Programming Assignment № 3

Nikki Kyllonen - LING 5801 04/12/2020

Part 1: Regular Expression Tagger

Listing 1: Accuracy of `regex_tagger`**.**

```
Regexp_tagger accuracy with dev_data: 0.5039006232440064
Regexp_tagger accuracy with train_data: 0.4870478397071626
Regexp_tagger accuracy with test_data: 0.4909984639016897
```

Listing 2: Regular Expressions used for `regex_tagger`**.**

```
patterns = [
        (r'(?:\.|\?|\!)', '.'),                     # sentence terminator
        (r'(?:[Bb][Ee]\b)', 'BE'),                  # verb 'to be'
        (r'\b(([Aa]+[Nn]*)|([Nn][Oo])|([Tt][Hh])|([y][Ee])|([Ee]very))\b', 'AT'), # articles
        (r'(?:[Ww]ere[^n])','BED'),                 # verb 'to be' past tense 2nd sing
        (r'\b(?:[Ww][Aa][Ss])\b', 'BEDZ'),          # verb 'to be' past tenst 1st/3rd sing
        (r'\b(?:[Aa][Mm])\b', 'BEM'),               # verb 'to be' present tense 1st sing
        (r'\b(?:[Ii][Ss])\b', 'BEZ'),               # verb 'to be' present tense 3rd sing
        # conjuctions, coordinating
        (r'\b(?:([Aa][Nn][Dd])|([Oo][Rr])|([Bb]ut)|([Pp]lus)|([Nn]*[Ee]ither)|([Nn]or)|([Yy]et)|([Mm]inus))\b','CC'),
        (r'\b(?:[Bb][Ee]+[Nn])\b','BEN'),           # verb 'to be', past part
        # prepositions
        (r'\b(?:([Ff][Oo][Rr])|([Ii][Nn]([Tt][Oo])*)|([Oo]([Ff]|[Nn]|(ut)|(ver)))|([Bb]([Yy]|efore|etween))|((([Cc]onsider)|([Rr]egard)|([Aa]ccord)|([Ii]nclud))ing)|([Aa]((mong)|(gainst)|([Ss])|(fter)|([Tt])    ))|([Tt]((([Oo])|(hrough)|(han)|(oward)))|([Ww]ith(out)*)|([Uu]((nder)|(pon)))|([Dd](uring|espite))|([Ss]ince)|([Pp]er)|([Ee]xcept))\b','IN'),
        # pronouns, nominal
        (r'\b(?:([Nn](one|othing|obody|o-one|othin)|([Ss](omething|omeone))|([Aa](nyone|nybody|nything)))|([Ee]very(thing|body|one))|[Oo]ne)\b','PN'),
        # pronouns, personal, nominative
        (r'\b(?:([Ii][Tt])|([Tt]*[Ss]*[Hh][Ee]+))\b','PPS'),

        ## original patterns ##
        (r'.*ing$', 'VBG'),                         # gerunds
        (r'.*ed$', 'VBD'),                          # simple past
        (r'.*es$', 'VBZ'),                          # 3rd singular present
        (r'.*ould$', 'MD'),                         # modals
        (r'.*\'s$', 'NN$'),                         # possessive nouns
        (r'.*s$', 'NNS'),                           # plural nouns
        (r'^-?[0-9]+(.[0-9]+)?$', 'CD'),            # cadinal numbers
        (r'.*', 'NN'),                              # nouns (default)
]
```

Part 2: Transformation-based Tagger

Listing 3: Accuracy of `brill_tagger` using `regex_tagger` with a max of 25 rules.

```
Brill_tagger accuracy with dev_data: 0.655404538129127
Brill_tagger accuracy with train_data: 0.6394370831419699
Brill with REGEX tagger accuracy with test_data: 0.6433179723502304
```

Listing 4: Rule templates used for my `brill_tagger`.

```
templates = [
         ## original templates ##
         Template(Pos([-1])),                # previous  POS tag
         Template(Pos([-1]), Word([0])),     # previous POS tag + current word

         ## my new templates ##
         Template(Pos([-2]), Pos([-1])),     # previous two POS tags (conjunctive)   (0%)
         Template(Pos([-2, -1])),            # previous two POS tags (disjunctive)   (<2%)
         Template(Word([0]), Word([-1])),    # current word + previous word          (0%)
         Template(Pos([-2]), Word([0])),     # prev prev POS tag + current word      (<1%)
         Template(Word([-1])),               # previous word                         (<0.1%)
         Template(Pos([-1]), Word([-1])),    # previous POS tag + previous word      (0%)
         #Template(Word([0]), Word([1]))      # current word + next word              (<0%)
         Template(Pos([-1]), Pos([0])),      # previous POS tag + current POS tag    (0%)
         #Template(Pos([-2]))                 # prev prev POS tag                     (<0%)
         Template(Pos([-3,-2,-1])),          # previous POS tags (disjunctive)       (<1%)
         Template(Pos([-2]), Pos([1])),      # previous POS tag + next POS tag       (<1%)
         Template(Pos([1])),                 # next POS tag                          (<0%)
         Template(Word([-2,-1])),            # previous two words (disjunctive)      (<0.1%)
         Template(Word([0])),                # current word                          (<3%)
         Template(Word([0]), Word([-1]), Pos([-1])) # current + prev word + prev POS (0%)
]
```

Part 3: Evaluation and Exploration

As can be seen in **Listing 1**, the accuracy of the `regexp_tagger` on the `test_data` was about 49%. This accuracy improved by over 14% with the addition of transformational rules. The improved accuracy of the transformation-based, `brill_tagger` can be seen in **Listing 3**. The transformational rules did not require as much thought and consideration to generate as the regular expressions. To create the regular expressions, I sifted through the tags and descriptions output by running `nltk.help.brown_tagset()`. I then used the RegExr website to help me create regular expressions that matched specific words. This was tedious and felt very inflexible and inextensible, but it worked to create a decent accuracy out of just over a dozen regular expressions.

Listing 5: Template statistics of `brill_tagger` using the `regex_tagger`. Generated using `.print_template_statistics()`.

```
TEMPLATE STATISTICS (TRAIN)  7 templates, 25 rules)
TRAIN ( 145883 tokens) initial 74831 0.4870 final: 52600 0.6394
#ID | Score (train) |  #Rules     | Template
--------------------------------------------
013 | 16239   0.730 |  18   0.720 | Template(Word([0]))
009 |  1968   0.089 |   2   0.080 | Template(Pos([-3, -2, -1]))
006 |  1503   0.068 |   1   0.040 | Template(Word([-1]))
011 |  1166   0.052 |   1   0.040 | Template(Pos([1]))
003 |   594   0.027 |   1   0.040 | Template(Pos([-2, -1]))
000 |   392   0.018 |   1   0.040 | Template(Pos([-1]))
012 |   369   0.017 |   1   0.040 | Template(Word([-2, -1]))

UNUSED TEMPLATES (8)
001 Template(Pos([-1]),Word([0]))
002 Template(Pos([-2]),Pos([-1]))
004 Template(Word([0]),Word([-1]))
005 Template(Pos([-2]),Word([0]))
007 Template(Pos([-1]),Word([-1]))
008 Template(Pos([-1]),Pos([0]))
010 Template(Pos([-2]),Pos([1]))
014 Template(Word([0]),Word([-1]),Pos([-1]))
```

Interestingly enough, **Listing 5** shows that the most useful template was the template which only referenced the current word. 18/25 of the rules generated were created using this incredibly simple template. This template reminds me of how I approached the regular expression tagger, where I mostly looked for specific words which I knew would most often be tagged as a certain tag. As shown in **Listing 6**, this template transformed 6 punctuation symbols and 12 commonly used words, such as *that*, *as*, and *his*.

Listing 6: Descriptions of the 25 rules learned by the `brill_tagger` using the `regex_tagger`.

```
(   Rule('013', 'NN', ',', [(Word([0]),',')]),
    Rule('006', 'NN', 'VB', [(Word([-1]),'to')]),
    Rule('009', 'VBZ', 'NNS', [(Pos([-3, -2, -1]),'NN')]),
    Rule('011', 'IN', 'TO', [(Pos([1]),'VB')]),
    Rule('013', 'NN', '``', [(Word([0]),'``')]),
    Rule('013', 'NN', "''", [(Word([0]),"''")]),
    Rule('013', 'NN', 'CS', [(Word([0]),'that')]),
    Rule('009', 'VBD', 'VBN', [(Pos([-3, -2, -1]),'NN')]),
    Rule('013', 'IN', 'CS', [(Word([0]),'as')]),
    Rule('013', 'NNS', 'PP$', [(Word([0]),'his')]),
    Rule('013', 'NN', 'BER', [(Word([0]),'are')]),
    Rule('013', 'NN', 'MD', [(Word([0]),'will')]),
    Rule('003', 'NN', 'VB', [(Pos([-2, -1]),'MD')]),
    Rule('013', 'NN', 'IN', [(Word([0]),'from')]),
    Rule('013', 'NNS', 'HVZ', [(Word([0]),'has')]),
    Rule('013', 'NNS', 'DT', [(Word([0]),'this')]),
    Rule('013', 'NN', '.', [(Word([0]),';')]),
    Rule('013', 'NN', '--', [(Word([0]),'--')]),
    Rule('013', 'NN', '*', [(Word([0]),'not')]),
    Rule('013', 'NN', 'WPS', [(Word([0]),'who')]),
    Rule('000', 'NN', 'NP', [(Pos([-1]),',')]),
    Rule('012', 'NN', 'NP', [(Word([-2, -1]),'Mrs.')]),
    Rule('013', 'NN', 'HV', [(Word([0]),'have')]),
    Rule('013', 'NN', 'VBD', [(Word([0]),'said')]),
    Rule('013', 'NN', 'BED', [(Word([0]),'were')])  )
```

Since so many of the generated rules involved singular words, I feel as though those cases would've been good candidates for my `regex_tagger` instead. By adding them as regular expressions into my `regex_tagger`, that would've likely cut down on the number of rules generated using the current word and allowed for the creation of more complex rules which might've used some of the 8 unused templates shown in **Listing 5**.

Surprisingly, even when the max number of generated rules was doubled to 50 rules, 69% of the rules were still created using the same current word template as before. There also were the same 8 unused templates, although `Template(Pos([1]))` and `Template(Pos([-1]))` were used much more often. These statistics can be seen in **Listing 7** below where it can also be seen that doubling the number of rules only increased the accuracy with the `test_data` by just over 3%.

Listing 7: Accuracy of and template statistics for the `brill_tagger` using the `regex_tagger` with a max of 50 rules.

```
Brill_tagger accuracy with dev_data: 0.6868662378595776
Brill_tagger accuracy with train_data: 0.6738824948760308
Brill with REGEX tagger accuracy with test_data: 0.6805529953917051

TEMPLATE STATISTICS (TRAIN)  7 templates, 50 rules)
TRAIN ( 145883 tokens) initial 74831 0.4870 final: 47575 0.6739
#ID | Score (train) |  #Rules     | Template
--------------------------------------------
013 | 18926   0.694 |  32   0.640 | Template(Word([0]))
009 |  2103   0.077 |   3   0.060 | Template(Pos([-3, -2, -1]))
011 |  2038   0.075 |   5   0.100 | Template(Pos([1]))
006 |  1503   0.055 |   1   0.020 | Template(Word([-1]))
000 |  1188   0.044 |   4   0.080 | Template(Pos([-1]))
003 |   924   0.034 |   3   0.060 | Template(Pos([-2, -1]))
012 |   574   0.021 |   2   0.040 | Template(Word([-2, -1]))

UNUSED TEMPLATES (8)
001 Template(Pos([-1]),Word([0]))
002 Template(Pos([-2]),Pos([-1]))
004 Template(Word([0]),Word([-1]))
005 Template(Pos([-2]),Word([0]))
007 Template(Pos([-1]),Word([-1]))
```

```
008 Template(Pos([-1]),Pos([0]))
010 Template(Pos([-2]),Pos([1]))
014 Template(Word([0]),Word([-1]),Pos([-1]))
```

A much more noticeable improvement can be seen when ditching the regular expressions altogether and using a unigram tagger as the baseline tagger. As shown below in **Listing 8**, the accuracy of this new transformation-based tagger increases to 89% when run against the `test_data`. Interestingly, the top two templates, each making up 28% of the 25 rules, were the `Template(Pos([1]))` and `Template(Pos([-1]))`. Both of these templates were previously shown to have increased in use when the original `brill_tagger` was run with 50 rules instead of 25.

Much more interestingly was the complete lack of use of the current word template: `Template(Word([0]))`. When considering that a current word template is essentially the only template upon which a unigram tagger operates, it becomes clear why this template was not used. Unigram taggers only look at the current word and therefore would have already learned all of the rules concerning singular words. With a baseline consisting of singular words and their tags, it's no wonder that this new `brill_unigram` tagger did not make use of that template.

Listing 8: Accuracy of and template statistics for the `brill_unigram` tagger using the `unigram_tagger`.

```
Brill with UNIGRAM tagger accuracy with dev_data: 0.8656816463910322
Brill with UNIGRAM tagger accuracy with train_data: 0.9456276605224735
Brill with UNIGRAM tagger accuracy with test_data: 0.890568356374808

TEMPLATE STATISTICS (TRAIN)  5 templates, 25 rules)
TRAIN ( 145883 tokens) initial 10288 0.9295 final:  7932 0.9456
#ID | Score (train) |  #Rules     | Template
--------------------------------------------
011 |   990   0.420 |   7   0.280 | Template(Pos([1]))
000 |   600   0.255 |   7   0.280 | Template(Pos([-1]))
001 |   345   0.146 |   5   0.200 | Template(Pos([-1]),Word([0]))
003 |   234   0.099 |   3   0.120 | Template(Pos([-2, -1]))
009 |   187   0.079 |   3   0.120 | Template(Pos([-3, -2, -1]))

UNUSED TEMPLATES (10)
002 Template(Pos([-2]),Pos([-1]))
004 Template(Word([0]),Word([-1]))
005 Template(Pos([-2]),Word([0]))
006 Template(Word([-1]))
007 Template(Pos([-1]),Word([-1]))
008 Template(Pos([-1]),Pos([0]))
010 Template(Pos([-2]),Pos([1]))
012 Template(Word([-2, -1]))
013 Template(Word([0]))
014 Template(Word([0]),Word([-1]),Pos([-1]))
```

The rules generated by this new and improved tagger seem much more rooted in linguistic concepts, as seen in **Listing 9** below. Most of the rules use previous part-of-speech tags or the next part-of-speech tag to more intelligently apply a tag to the current word. It is no wonder that the accuracy is so much higher than with a more brute force baseline tagger such as the `regex_tagger`.

Listing 9: Descriptions of the 25 rules learned by the `brill_unigram` tagger using the `unigram_tagger`.

```
(     Rule('011', 'TO', 'IN', [(Pos([1]),'AT')]),
      Rule('000', 'NN', 'VB', [(Pos([-1]),'TO')]),
      Rule('011', 'TO', 'IN', [(Pos([1]),'NP')]),
      Rule('001', 'IN', 'IN-TL', [(Pos([-1]),'NN-TL'), (Word([0]),'of')]),
      Rule('000', 'NN', 'VB', [(Pos([-1]),'MD')]),
      Rule('003', 'VB', 'NN', [(Pos([-2, -1]),'AT')]),
      Rule('009', 'VBD', 'VBN', [(Pos([-3, -2, -1]),'HVZ')]),
      Rule('011', 'NP', 'NP-TL', [(Pos([1]),'NN-TL')]),
      Rule('000', 'VBN', 'VBD', [(Pos([-1]),'NP')]),
      Rule('001', 'PPS', 'PPO', [(Pos([-1]),'VB'), (Word([0]),'it')]),
      Rule('000', 'VBN', 'VBD', [(Pos([-1]),'PPS')]),
      Rule('011', 'TO', 'IN', [(Pos([1]),'CD')]),
      Rule('011', 'TO', 'IN', [(Pos([1]),'NNS')]),
      Rule('003', 'VBD', 'VBN', [(Pos([-2, -1]),'BEDZ')]),
      Rule('001', 'CS', 'DT', [(Pos([-1]),'IN'), (Word([0]),'that')]),
      Rule('003', 'VBD', 'VBN', [(Pos([-2, -1]),'BE')]),
      Rule('011', 'TO', 'IN', [(Pos([1]),'PP$')]),
      Rule('000', 'NP', 'NP-TL', [(Pos([-1]),'JJ-TL')]),
      Rule('009', 'VBD', 'VBN', [(Pos([-3, -2, -1]),'HVD')]),
      Rule('000', 'VBN', 'VBD', [(Pos([-1]),'PPSS')]),
      Rule('000', 'NN', 'VB', [(Pos([-1]),'PPSS')]),
      Rule('001', 'PPS', 'PPO', [(Pos([-1]),'IN'), (Word([0]),'it')]),
      Rule('009', 'VBD', 'VBN', [(Pos([-3, -2, -1]),'HV')]),
      Rule('011', 'TO', 'IN', [(Pos([1]),'PPO')]),
      Rule('001', 'RB', 'AP', [(Pos([-1]),'AT'), (Word([0]),'only')])    )
```