Нечеткая логика для обработки лингвистических неопределенностей

- Нечеткая логика и теория нечетких множеств, лингвистические переменные.
- Построение системы для интерпретации качественных оценок в отзывах и рецензиях. Реализация механизма вывода на основе нечетких правил для анализа тональности с учетом неопределенных формулировок.

Левая теория по нечеткой логике.
Лекция № 1 Нечеткие множества.pdf
Лекция № 2 Нечеткая логика.pdf
Лекция № 3 Нечеткое управление.pdf

Лекция С.Н. Позднякова по Нечеткой Логике (2021):
Лекция МЛиТА 24.05.2021 в Zoom (алгоритмически неразрешимые задачи)

~~Based (! РУКОСЛОВАРЬ !) pipeline:~~

1. **Вход**: текст отзыва / рецензии.
2. **Предобработка текста** (чисто концептуально):
   - выделение ключевых фраз: «очень понравилось», «в целом нормально», «ожидал большего», «ужасная доставка», «вкус отличный», «так себе»;
   - определение базовой тональности фраз (положительная/ отрицательная/нейтральная);
   - поиск модификаторов: «очень», «немного», «в целом», «скорее», «чуть-чуть».
3. **Преобразование в значения лингвистических переменных**

- из фраз «очень понравилось», «обожаю», «супер» → высокая положительная тональность + высокая интенсивность эмоций;
- «нормально», «в целом неплохо» → слабо положительная тональность + низкая/умеренная интенсивность + неуверенность.

4. **Нечеткий вывод на основе правил**:
   - есть база правил вида «ЕСЛИ (тональность по словам = скорее положительная) И (интенсивность = низкая) ТО (итог = слабоположительная)»;
   - правила комбинируются, давая нечеткий результат.
5. **Дефаззификация** (если нужно число):
   - переводим итоговую нечеткую оценку в число, например от −1 до 1 (общая тональность) или от 0 до 100 (индекс удовлетворенности).

Вывод: система — это цепочка «текст → лингвистические оценки → нечеткие правила → итоговая тональность», где нечеткая логика служит сердцем интерпретации неопределённых формулировок.

Если коротко, **перед кодом** нам надо чётко решить:

1. Какие *численные признаки* идут в fuzzy-часть.
2. Как устроен *словарь* (типы слов, поля, коэффициенты).
3. Список *лингвистических переменных* и *термов* для каждой.
4. Конкретные *функции принадлежности* (формы и числа).
5. *Выходную переменную*, её термы и диапазон.
6. Тип *вывода* (Мамдани, min/max) и *дефаззификации* (центр тяжести).
7. Формат и первичный набор *правил*.
8. Формат *конфигов* и небольшой тестовый набор отзывов.

Процесс

---

https://github.com/mikhailshceglov/fuzzy_reviews

Этап 1

Главная задача этапа: получит словарь с леммами, полярность, частотой.
Мы берём уже готовую русскую BERT-подобную модель. На вход она ждёт **текст предложения**, на выход даёт **логиты по классам** (обычно 3 класса: NEG / NEU / POS). Превращаем классы в число ∈ [-1, 1]. Для каждого класса задаём **вес** в диапазоне [-1, 1]. Если у модели явные метки:

- 0 → NEG → вес -1.0
- 1 → NEU → вес 0.0
- 2 → POS → вес +1.0
  мы строим вектор весов (по id меток).
  Считаем **ожидаемое значение** по этим весам:

$$\text{score}(s) = \sum_i p_i \cdot w_i$$

где

- pi — вероятность класса i,
- wi — вес класса i в [-1, 1]
  На выходе для **каждого предложения** `s` получаем `sentiment_score(s) ∈ [-1, 1]`:
- ближе к -1 → уверенно негативное,
- около 0 → нейтральное/смешанное,
- ближе к +1 → уверенно позитивное.
  Глобальная идея: мы не просто берём класс POS/NEG, а считаем число, которое учитывает распределение вероятностей по классам — если модель сомневается, score ближе к 0.

2. Как из предложений собрать полярность для лемм. Теперь у нас для каждого предложения есть:

- сам текст предложения,
- множество лемм в нём (`lemmas_in_sentence`),
- `sentiment_score` для этого предложения.
  если слово повторилось 3 раза в одном и том же предложении, мы считаем, что оно встретилось **1 раз в этом предложении**.
  Это важно:
- мы хотим `freq(w)` = «сколько **разных предложений** содержит лемму w», а не «общее число вхождений» — иначе одно длинное предложение с повтором «ужасный» могло бы внести непропорционально большой вклад. если слово повторилось 3 раза в одном и том же предложении, мы считаем, что оно встретилось **1 раз в этом предложении**.

$$\text{freq}(w) = \text{число предложений, где есть } w$$

$$\text{polarity}(w) = \frac{1}{\text{freq}(w)} \sum_{s \ni w} \text{score}(s)$$

Замечания 1 этапа (возможные доработки):

- Проверить корректность словаря (корреляция между двумя разделенными reviews?, кор тест короче)

Этап 2

Хотим получить на выходе: features.jsonl — это JSON Lines: каждая строка файла = один отзыв в виде JSON-объекта с числовыми признаками. Пример (условный, у тебя будет так же по структуре):


```
{"id":0,"pos":1.25,"neg":0.10,"score_crisp":0.851064,"intensi
ty":0.4375,"tokens_alpha":18,"lex_hits":7,"coverage":0.388889
```

```
,"intensifiers_count":1,"downtoners_count":0,"hedges_count":0
,"negations_count":0,"contrast_count":1}
```


Что значит каждое поле

- **id** — номер отзыва (строка в исходном `rureviews_reviews.txt`).
- **pos** — суммарный позитив по словам из лексикона (складываем все `polarity > 0` после учёта модификаторов).
- **neg** — суммарный негатив (складываем `-polarity` для `polarity < 0` после модификаторов). Всегда ≥ 0.
- **score_crisp** — главный “чёткий” скор тональности:

$$score = \frac{pos - neg}{pos + neg + \varepsilon}$$

  Диапазон примерно **[-1; 1]**: ближе к +1 — позитив, к -1 — негатив.
- **intensity** — средняя “сила эмоций” по найденным словам: среднее значение `|polarity|` по тем леммам, которые нашлись в словаре (после модификаторов). **0..1**.

Качество покрытия словарём:

- **tokens_alpha** — сколько слов (токенов) в отзыве мы вообще считали “словами” (буквы).
- **lex_hits** — сколько из этих токенов реально нашли в твоём авто-словаре (после лемматизации/нормализации).
- **coverage** — доля покрытия словарём: `lex_hits / tokens_alpha`. Чем выше, тем “увереннее” лексикон описывает этот отзыв.

Счётчики модификаторов:

- **intensifiers_count** — сколько усилителей встретилось (“очень”, “крайне”…).
- **downtoners_count** — сколько ослабителей/смягчителей (“немного”, “слегка”…).

- **hedges_count** — сейчас равен `downtoners_count` (я их приравнял; если захочешь — выделим отдельный список “хеджеров”).
- **negations_count** — сколько отрицаний (“не”, “ни”…).
- **contrast_count** — сколько маркеров контраста (“но”, “однако”…).

Улучшения 2 этапа:

- Умнее (чем руками) сделать словарь модификаторов (yaml).
- Убрать лишние численны характеристики (если они появятся).