Министерство Науки и Высшего Образования Российской Федерации

Национальный Исследовательский Институт Высшая Школа Экономики

Факультет Компьютерных Наук

Школа Анализа Данных и Искусственного Интеллекта

ДОМАШНЕЕ ЗАДАНИЕ №1

СЕГМЕНТАЦИЯ И МОРФОЛОГИЧЕСКИЙ АНАЛИЗ ТЕКСТА, СТАТИСТИКА

Компьютерная лингвистика и анализ текстов

Студент *М.Д. Кирдин*

Преподаватель *Е.И. Большакова*

Москва, 2025г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В данной работе была поставлена цель провести исследование качества разрешения морфологической омонимии одного из морфоанализаторов для русского языка, подключив его к своей программе. Был выбран *pymorphy2* вследствие легкости его использования.

Для оценки были использованы материалы соревнования по морфологической разметке *MorphoRuEval — 2017*, а именно – тексты в разметке *OpenCorpora*. Они состоят из объединения материалов *Live Journal from General Internet-Corpus of Russian, 30 million words*, *Librusec, 300 million words* и *Social networks, 50 million words*. В этой работе на стояла цель выяснить связь качества морфологического анализа текста в зависимости от тематики, поэтому тексты были объединены в один файл.

1 ХОД РАБОТЫ

Была поставлена задача создания программы, которая сравнивает граммемы из размеченного текста с граммемами предложенными морфоанализатором. Для этого была реализована программа на языке *Python*, доступная в приложении А.

Разметка текстовых данных не совпадала с разметкой, предлагаемой морфоанализатором. По этой причине было необходим перевод морфологических тегов. В данном случае большая часть тегов отличилась лишь названием, поэтому было решено использовать словари, связывающие теги из корпуса с тегами из морфоанализатора ($POS\_ALIASES$ и $VALUE\_ALIASES$). Стоит заметить, что в разметке текста, в отличие от предлагаемой морфоанализатором, не различались прилагательные и причастные, форма (краткая/полная и компаратив/суперлатив) была тегом самого прилагательного (в морфоанализаторе краткие и полные прилагательные считались различными частями речи), не различались герундий и инфинитив глаголов, а также настоящее и будущее время были объединены в один тег.

В данной работе были выбраны следующие метрики качества разрешения морфологической омонимии: **точность по меткам**, **точность по словам** и **точность по предложениям**. При этом рассматривались только только следующие части речи: имя существительное, имя прилагательное, наречие, числительное, местоимение. Для точности по меткам было посчитано отношение количества совпадающих меток и общего количества меток, по словам — отношение количества слов с полностью совпадающими метками и общего числа рассмотренных слов, для предложений — отношение количества предложений в которых полностью совпали метки каждого рассмотренного слова к общему числу предложений. В ходе работы морфоанализатор предлагает несколько вариантов разрешения омонимии с соответствующими им вероятностями, поэтому было принято решение рассматривать исключительно наиболее вероятные из них.

В результате работы программы были получены следующие значения:

- точность по меткам — $89,36\%$;
- точность по словам — $73,91\%$;
- точность по предложениям — $30,79\%$.

РЕЗУЛЬТАТЫ

Согласно полученным результатам, морфоанализатор *pymorphy2* относительно ставит теги с высокой точностью, но

ПРИЛОЖЕНИЕ А

Листинг А.1. Программа run.py

```
from argparse import ArgumentParser,
  ArgumentDefaultsHelpFormatter
from pymorphy2 import MorphAnalyzer


def get_sents(infile: str):
    """
    Get word forms from a corpus at `infile`.
    """
    with open(infile, "r", encoding="utf8") as fin:
        answer, curr_sent = [], []
        for line in fin:
            line = line.strip()
            if line == "":
                if len(curr_sent) > 0:
                    answer.append(curr_sent)
                curr_sent = []
                continue
            splitted = line.split("\t")
            if len(splitted) == 10:
                word = splitted[1]
                lemma = splitted[2]
                pos = splitted[3]
                tags = splitted[4:]
            elif len(splitted) == 9:
                word = splitted[1]
                pos = splitted[2]
                tags = splitted[3:]
                lemma = None
            else:
                raise ValueError(
```

```
                    f"Each line should have 9 or 10
                       columns. Got {len(splitted)}"
                )
            if tags[1] != "_":
                tags = dict(elem.split("=") for elem in
                    tags[1].split("|"))
            else:
                tags = dict()
            curr_sent.append([word, pos, tags, lemma])
    if len(curr_sent) > 0:
        answer.append(curr_sent)
    return answer


def get_cats_to_measure(pos):
    """
    In the corpus used in this task there are several
      POS that are intended to be used, which are
      speciefied below. Note that several POS are
      merged into one (e.g. participles and adjectives
      ) in the corpus we used.
    """
    if pos == "NOUN":
        return ["Animacy", "Gender", "Number", "Case"]
    elif pos == "ADJ":
        return ["Gender", "Number", "Case", "Variant",
          "Degree"]
    elif pos == "PRON":
        return ["Gender", "Number", "Case"]
    elif pos == "DET":
        return ["Gender", "Number", "Case"]
    elif pos == "VERB":
        return ["Aspect", "Gender", "Number", "VerbForm
          ", "Mood", "Tense"]
```

```
    elif pos == "ADV":
        return ["Degree"]
    elif pos == "NUM":
        return ["Gender", "Case", "NumForm"]
    else:
        return []


# Due to the issues stemming from the way pymorphy2
  tags work, we have to introduce
# the following dictionaries so that tag comparison
  works correctly.
POS_ALIASES = {
    "ADJ": ["ADJF", "ADJS", "COMP", "PRTF", "PRTS"],
    "ADV": ["ADVB"],
    "NOUN": ["NOUN"],
    "VERB": ["INFN", "GRND"],
    "DET": ["NPRO"],
    "PRON": ["NPRO"],
    "NUM": ["NUMR"],
    "PROPN": ["NOUN"],
}


VALUE_ALIASES = {
    "short": ["ADJS", "PRTS"],
    "cmp": ["COMP"],
    "sup": ["ADJF"],
    "pos": ["ADVB"],
    "anim": ["anim"],
    "inan": ["inan"],
    "perf": ["perf"],
    "imp": ["impf"],
    "nom": ["nomn"],
```

```
    "gen": ["gent"],
    "dat": ["datv"],
    "acc": ["accs"],
    "loc": ["loct"],
    "ind": ["indc"],
    "ins": ["ablt"],
    "fem": ["femn"],
    "masc": ["masc"],
    "neut": ["neut"],
    "sing": ["sing"],
    "plur": ["plur"],
    "Brev": ["Short", "Brev"],
    "Short": ["Short", "Brev"],
    "past": ["past"],
    "notpast": ["pres", "futr"],
    "inf": ["INFN"],
    "1": ["1per"],
    "2": ["2per"],
    "3": ["3per"],
}


def count_matching_tags(pos: str, first: dict, second:
  MorphAnalyzer.TagClass) -> int:
    """
    Count the number of matching tags
    """
    ans = 0
    cats_to_measure = get_cats_to_measure(pos)
    for cat, value in first.items():
        if cat in cats_to_measure:
            ans += set(VALUE_ALIASES.get(value.lower(),
                [])) in second
    return ans
```

```
if __name__ == "__main__":

    parser = ArgumentParser(formatter_class=
      ArgumentDefaultsHelpFormatter)
    parser.add_argument(
        "textpath", action="store", help="Path to the
          text to be parsed."
    )
    args = vars(parser.parse_args())

    SENTS = get_sents(args["textpath"])
    MorphoAnalyzer = MorphAnalyzer()

    corr_sents = 0
    corr_words = 0
    corr_words_prev = 0
    total_words = 0
    total_words_prev = 0
    corr_tags = 0
    total_tags = 0

    for SENT in SENTS:

        corr_words_prev = corr_words
        total_words_prev = total_words

        for WORD in SENT:

            # We do not consider punctiation, particles
              , conjugates etc.
            if get_cats_to_measure(WORD[1]):
```

```
                pymorphy_tags = MorphoAnalyzer.parse(
                  WORD[0])[0].tag

                if not (WORD[1] == "NUM" and WORD[2].
                  get("Form", False)) and (
                    str(pymorphy_tags.POS) == WORD[1]
                    or str(pymorphy_tags.POS) in
                      POS_ALIASES.get(WORD[1])
                ):

                    eq_tags = count_matching_tags(WORD
                      [1], WORD[2], pymorphy_tags)
                    all_tags = len(WORD[2])

                    corr_tags += eq_tags
                    total_tags += all_tags

                    corr_words += all_tags == eq_tags
                    total_words += 1

    if corr_words - corr_words_prev == total_words
      - total_words_prev:
        corr_sents += 1

print(f"Accuracy(correct sents / all sents): {
  corr_sents / len(SENTS) * 100:0.2f}%")
print(
    f"Accuracy(correct words / all words): {
      corr_words / total_words * 100:0.2f}%"
)
print(f"Accuracy(correct tags / all tags): {
  corr_tags / total_tags * 100:0.2f}%")
```