17. Практическое занятие: Техника автоматической коррекции текста. Поиск ошибок и восстановление

Цель занятия

Изучить алгоритмы обнаружения и исправления ошибок в тексте, освоить работу с расстоянием редактирования и научиться применять вероятностные модели для выбора наиболее подходящих кандидатов на исправление.

Теоретический минимум

Автоматическая коррекция (Spell Checking) обычно состоит из трех этапов:

1. **Обнаружение (Detection):** поиск слов, отсутствующих в словаре или нарушающих контекст.
2. **Генерация кандидатов (Candidate Generation):** создание списка слов, близких по написанию к ошибочному (обычно с расстоянием Левенштейна 1 или 2).
3. **Ранжирование (Ranking):** выбор лучшего варианта на основе частотности слова или контекста (модели N-грамм).

Задание 1. Поиск ошибок через проверку по словарю

Самый простой способ найти ошибку — проверить, есть ли слово в эталонном списке.

Инструкция:

Создайте мини-словарь и напишите функцию, которая находит слова, в которых, вероятно, допущена ошибка.

Python
```
dictionary = {"мама", "мыла", "раму", "папа", "купил", "хлеб"}

def find_typos(text, vocab):
    tokens = text.lower().split()
    typos = [word for word in tokens if word not in vocab]
    return typos

test_text = "мама мыла рому а папа купил хлюб"
print("Возможные ошибки:", find_typos(test_text, dictionary))
```

Задание 2. Генерация кандидатов (Расстояние Левенштейна)

Для исправления слова «хлюб» нам нужно найти ближайшие к нему слова. Чаще всего ошибки возникают из-за пропуска буквы, лишней буквы или замены одной буквы на другую.

Инструкция:

Используйте библиотеку textdistance или Levenshtein для поиска слов из словаря, которые отличаются от «хлюб» на 1 действие.

Python
```
import textdistance

target = "хлюб"
candidates = [word for word in dictionary if
textdistance.levenshtein(target, word) <= 1]

print(f"Кандидаты для '{target}':", candidates)
```

Задание 3. Вероятностная модель коррекции (Питер Норвиг)

Если кандидатов несколько, мы выбираем тот, который чаще встречается в языке. Это классический подход, предложенный Питером Норвигом.

$$P(c|w) \propto P(w|c) P(c)$$

Где $P(c)$ — вероятность слова в языке, а $P(w|c)$ — вероятность того, что при написании слова c будет допущена ошибка w.

Инструкция:

Используйте библиотеку pyspellchecker для автоматического исправления предложения.

Python
```
from spellchecker import SpellChecker

spell = SpellChecker(language='ru')

words = ["мама", "мыла", "раму", "интилект", "прагромирование"]
corrected = [spell.correction(word) for word in words]

print("Результат коррекции:", corrected)
```

Задание 4. Контекстная коррекция и опечатки, образующие реальные слова

Сложнее всего исправить ошибку, если в результате получилось другое существующее слово (например, «мама мыла рому» вместо «раму»). Обычный словарь здесь не поможет.

Инструкция:

Объясните, как использование N-грамм (биграмм) помогает решить эту проблему.

Пример: Какая пара слов более вероятна в русском корпусе: «мыла раму» или «мыла рому»?

Задание 5. Современные методы: Библиотека PyHunspell и Transformers

Для профессиональной коррекции используются морфологические движки (Hunspell) или нейросетевые модели (BERT), которые «видят» контекст всего предложения.

Инструкция:

Попробуйте исправить фразу "привет как дила" с помощью модели Sage от SberDevices (концептуально). Подумайте, почему нейросеть лучше справится с исправлением сокращений и сленга.

Python

```
# Пример использования контекстных моделей (требует библиотеку sage-spellcheck)
# from sage.speller import Speller
# speller = Speller.from_pretrained("sberbank-ai/sage-m2m-no-edits")
# print(speller.scale("привет как дила"))
```

Контрольные вопросы

1. Почему расстояние Левенштейна больше 2 редко используется для генерации кандидатов?
2. В чем разница между "опечаткой" (typo) и "грамматической ошибкой" (grammar error) с точки зрения NLP?
3. Как раскладка клавиатуры (QWERTY/ЙЦУКЕН) может помочь в ранжировании кандидатов на исправление?

Итог работы

Вы изучили механизмы, которые стоят за современными системами автодополнения и проверки правописания. Умение настраивать такие системы необходимо для создания качественных интерфейсов ввода данных и предварительной очистки корпусов перед анализом.