

```

1  # This is a sample Python script.
2
3  # Press Shift+F10 to execute it or replace it with your code.
4  # Press Double Shift to search everywhere for classes, files, tool windows, actions,
   and settings.
5
6  import re
7  import os
8  import argparse
9  import numpy as np
10 import pandas as pd
11 from tqdm import tqdm
12
13 POS Tagger
14
15 # utils functions
16 # NN filling rules
17 def u_1(word): # assign the same tag in case of only special character
18     regex = re.compile("[^A-Za-z0-9]")
19     return re.findall(regex, word)[0] if (regex.search(word) != None) else "NN"
20
21 def u_2(word): # assign the CD tag in case of unknown number
22     return "CD" if (re.match('^\d+(\.\d+)*$', word) != None) else "NN"
23
24
25 def u_3(word): # assign the NNP tag of hyphenated digits
26     regex = re.compile(r'\d+(?:-\d+)+')
27     return "NNP" if (regex.search(word) != None) else "NN"
28
29
30 def u_4(word): # assign the NNP tag of hyphenated words
31     regex = re.compile(r'[a-zA-Z]+(?:-[a-zA-Z]+)+')
32     return "JJ" if (regex.search(word) != None) else "NN"
33
34
35 def u_5(word): # assign the JJ tag to the words containing year
36     string = 'year'
37     return "JJ" if string in word and (len(string) < len(word)) else "NN"
38
39
40 # error removing rules
41 def E_1(word, tag): # assign NN tag to selling
42     return "NN" if word == "selling" and tag == "VBG" else tag
43
44
45 def E_2(word, tag): # assign NNS tag to calls
46     return "NNS" if word == "calls" and tag == "VBZ" else tag
47
48
49 def E_3(word, tag): # assign VB tag to fall
50     return "VB" if word == "fall" and tag == "NN" else tag
51
52
53 # combine the rule 4 and 5. assign the NNP tag to word ending with s. assign the NNPS
   tag if word belong to nationality
54 def E_4(word, tag):
55     if word[-1] == "s" and tag == "NNPS":
56         countries = ['Americans', 'Soviets', 'Olympics', 'Workers', 'Yankees', 'Greeks',
57                     'Germans', 'Moslems', 'Europeans', 'Jews', 'Republicans',
58                     'Democrats', 'Representatives', 'Treasurys']
59         return tag if word in countries else "NNP"
60     return tag
61 #####
62 #####
63
64 # read text file

```

```

65 def read_file(file_name):
66     print("Start Reading File, {}".format(file_name.rsplit('/', 1)[-1]))
67
68     # open file
69     with open(training_file) as f:
70         words, tags = [], []
71         # iterate the file line by line
72         for line in tqdm(f.readlines()):
73             # split the line by last / to separate the word and tag
74             word, tag = line.strip().rsplit('/', 1)
75             words.append(word) # append in words list
76             tags.append(tag) # append in tags list
77
78     print("Prepare Dataset for File {}".format(file_name.rsplit('/', 1)[-1]))
79     df = pd.DataFrame([words, tags], index=['Words', 'Tags']).T # prepare dataset
80     print("Successfully Read and Prepare File, {} \U0001f600
81         \n\n".format(file_name.rsplit('/', 1)[-1]))
82     return df
83
84 def train_tagger(dataframe):
85     print("Start Training of POS Tagger....")
86
87     tagger_words = []
88     tagger_tags = []
89     distinct_words = dataframe['Words'].unique()
90     for word in tqdm(distinct_words):
91         temp_df = dataframe[dataframe['Words'] == word] # get records of specific word
92         max_prob_tag = temp_df['Tags'].value_counts().index[0] # find the tag with
93             maximum probability
94         tagger_words.append(word) # append word
95         tagger_tags.append(max_prob_tag) # append tag
96
97     print("Saving the Probabilities of Tagger..")
98     tagger_df = pd.DataFrame([tagger_words, tagger_tags], index=['Words', 'Tags']).T #
99     prepare dataframe
100     tagger_df.to_csv("tagger_df.csv", index=False) # write dataframe
101     print("Successfully Train the POS Tagger! \U0001f600 \n\n")
102     return tagger_df
103
104 def prediction_0(testing_file, tagger_df, output_file):
105     print("Start POS Tagging of Test Words....")
106     # open file
107     with open(testing_file) as f:
108         words, tags = [], []
109         # remove previously existing file
110         os.remove(output_file) if os.path.exists(output_file) else None
111         pred_files = open(output_file, "w+") # create new file
112
113         # iterate the file line by line
114         for line in tqdm(f.readlines()):
115             word = line.strip() # remove extra white spaces from both side of the word
116             # assign tag according to given criteria
117             tag = tagger_df[tagger_df['Words'] == word]['Tags'].values[0] if (
118                 word in tagger_df['Words'].values) else 'NN'
119
120             words.append(word) # append words
121             tags.append(tag) # append tags
122             pred_files.write(word + "/" + tag + "\n") # write words and their tags
123     print("Successfully Tagged POS Tags to Test Words! \U0001f600 \n\n")
124
125 # prediction module using updated rules
126 def prediction_1(testing_file, tagger_df, output_file):
127     print("Start POS Tagging of Test Words....")
128     # open file

```

```

129 with open(testing_file) as f:
130     words, tags = [], []
131     # remove previously existing file
132     os.remove(output_file) if os.path.exists(output_file) else None
133     pred_files = open(output_file, "w+") # create new file
134
135     # iterate the file line by line
136     for line in tqdm(f.readlines()):
137         word = line.strip() # remove extra white spaces from both side of the word
138         # assign tag according updated criteria of NN
139         if word in tagger_df['Words'].values:
140             tag = tagger_df[tagger_df['Words'] == word]['Tags'].values[0]
141         else:
142             # rule of handling NN
143             tag = u_1(word)
144             tag = u_2(word) if tag == "NN" else tag
145             tag = u_3(word) if tag == "NN" else tag
146             tag = u_4(word) if tag == "NN" else tag
147             tag = u_5(word) if tag == "NN" else tag
148
149         # remove error by manual rules
150         tag = E_1(word, tag)
151         tag = E_2(word, tag)
152         tag = E_3(word, tag)
153         tag = E_4(word, tag)
154
155         words.append(word) # append words
156         tags.append(tag) # append tags
157         pred_files.write(word + "/" + tag + "\n") # write words and their tags
158     print("Successfully Tagged POS Tags to Test Words! \U0001f600 \n\n")
159
160
161 # Press the green button in the gutter to run the script.
162 if __name__ == '__main__':
163     parser = argparse.ArgumentParser()
164     parser.add_argument('--mode', required=True, default=0)
165     parser.add_argument('--training_file', required=True, default="")
166     parser.add_argument('--testing_file', required=True, default="")
167     parser.add_argument('--output_file', required=True, default="")
168     args = parser.parse_args()
169
170     mode = args.mode
171     training_file = args.training_file
172     testing_file = args.testing_file
173     output_file = args.output_file
174
175     train_df = read_file(training_file)
176     tagger_df = train_tagger(train_df)
177
178     if mode == 0:
179         prediction_0(testing_file, tagger_df, output_file)
180     if mode == 1:
181         prediction_1(testing_file, tagger_df, output_file)
182
183

```

```

1  import re
2  import os
3  import argparse
4  import numpy as np
5  import pandas as pd
6  from tqdm import tqdm
7
8  Tagger Eval
9
10 def read_tags(file_name):
11     # open file
12     with open(file_name) as f:
13         tags = []
14         words = []
15         # ite rate the file line by line
16         for line in f.readlines():
17             # split the line by last / to separate the word and tag
18             word, tag = line.strip().rsplit('/', 1)
19             tags.append(tag) # append in tags list
20             words.append(word)
21
22     df = pd.DataFrame([words, tags], index=['Words', 'Tags']).T # prepare dataset
23     ff = str(file_name.rsplit('/', 1)[-1].rsplit('.', 1)[0]) + ".csv"
24     df.to_csv(ff, index=False)
25
26     return tags
27
28 def evaluate_tags(output_file, test_tags, pred_tags):
29     # compute accuracy score segment
30     accuracy = []
31     for i in range(len(test_tags)):
32         accuracy.append(1 if test_tags[i] == pred_tags[i] else accuracy.append(0)
33
34     acc_score = np.mean(accuracy)
35     print("Accuracy Score: {}".format(acc_score))
36
37     # calculate confusion metrix block
38     tags_name = sorted(set(test_tags) | set(pred_tags))
39     c = len(tags_name) # Number of classes
40     confusion_metrix_ = np.zeros((c, c))
41
42     for i in range(len(test_tags)):
43         confusion_metrix_[tags_name.index(pred_tags[i])][tags_name.index(test_tags[i])]
44             += 1
45
46     # write confusion metrix in a file
47     os.remove(output_file) if os.path.exists(output_file) else None # remove
48     previouslt existing file
49     eval_file = open(output_file, "w+") # create new file
50
51     for i in tqdm(range(0, len(tags_name))):
52         preds = confusion_metrix_[i]
53         true_false_positives = np.where(preds != 0)[0]
54         for index in true_false_positives:
55             eval_file.write(tags_name[i] + " " + tags_name[index] + " : " +
56                 str(int(preds[index])) + " \n")
57
58     print("Confusion Metrix Results are Write Successfuly in {} \U0001f600
59     \n\n".format(output_file.rsplit('/', 1)[-1]))
60
61 # Press the green button in the gutter to run the script.
62 if __name__ == '__main__':
63     parser = argparse.ArgumentParser()
64     parser.add_argument('--key_file', required=True, default="")
65     parser.add_argument('--answer_file', required=True, default="")
66     parser.add_argument('--output_file', required=True, default="")

```

```
64     args = parser.parse_args()
65
66     mode = args.mode
67     test_tag_file = args.key_file
68     pred_tag_file = args.answer_file
69     output_file = args.output_file
70
71     test_tags = read_tags(test_tag_file)
72     pred_tags = read_tags(pred_tag_file)
73     evaluate_tags(output_file, test_tags, pred_tags)
```

POS-test-0-eval

1 # # : 5
2 \$ \$: 375
3 ' ' ' ' : 528
4 ((: 76
5)) : 76
6 , , : 3070
7 . . : 2363
8 : : : 336
9 CC CC : 1361
10 CC IN : 2
11 CC NN : 3
12 CC RB : 2
13 CD CD : 1848
14 CD JJ : 3
15 CD LS : 4
16 CD NN : 6
17 CD NNP : 7
18 CD NNS : 9
19 CD PRP : 6
20 DT CC : 11
21 DT DT : 4747
22 DT IN : 4
23 DT NNP : 5
24 DT PDT : 11
25 DT RB : 14
26 DT UH : 1
27 EX EX : 57
28 EX RB : 6
29 FW FW : 3
30 FW NN : 1
31 FW NNP : 2
32 IN DT : 68
33 IN IN : 5830
34 IN JJ : 3
35 IN NN : 2
36 IN NNP : 1
37 IN RB : 180
38 IN RP : 94
39 IN VB : 4
40 IN VBP : 1
41 IN WDT : 138
42 JJ DT : 2
43 JJ IN : 7
44 JJ JJ : 3098
45 JJ JJR : 1
46 JJ NN : 124
47 JJ NNP : 63
48 JJ NNPS : 5
49 JJ NNS : 1
50 JJ PDT : 4
51 JJ RB : 70
52 JJ RBR : 2
53 JJ VB : 26
54 JJ VBD : 3
55 JJ VBG : 4
56 JJ VBN : 8
57 JJ VBP : 5
58 JJR JJR : 188
59 JJR NNP : 1
60 JJR RBR : 77
61 JJR VB : 1
62 JJS JJ : 1
63 JJS JJS : 122
64 JJS RB : 2
65 JJS RBS : 30
66 MD MD : 584
67 MD NN : 3
68 NN CD : 87
69 NN DT : 2

70 NN FW : 14
71 NN JJ : 320
72 NN JJR : 3
73 NN JJS : 3
74 NN NN : 7165
75 NN NNP : 511
76 NN NNPS : 6
77 NN NNS : 103
78 NN PDT : 3
79 NN RB : 44
80 NN VB : 261
81 NN VBD : 33
82 NN VBG : 73
83 NN VBN : 19
84 NN VBP : 61
85 NN VBZ : 13
86 NNP FW : 2
87 NNP JJ : 49
88 NNP JJR : 1
89 NNP JJS : 1
90 NNP NN : 49
91 NNP NNP : 5184
92 NNP NNPS : 4
93 NNP NNS : 7
94 NNP RB : 2
95 NNP VB : 3
96 NNPS NNP : 134
97 NNPS NNPS : 22
98 NNPS NNS : 6
99 NNS JJ : 1
100 NNS NN : 6
101 NNS NNP : 22
102 NNS NNPS : 5
103 NNS NNS : 3314
104 NNS VBZ : 64
105 POS '' : 5
106 POS POS : 551
107 POS VBZ : 77
108 PRP PRP : 1042
109 PRP\$ PRP : 8
110 PRP\$ PRP\$: 510
111 RB CC : 1
112 RB FW : 1
113 RB IN : 23
114 RB JJ : 31
115 RB NN : 1
116 RB NNP : 2
117 RB RB : 1724
118 RB RBR : 8
119 RB RP : 83
120 RB VB : 2
121 RB WRB : 1
122 RBR JJR : 5
123 RBR RBR : 25
124 RBS JJS : 2
125 SYM SYM : 1
126 TO TO : 1245
127 UH UH : 7
128 VB JJ : 15
129 VB NN : 114
130 VB NNP : 2
131 VB RB : 6
132 VB VB : 1122
133 VB VBD : 5
134 VB VBN : 21
135 VB VBP : 133
136 VBD JJ : 38
137 VBD NN : 5
138 VBD VB : 4

139 VBD VBD : 1573
140 VBD VBN : 220
141 VBD VBP : 1
142 VBG JJ : 34
143 VBG NN : 86
144 VBG NNP : 1
145 VBG VBG : 741
146 VBN JJ : 117
147 VBN NN : 3
148 VBN VB : 5
149 VBN VBD : 223
150 VBN VBN : 832
151 VBN VBP : 3
152 VBP JJ : 2
153 VBP NN : 11
154 VBP NNP : 1
155 VBP VB : 161
156 VBP VBP : 598
157 VBZ NN : 1
158 VBZ NNP : 3
159 VBZ NNS : 73
160 VBZ VBZ : 1082
161 WDT WDT : 141
162 WP WDT : 1
163 WP WP : 111
164 WP\$ WP\$: 21
165 WRB WRB : 132
166 `` `` : 535
167

POS-test-1-eval

1 # # : 5
2 \$ \$: 375
3 % JJ : 1
4 & NNP : 1
5 ' NNP : 2
6 '' '' : 528
7 ((: 76
8)) : 76
9 , , : 3070
10 , CD : 15
11 - CD : 5
12 - JJ : 131
13 - JJR : 1
14 - NN : 43
15 - NNP : 22
16 - NNS : 8
17 - RB : 3
18 - VB : 2
19 - VBN : 1
20 . . : 2363
21 . CD : 56
22 . JJ : 3
23 . NNP : 4
24 : : : 336
25 : CD : 2
26 CC CC : 1361
27 CC IN : 2
28 CC NN : 3
29 CC RB : 2
30 CD CD : 1857
31 CD JJ : 3
32 CD LS : 4
33 CD NN : 6
34 CD NNP : 8
35 CD NNS : 9
36 CD PRP : 6
37 DT CC : 11
38 DT DT : 4747
39 DT IN : 4
40 DT NNP : 5
41 DT PDT : 11
42 DT RB : 14
43 DT UH : 1
44 EX EX : 57
45 EX RB : 6
46 FW FW : 3
47 FW NN : 1
48 FW NNP : 2
49 IN DT : 68
50 IN IN : 5830
51 IN JJ : 3
52 IN NN : 2
53 IN NNP : 1
54 IN RB : 180
55 IN RP : 94
56 IN VB : 4
57 IN VBP : 1
58 IN WDT : 138
59 JJ DT : 2
60 JJ IN : 7
61 JJ JJ : 3099
62 JJ JJR : 1
63 JJ NN : 124
64 JJ NNP : 63
65 JJ NNPS : 5
66 JJ NNS : 1
67 JJ PDT : 4
68 JJ RB : 70
69 JJ RBR : 2

70 JJ VB : 26
71 JJ VBD : 3
72 JJ VBG : 4
73 JJ VBN : 8
74 JJ VBP : 5
75 JJR JJR : 188
76 JJR NNP : 1
77 JJR RBR : 77
78 JJR VB : 1
79 JJS JJ : 1
80 JJS JJS : 122
81 JJS RB : 2
82 JJS RBS : 30
83 MD MD : 584
84 MD NN : 3
85 NN DT : 2
86 NN FW : 14
87 NN JJ : 183
88 NN JJR : 2
89 NN JJS : 3
90 NN NN : 7130
91 NN NNP : 480
92 NN NNPS : 6
93 NN NNS : 95
94 NN PDT : 3
95 NN RB : 41
96 NN VB : 252
97 NN VBD : 33
98 NN VBG : 91
99 NN VBN : 18
100 NN VBP : 58
101 NN VBZ : 13
102 NNP FW : 2
103 NNP JJ : 49
104 NNP JJR : 1
105 NNP JJS : 1
106 NNP NN : 49
107 NNP NNP : 5314
108 NNP NNPS : 4
109 NNP NNS : 12
110 NNP RB : 2
111 NNP VB : 3
112 NNPS NNP : 4
113 NNPS NNPS : 22
114 NNPS NNS : 1
115 NNS JJ : 1
116 NNS NN : 6
117 NNS NNP : 22
118 NNS NNPS : 5
119 NNS NNS : 3332
120 NNS VBZ : 67
121 POS ' ' : 5
122 POS POS : 551
123 POS VBZ : 77
124 PRP PRP : 1042
125 PRP\$ PRP : 8
126 PRP\$ PRP\$: 510
127 RB CC : 1
128 RB FW : 1
129 RB IN : 23
130 RB JJ : 31
131 RB NN : 1
132 RB NNP : 2
133 RB RB : 1724
134 RB RBR : 8
135 RB RP : 83
136 RB VB : 2
137 RB WRB : 1
138 RBR JJR : 5

139 RBR RBR : 25
140 RBS JJS : 2
141 SYM SYM : 1
142 TO TO : 1245
143 UH UH : 7
144 VB JJ : 15
145 VB NN : 119
146 VB NNP : 2
147 VB RB : 6
148 VB VB : 1129
149 VB VBD : 5
150 VB VBN : 21
151 VB VBP : 136
152 VBD JJ : 38
153 VBD NN : 5
154 VBD VB : 4
155 VBD VBD : 1573
156 VBD VBN : 220
157 VBD VBP : 1
158 VBG JJ : 34
159 VBG NN : 73
160 VBG NNP : 1
161 VBG VBG : 723
162 VBN JJ : 117
163 VBN NN : 3
164 VBN VB : 5
165 VBN VBD : 223
166 VBN VBN : 832
167 VBN VBP : 3
168 VBP JJ : 2
169 VBP NN : 11
170 VBP NNP : 1
171 VBP VB : 161
172 VBP VBP : 598
173 VBZ NN : 1
174 VBZ NNP : 3
175 VBZ NNS : 55
176 VBZ VBZ : 1079
177 WDT WDT : 141
178 WP WDT : 1
179 WP WP : 111
180 WP\$ WP\$: 21
181 WRB WRB : 132
182 \ JJ : 1
183 \ NNP : 1
184 `` `` : 535
185