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

ВЕДЕНИЕ 3

1 ХОД РАБОТЫ 4

РЕЗУЛЬТАТЫ 6

ЗАКЛЮЧЕНИЕ 7

ПРИЛОЖЕНИЕ А 8

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**, точность по **предложениям** и точность по каждой из **неслужебных частей речи**. Неслужебными считались: существительные, прилагательные, наречия, числительные, местоимения и детерминативы. Для точности по меткам было посчитано отношение количества совпадающих меток и общего количества меток, по словам и частям речи — отношение количества слов с полностью совпадающими метками и общего числа рассмотренных слов, по предложения — 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.

С результатами работы программы можно ознакомиться в таблице 1.

Параметр	Правильно определено морфоанализатором	Общее число	Точность
предложения	11856	38508	0,3079
слова	171119	231537	0,7391
метки	768528	860008	0,8936
существительные	102735	121387	0,8463
прилагательные	31533	47054	0,6701
местоимения	1724	8793	0,1961
детерминативы	60	158	0,3797
глаголы	22770	41277	0,5516
наречия	11626	11626	1,0
числительные	671	1242	0,5402

Таблица 1. Метрики, полученные в результате работы программы

РЕЗУЛЬТАТЫ

Согласно таблице 1, наилучшее качество разрешения омонимии наблюдается у наречий. Это объясняется тем, что в корпусе наречия либо не имели тегов, либо имели единственный тег — степень. Также можно заметить, что среди существительных и прилагательных наивысшее качество разрешения омонимии. Для остальных местоимений и детерминативов точность разрешения омонимии меньше 50%.

ЗАКЛЮЧЕНИЕ

Морфоанализатор *pymophy2* стоит использовать для автоматического разрешения морфологической омонимии среди существительных и прилагательных. При этом стоит заметить, что он расставляет метки в целом с высокой точностью.

ПРИЛОЖЕНИЕ А

Листинг А.1. Программа run.py

```
"""
Use python run.py -h in order to look up information
   about arguments.
"""
from argparse import ArgumentParser,
   ArgumentDefaultsHelpFormatter
from pymorphy2 import MorphAnalyzer
from pandas import DataFrame
from numpy import zeros


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()

    df_total = DataFrame(
        data=zeros((1, 10)),
        columns=[
            "sentences",
            "words",
            "tags",
            "NOUN",
            "ADJ",
            "PRON",
            "DET",
            "VERB",
            "ADV",
            "NUM",
```

```
    ],
)
df_corr = df_total.copy()

corr_words_prev = 0
total_words_prev = 0

df_total.at[0, "sentences"] = len(SENTS)

for SENT in SENTS:

    corr_words_prev = df_corr.at[0, "words"]
    total_words_prev = df_total.at[0, "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])
```

```
                    df_corr.at[0, "tags"] += eq_tags
                    df_total.at[0, "tags"] += all_tags

                    if all_tags == eq_tags:
                        df_corr.at[0, "words"] += 1
                        df_corr.at[0, WORD[1]] += 1

                    df_total.at[0, WORD[1]] += 1
                    df_total.at[0, "words"] += 1

        if (
            df_corr.at[0, "words"] - corr_words_prev
            == df_total.at[0, "words"] -
              total_words_prev
        ):
            df_corr.at[0, "sentences"] += 1

    print(df_corr)
    print(df_total)
    print(df_corr.div(df_total))
```