```python
# This is a sample Python script.

# Press Shift+F10 to execute it or replace it with your code.
# Press Double Shift to search everywhere for classes, files, tool windows, actions, and settings.

import re
import os
import argparse
import numpy as np
import pandas as pd
from tqdm import tqdm
```

POS Tagger

```python
# utils functions
# NN filling rules
def u_1(word):  # assign the same tag in case of only special character
    regex = re.compile("[^A-Za-z0-9]")
    return re.findall(regex, word)[0] if(regex.search(word) != None) else "NN"


def u_2(word):  # assign the CD tag in case of unknown number
    return "CD" if (re.match('^\d+(\.\d+)*$', word) != None) else "NN"


def u_3(word):  # assign the NNP tag of hyphenated digits
    regex = re.compile(r'\d+(?:-\d+)+')
    return "NNP" if (regex.search(word) != None) else "NN"


def u_4(word):  # assign the NNP tag of hyphenated words
    regex = re.compile(r'[a-zA-Z]+(?:-[a-zA-Z]+)+')
    return "JJ" if (regex.search(word) != None) else "NN"


def u_5(word):  # assign the JJ tag to the words containing year
    string = 'year'
    return "JJ" if string in word and (len(string) < len(word)) else "NN"


# error removing rules
def E_1(word, tag):  # assign NN tag to selling
    return "NN" if word == "selling" and tag == "VBG" else tag


def E_2(word, tag):  # assign NNS tag to calls
    return "NNS" if word == "calls" and tag == "VBZ" else tag


def E_3(word, tag):  # assign VB tag to fall
    return "VB" if word == "fall" and tag == "NN" else tag


# combine the rule 4 and 5. assign the NNP tag to word ending with s. assign the NNPS tag if word belong to nationality
def E_4(word, tag):
    if word[-1] == "s" and tag == "NNPS":
        countries = ['Americans', 'Soviets', 'Olympics', 'Workers', 'Yankees', 'Greeks',
                     'Germans', 'Moslems', 'Europeans', 'Jews', 'Republicans',
                     'Democrats', 'Representatives', 'Treasurys']
        return tag if word in countries else "NNP"
    return tag
################################################################################################################################


# 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
        \n\n".format(file_name.rsplit('/', 1)[-1]))
81      return df
82
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
            maximum probability
93          tagger_words.append(word)  # append word
94          tagger_tags.append(max_prob_tag)  # append tag
95
96      print("Saving the Probabilities of Tagger..")
97      tagger_df = pd.DataFrame([tagger_words, tagger_tags], index=['Words', 'Tags']).T  #
        prepare dataframe
98      tagger_df.to_csv("tagger_df.csv", index=False)  # write dataframe
99      print("Successfully Train the POS Tagger! \U0001f600 \n\n")
100     return tagger_df
101
102
103 def prediction_0(testing_file, tagger_df, output_file):
104     print("Start POS Tagging of Test Words....")
105     # open file
106     with open(testing_file) as f:
107         words, tags = [], []
108         # remove previously existing file
109         os.remove(output_file) if os.path.exists(output_file) else None
110         pred_files = open(output_file, "w+")  # create new file
111
112         # iterate the file line by line
113         for line in tqdm(f.readlines()):
114             word = line.strip()  # remove extra white spaces from both side of the word
115             # assign tag according to given criteria
116             tag = tagger_df[tagger_df['Words'] == word]['Tags'].values[0] if (
117                         word in tagger_df['Words'].values) else 'NN'
118
119             words.append(word)  # append words
120             tags.append(tag)  # append tags
121             pred_files.write(word + "/" + tag + "\n")  # write words and their tags
122     print("Successfully Tagged POS Tags to Test Words! \U0001f600 \n\n")
123
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
```

```
    with open(testing_file) as f:
        words, tags = [], []
        # remove previously existing file
        os.remove(output_file) if os.path.exists(output_file) else None
        pred_files = open(output_file, "w+")  # create new file

        # iterate the file line by line
        for line in tqdm(f.readlines()):
            word = line.strip()  # remove extra white spaces from both side of the word
            # assign tag according updated criteria of NN
            if word in tagger_df['Words'].values:
                tag = tagger_df[tagger_df['Words'] == word]['Tags'].values[0]
            else:
                # rule of handling NN
                tag = u_1(word)
                tag = u_2(word) if tag == "NN" else tag
                tag = u_3(word) if tag == "NN" else tag
                tag = u_4(word) if tag == "NN" else tag
                tag = u_5(word) if tag == "NN" else tag

            # remove error by manual rules
            tag = E_1(word, tag)
            tag = E_2(word, tag)
            tag = E_3(word, tag)
            tag = E_4(word, tag)

            words.append(word)  # append words
            tags.append(tag)  # append tags
            pred_files.write(word + "/" + tag + "\n")  # write words and their tags
    print("Successfuly Tagged POS Tags to Test Words! \U0001f600 \n\n")


# Press the green button in the gutter to run the script.
if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument('--mode', required=True, default=0)
    parser.add_argument('--training_file', required=True, default="")
    parser.add_argument('--testing_file', required=True, default="")
    parser.add_argument('--output_file', required=True, default="")
    args = parser.parse_args()

    mode = args.mode
    training_file = args.training_file
    testing_file = args.testing_file
    output_file = args.output_file

    train_df = read_file(training_file)
    tagger_df = train_tagger(train_df)

    if mode == 0:
        prediction_0(testing_file, tagger_df, output_file)
    if mode == 1:
        prediction_1(testing_file, tagger_df, output_file)


```

```python
import re
import os
import argparse
import numpy as np
import pandas as pd
from tqdm import tqdm
```

Tagger Eval

```python
def read_tags(file_name):
    # open file
    with open(file_name) as f:
        tags = []
        words = []
        # ite rate the file line by line
        for line in f.readlines():
            # split the line by last / to separate the word and tag
            word, tag = line.strip().rsplit('/', 1)
            tags.append(tag)  # append in tags list
            words.append(word)

    df = pd.DataFrame([words, tags], index=['Words', 'Tags']).T  # prepare dataset
    ff = str(file_name.rsplit('/', 1)[-1].rsplit('.', 1)[0]) + ".csv"
    df.to_csv(ff, index=False)

    return tags


def evaluate_tags(output_file, test_tags, pred_tags):
    # compute accuracy score segment
    accuracy = []
    for i in range(len(test_tags)):
        accuracy.append(1) if test_tags[i] == pred_tags[i] else accuracy.append(0)

    acc_score = np.mean(accuracy)
    print("Accuracy Score: {}".format(acc_score))

    # calculate confusion metrix block
    tags_name = sorted(set(test_tags) | set (pred_tags))
    c = len(tags_name)  # Number of classes
    confusion_metrix_ = np.zeros((c, c))

    for i in range(len(test_tags)):
        confusion_metrix_[tags_name.index(pred_tags[i])][tags_name.index(test_tags[i])]
        += 1

    # write confusion metrix in a file
    os.remove(output_file) if os.path.exists(output_file) else None  # remove
    previouslt existing file
    eval_file = open(output_file, "w+")  # create new file

    for i in tqdm(range(0, len(tags_name))):
        preds = confusion_metrix_[i]
        true_false_positives = np.where(preds != 0)[0]
        for index in true_false_positives:
            eval_file.write(tags_name[i] + " " + tags_name[index] + " : " +
            str(int(preds[index])) + " \n")

    print("Confusion Metrix Results are Write Successfuly in {} \U0001f600
    \n\n".format(output_file.rsplit('/', 1)[-1]))


# Press the green button in the gutter to run the script.
if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument('--key_file', required=True, default="")
    parser.add_argument('--answer_file', required=True, default="")
    parser.add_argument('--output_file', required=True, default="")
```

```
    args = parser.parse_args()

    mode = args.mode
    test_tag_file = args.key_file
    pred_tag_file = args.answer_file
    output_file = args.output_file

    test_tags = read_tags(test_tag_file)
    pred_tags = read_tags(pred_tag_file)
    evaluate_tags(output_file, test_tags, pred_tags)
```

POS-test-0-eval

```
# # : 5
$ $ : 375
'' '' : 528
( ( : 76
) ) : 76
, , : 3070
. . : 2363
: : : 336
CC CC : 1361
CC IN : 2
CC NN : 3
CC RB : 2
CD CD : 1848
CD JJ : 3
CD LS : 4
CD NN : 6
CD NNP : 7
CD NNS : 9
CD PRP : 6
DT CC : 11
DT DT : 4747
DT IN : 4
DT NNP : 5
DT PDT : 11
DT RB : 14
DT UH : 1
EX EX : 57
EX RB : 6
FW FW : 3
FW NN : 1
FW NNP : 2
IN DT : 68
IN IN : 5830
IN JJ : 3
IN NN : 2
IN NNP : 1
IN RB : 180
IN RP : 94
IN VB : 4
IN VBP : 1
IN WDT : 138
JJ DT : 2
JJ IN : 7
JJ JJ : 3098
JJ JJR : 1
JJ NN : 124
JJ NNP : 63
JJ NNPS : 5
JJ NNS : 1
JJ PDT : 4
JJ RB : 70
JJ RBR : 2
JJ VB : 26
JJ VBD : 3
JJ VBG : 4
JJ VBN : 8
JJ VBP : 5
JJR JJR : 188
JJR NNP : 1
JJR RBR : 77
JJR VB : 1
JJS JJ : 1
JJS JJS : 122
JJS RB : 2
JJS RBS : 30
MD MD : 584
MD NN : 3
NN CD : 87
NN DT : 2
```

```
  NN FW : 14
  NN JJ : 320
  NN JJR : 3
  NN JJS : 3
  NN NN : 7165
  NN NNP : 511
  NN NNPS : 6
  NN NNS : 103
  NN PDT : 3
  NN RB : 44
  NN VB : 261
  NN VBD : 33
  NN VBG : 73
  NN VBN : 19
  NN VBP : 61
  NN VBZ : 13
  NNP FW : 2
  NNP JJ : 49
  NNP JJR : 1
  NNP JJS : 1
  NNP NN : 49
  NNP NNP : 5184
  NNP NNPS : 4
  NNP NNS : 7
  NNP RB : 2
  NNP VB : 3
  NNPS NNP : 134
  NNPS NNPS : 22
  NNPS NNS : 6
  NNS JJ : 1
  NNS NN : 6
  NNS NNP : 22
  NNS NNPS : 5
  NNS NNS : 3314
  NNS VBZ : 64
  POS '' : 5
  POS POS : 551
  POS VBZ : 77
  PRP PRP : 1042
  PRP$ PRP : 8
  PRP$ PRP$ : 510
  RB CC : 1
  RB FW : 1
  RB IN : 23
  RB JJ : 31
  RB NN : 1
  RB NNP : 2
  RB RB : 1724
  RB RBR : 8
  RB RP : 83
  RB VB : 2
  RB WRB : 1
  RBR JJR : 5
  RBR RBR : 25
  RBS JJS : 2
  SYM SYM : 1
  TO TO : 1245
  UH UH : 7
  VB JJ : 15
  VB NN : 114
  VB NNP : 2
  VB RB : 6
  VB VB : 1122
  VB VBD : 5
  VB VBN : 21
  VB VBP : 133
  VBD JJ : 38
  VBD NN : 5
  VBD VB : 4
```

```
  VBD VBD : 1573
  VBD VBN : 220
  VBD VBP : 1
  VBG JJ : 34
  VBG NN : 86
  VBG NNP : 1
  VBG VBG : 741
  VBN JJ : 117
  VBN NN : 3
  VBN VB : 5
  VBN VBD : 223
  VBN VBN : 832
  VBN VBP : 3
  VBP JJ : 2
  VBP NN : 11
  VBP NNP : 1
  VBP VB : 161
  VBP VBP : 598
  VBZ NN : 1
  VBZ NNP : 3
  VBZ NNS : 73
  VBZ VBZ : 1082
  WDT WDT : 141
  WP WDT : 1
  WP WP : 111
  WP$ WP$ : 21
  WRB WRB : 132
  `` `` : 535

```

POS-test-1-eval

```
# # : 5
$ $ : 375
% JJ : 1
& NNP : 1
' NNP : 2
'' '' : 528
( ( : 76
) ) : 76
, , : 3070
, CD : 15
- CD : 5
- JJ : 131
- JJR : 1
- NN : 43
- NNP : 22
- NNS : 8
- RB : 3
- VB : 2
- VBN : 1
. . : 2363
. CD : 56
. JJ : 3
. NNP : 4
: : : 336
: CD : 2
CC CC : 1361
CC IN : 2
CC NN : 3
CC RB : 2
CD CD : 1857
CD JJ : 3
CD LS : 4
CD NN : 6
CD NNP : 8
CD NNS : 9
CD PRP : 6
DT CC : 11
DT DT : 4747
DT IN : 4
DT NNP : 5
DT PDT : 11
DT RB : 14
DT UH : 1
EX EX : 57
EX RB : 6
FW FW : 3
FW NN : 1
FW NNP : 2
IN DT : 68
IN IN : 5830
IN JJ : 3
IN NN : 2
IN NNP : 1
IN RB : 180
IN RP : 94
IN VB : 4
IN VBP : 1
IN WDT : 138
JJ DT : 2
JJ IN : 7
JJ JJ : 3099
JJ JJR : 1
JJ NN : 124
JJ NNP : 63
JJ NNPS : 5
JJ NNS : 1
JJ PDT : 4
JJ RB : 70
JJ RBR : 2
```

```
  JJ VB : 26
  JJ VBD : 3
  JJ VBG : 4
  JJ VBN : 8
  JJ VBP : 5
  JJR JJR : 188
  JJR NNP : 1
  JJR RBR : 77
  JJR VB : 1
  JJS JJ : 1
  JJS JJS : 122
  JJS RB : 2
  JJS RBS : 30
  MD MD : 584
  MD NN : 3
  NN DT : 2
  NN FW : 14
  NN JJ : 183
  NN JJR : 2
  NN JJS : 3
  NN NN : 7130
  NN NNP : 480
  NN NNPS : 6
  NN NNS : 95
  NN PDT : 3
  NN RB : 41
  NN VB : 252
  NN VBD : 33
  NN VBG : 91
  NN VBN : 18
  NN VBP : 58
  NN VBZ : 13
  NNP FW : 2
  NNP JJ : 49
  NNP JJR : 1
  NNP JJS : 1
  NNP NN : 49
  NNP NNP : 5314
  NNP NNPS : 4
  NNP NNS : 12
  NNP RB : 2
  NNP VB : 3
  NNPS NNP : 4
  NNPS NNPS : 22
  NNPS NNS : 1
  NNS JJ : 1
  NNS NN : 6
  NNS NNP : 22
  NNS NNPS : 5
  NNS NNS : 3332
  NNS VBZ : 67
  POS '' : 5
  POS POS : 551
  POS VBZ : 77
  PRP PRP : 1042
  PRP$ PRP : 8
  PRP$ PRP$ : 510
  RB CC : 1
  RB FW : 1
  RB IN : 23
  RB JJ : 31
  RB NN : 1
  RB NNP : 2
  RB RB : 1724
  RB RBR : 8
  RB RP : 83
  RB VB : 2
  RB WRB : 1
  RBR JJR : 5
```

```
  RBR RBR : 25
  RBS JJS : 2
  SYM SYM : 1
  TO TO : 1245
  UH UH : 7
  VB JJ : 15
  VB NN : 119
  VB NNP : 2
  VB RB : 6
  VB VB : 1129
  VB VBD : 5
  VB VBN : 21
  VB VBP : 136
  VBD JJ : 38
  VBD NN : 5
  VBD VB : 4
  VBD VBD : 1573
  VBD VBN : 220
  VBD VBP : 1
  VBG JJ : 34
  VBG NN : 73
  VBG NNP : 1
  VBG VBG : 723
  VBN JJ : 117
  VBN NN : 3
  VBN VB : 5
  VBN VBD : 223
  VBN VBN : 832
  VBN VBP : 3
  VBP JJ : 2
  VBP NN : 11
  VBP NNP : 1
  VBP VB : 161
  VBP VBP : 598
  VBZ NN : 1
  VBZ NNP : 3
  VBZ NNS : 55
  VBZ VBZ : 1079
  WDT WDT : 141
  WP WDT : 1
  WP WP : 111
  WP$ WP$ : 21
  WRB WRB : 132
  \ JJ : 1
  \ NNP : 1
  `` `` : 535

```