshape(-1, self.num_heads, X.shape[1], X.shape[2])
        X = X.permute(0, 2, 1, 3)
        return X.reshape(X.shape[0], X.shape[1], -1)

# Класс для позиционного кодирования
class PositionalEncoding(nn.Module):
    def __init__(self, num_hiddens, dropout, max_len=1000):
        super().__init__()
        self.dropout = nn.Dropout(dropout)
        self.P = torch.zeros((1, max_len, num_hiddens))
        X = torch.arange(max_len, dtype=torch.float32).reshape(
            -1, 1) / torch.pow(10000, torch.arange(
            0, num_hiddens, 2, dtype=torch.float32) / num_hiddens)
```

```
        self.P[:, :, 0::2] = torch.sin(X)
        self.P[:, :, 1::2] = torch.cos(X)

    def forward(self, X):
        X = X + self.P[:, :X.shape[1], :].to(X.device)
        return self.dropout(X)

# Класс для позиционно-зависимой сети
class PositionWiseFFN(nn.Module):
    def __init__(self, ffn_num_hiddens, ffn_num_outputs):
        super().__init__()
        self.dense1 = nn.LazyLinear(ffn_num_hiddens)
        self.relu = nn.ReLU()
        self.dense2 = nn.LazyLinear(ffn_num_outputs)

    def forward(self, X):
        return self.dense2(self.relu(self.dense1(X)))

# Класс для нормализации и добавления
class AddNorm(nn.Module):
    def __init__(self, norm_shape, dropout):
        super().__init__()
        self.dropout = nn.Dropout(dropout)
        self.ln = nn.LayerNorm(norm_shape)

    def forward(self, X, Y):
        return self.ln(self.dropout(Y) + X)

# Класс для блока трансформера
class TransformerEncoderBlock(nn.Module):
    def __init__(self, num_hiddens, ffn_num_hiddens, num_heads,
                 dropout, use_bias=False):
        super().__init__()
        self.attention = MultiHeadAttention(num_hiddens, num_heads,
                                            dropout, use_bias)
        self.addnorm1 = AddNorm(num_hiddens, dropout)
        self.ffn = PositionWiseFFN(ffn_num_hiddens, num_hiddens)
        self.addnorm2 = AddNorm(num_hiddens, dropout)

    def forward(self, X, valid_lens):
        Y = self.addnorm1(X, self.attention(X, X, X, valid_lens))
```

```
        return self.addnorm2(Y, self.ffn(Y))

# Класс для кодировщика трансформера
class TransformerEncoder(d2l.Encoder):
    def __init__(self, vocab_size, num_hiddens, ffn_num_hiddens,
                 num_heads, num_blks, dropout, use_bias=False):
        super().__init__()
        self.num_hiddens = num_hiddens
        self.embedding = nn.Embedding(vocab_size, num_hiddens)
        self.pos_encoding = PositionalEncoding(num_hiddens, dropout)
        self.blks = nn.Sequential()
        for i in range(num_blks):
            self.blks.add_module("block"+str(i),
                TransformerEncoderBlock(
                num_hiddens, ffn_num_hiddens,
                num_heads, dropout, use_bias))

    def forward(self, X, valid_lens):
        X = self.pos_encoding(self.embedding(X) * math.sqrt(self.num_h
        for blk in self.blks:
            X = blk(X, valid_lens)
        return X

# Класс для классификатора
class TextClassifier(nn.Module):
    def __init__(self, vocab_size, num_hiddens, ffn_num_hiddens,
                 num_heads, num_blks, num_classes, dropout):
        super().__init__()
        self.encoder = TransformerEncoder(vocab_size, num_hiddens, ffn
        self.fc = nn.Linear(num_hiddens, num_classes)

    def forward(self, X, valid_lens):
        X = self.encoder(X, valid_lens)
        # Используем только первый токен
        #(классификация по началу текста)
        X = X[:, 0, :]
        return self.fc(X)

# Загрузка данных
class TextDataset(Dataset):
    def __init__(self, data_dir, tokenizer, max_len):
```

```
        self.data_dir = data_dir
        self.tokenizer = tokenizer
        self.max_len = max_len
        self.texts, self.labels = self.load_data()

    def load_data(self):
        texts = []
        labels = []
        label_map = {'A1': 0, 'A2': 1, 'B1': 2, 'B2': 3, 'C1': 4}
        for folder in os.listdir(self.data_dir):
            folder_path = os.path.join(self.data_dir, folder)
            if os.path.isdir(folder_path):
                label = label_map[folder]
                for file_name in os.listdir(folder_path):
                    if file_name.endswith('.txt'):
                        file_path = os.path.join(folder_path, file_nam
                        with open(file_path, 'r', encoding='utf-8') as
                            text = f.read()
                            texts.append(text)
                            labels.append(label)
        return texts, labels

    def __len__(self):
        return len(self.texts)

    def __getitem__(self, idx):
        text = self.texts[idx]
        label = self.labels[idx]
        encoding = self.tokenizer(text, truncation=True, padding='max_
        return encoding['input_ids'].squeeze(0), encoding['attention_m

# Параметры
vocab_size = 30522  # Размер словаря (BERT)
num_hiddens = 768
ffn_num_hiddens = 3072
num_heads = 12
num_blks = 12
num_classes = 5
dropout = 0.1
max_len = 512
batch_size = 16
```

```
num_epochs = 10
learning_rate = 5e-5

# Инициализация токенизатора
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')

# Загрузка данных
train_dataset = TextDataset('path_to_train_data', tokenizer, max_len)
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffl

# Инициализация модели
model = TextClassifier(vocab_size, num_hiddens, ffn_num_hiddens, num_h

# Оптимизатор и функция потерь
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
criterion = nn.CrossEntropyLoss()

# Обучение модели
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model.to(device)

for epoch in range(num_epochs):
    model.train()
    total_loss = 0
    for input_ids, attention_mask, labels in train_loader:
        input_ids, attention_mask, labels = input_ids.to(device), atte
        optimizer.zero_grad()
        outputs = model(input_ids, attention_mask)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        total_loss += loss.item()
    print(f'Epoch {epoch+1}, Loss: {total_loss/len(train_loader)}')
```

Список литературы

[1] Хитрый А. В., Мазалов В. В., Буре Н. А., Дробная П. В. Теоретико-игровая оценка сложности учебных текстов // Вестник Санкт-Петербургского университета. Прикладная математика. Информатика. Процессы управления. 2023. Т. 19. Вып. 4. С. 509–521. URL: https://doi.org/10.21638/11701/spbu10.2023.407

[2] Mazalov V. V. Matematicheskaya teoriya igr i prilozheniya. Uchebnoe posobie. 2-e izd. [Mathematical game theory and applications. Textbook]. 2nd ed. St. Petersburg, Lan' Publ., 2016, 448 p. (In Russian)

[3] Molinero X., Laamiri A., Riquelme F. Readability and power indices. The Fifteenth International Conference on Game Theory and Management (GTM 2021). St. Petersburg, 2021, p.

[4] Лапошина, А. Н. Лингводидактическое обоснование применения автоматической оценки сложности учебного текста в преподавании РКИ: диссертация на соискание ученой степени кандидата педагогических наук / Лапошина Антонина Николаевна. – Москва, 2023. – 189 с.

[5] Oborneva I. V. Matematicheskaya model' ocenki uchebnyh tekstov [A mathematical model forevaluating instructional texts]. Vestnik of Moscow State Pedagogical University. Series Information and Informatization of education, 2005, no. 1 (4), pp. 141–147. (In Russian)

[6] Coleman M., Liau T. L. A computer readability formula designed for machine scoring. Journal of Applied Psychology, 1975, no. 60, pp. 283–284.

[7] Texts for teaching Russian as a foreign language. [Электронный ресурс]: URL: https://github.com/arkty/ru_learning_data

[8] Лапошина А. Н. Корпус текстов учебников РКИ как инструмент анализа учебных материалов. Русский язык за рубежом. 2020. № 6 (283). С. 22-28

[9] Flesch R. A new readability yardstick. Journal of Applied Psychology, 1948, no. 3, pp. 221–233.

[10] Kuratov, Y. Adaptation of deep bidirectional multilingual transformers for russian language [Текст] / Y. Kuratov, M. Arkhipov // arXiv preprint arXiv:1905.07213. — 2019.

[11] Huggingface's transformers: State-of-the-art natural language processing [Текст] / T. Wolf [и др.] // arXiv preprint arXiv:1910.03771. — 2019.

[12] Loshchilov, I. Decoupled weight decay regularization [Текст] / I. Loshchilov, F. Hutter // arXiv preprint arXiv:1711.05101. — 2017.