In [20]:
```
from __future__ import division #To avoid integer division
from operator import itemgetter
```

In [21]:
```
with open("words_pos.csv", "r") as myfile:
    tr_str = myfile.read()
tr_li = tr_str.split()
num_words_train = len(tr_li)
```

In [22]:
```
train_li_words = ['']
train_li_words*= num_words_train

train_li_tags = ['']
train_li_tags*= num_words_train
```

In [23]:
```
noun_reduced_list = ['NN','NNS','NNP','NNPS']
verb_reduced_list = ['VB','VBD','VBG','VBN','VBP','VBZ']
adjec_reduced_list = ['JJ', 'JJR', 'JJS']
adv_reduced_list = ['RB', 'RBR', 'RBS']
pronoun_reduced_list = ['PRP', 'PRP$', 'RP']
```

In [24]:
```
for i in range(num_words_train):
    temp_li = tr_li[i]
    train_li_words[i] = temp_li[0]
    if temp_li[1] in noun_reduced_list:
        train_li_tags[i] = 'N'
    elif temp_li[1] in verb_reduced_list:
        train_li_tags[i] = 'V'
    elif temp_li[1] in adjec_reduced_list:
        train_li_tags[i] = 'ADJ'
    elif temp_li[1] in adv_reduced_list:
        train_li_tags[i] = 'ADV'
    elif temp_li[1] in pronoun_reduced_list:
        train_li_tags[i] = 'PRO'
    else:
        train_li_tags[i] = temp_li[1]
```

In [7]:
```
k = sorted(list(set(train_li_tags)))
print(k)
dict2_tag_follow_tag_ = {}
"""Nested dictionary to store the transition probabilities
each tag A is a key of the outer dictionary
the inner dictionary is the corresponding value
The inner dictionary's key is the tag B following A
and the corresponding value is the number of times B follows A
"""

dict2_word_tag = {}
"""Nested dictionary to store the emission probabilities.
Each word W is a key of the outer dictionary
The inner dictionary is the corresponding value
The inner dictionary's key is the tag A of the word W
and the corresponding value is the number of times A is a tag of W
"""

dict_word_tag_baseline = {}
#Dictionary with word as key and its most frequent tag as value
```

```
[',', '0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'w']
```

In [8]:
```
for i in range(num_words_train-1):
    outer_key = train_li_tags[i]
    inner_key = train_li_tags[i+1]
    dict2_tag_follow_tag_[outer_key]=dict2_tag_follow_tag_.get(outer_key,{})
    dict2_tag_follow_tag_[outer_key][inner_key] = dict2_tag_follow_tag_[outer_key].get(inne
    dict2_tag_follow_tag_[outer_key][inner_key]+=1

    outer_key = train_li_words[i]
    inner_key = train_li_tags[i]
    dict2_word_tag[outer_key]=dict2_word_tag.get(outer_key,{})
    dict2_word_tag[outer_key][inner_key] = dict2_word_tag[outer_key].get(inner_key,0)
    dict2_word_tag[outer_key][inner_key]+=1
```

In [9]:
```
# Check if 'NN' key exists in dict2_tag_follow_tag_
if 'NN' in dict2_tag_follow_tag_:
    print(dict2_tag_follow_tag_['NN'])
else:
    print("Key 'NN' not found in dict2_tag_follow_tag_")

# Check if 'Rudolph' key exists in dict2_word_tag
if 'Rudolph' in dict2_word_tag:
    print(dict2_word_tag['Rudolph'])
else:
    print("Key 'Rudolph' not found in dict2_word_tag")
```

```
Key 'NN' not found in dict2_tag_follow_tag_
Key 'Rudolph' not found in dict2_word_tag
```

In [10]:
```
"""Converting counts to probabilities in the two nested dictionaries
& also converting the nested dictionaries to outer dictionary with inner sorted lists
"""
for key in dict2_tag_follow_tag_:
    di = dict2_tag_follow_tag_[key]
    s = sum(di.values())
    for innkey in di:
        di[innkey] /= s
    di = di.items()
    di = sorted(di,key=lambda x: x[0])
    dict2_tag_follow_tag_[key] = di

for key in dict2_word_tag:
    di = dict2_word_tag[key]
    dict_word_tag_baseline[key] = max(di, key=di.get)
    s = sum(di.values())
    for innkey in di:
        di[innkey] /= s
    di = di.items()
    di = sorted(di,key=lambda x: x[0])
    dict2_word_tag[key] = di
```

In [11]:

```
###Testing Phase###
with open("output.txt", "r") as myfile:
    te_str = myfile.read()

te_li = te_str.split()
num_words_test = len(te_li)

test_li_words = ['']
test_li_words*= num_words_test

test_li_tags = ['']
test_li_tags*= num_words_test

output_li = ['']
output_li*= num_words_test

output_li_baseline = ['']
output_li_baseline*= num_words_test

num_errors = 0
num_errors_baseline = 0
```

In [12]:

```
for i in range(num_words_test):
    temp_li = te_li[i].split("/")
    test_li_words[i] = temp_li[0]
    if temp_li[1] in noun_reduced_list:
        test_li_tags[i] = 'N'
    elif temp_li[1] in verb_reduced_list:
        test_li_tags[i] = 'V'
    elif temp_li[1] in adjec_reduced_list:
        test_li_tags[i] = 'ADJ'
    elif temp_li[1] in adv_reduced_list:
        test_li_tags[i] = 'ADV'
    elif temp_li[1] in pronoun_reduced_list:
        test_li_tags[i] = 'PRO'
    else:
        test_li_tags[i] = temp_li[1]

    output_li_baseline[i] = dict_word_tag_baseline.get(temp_li[0], '')
    # If unknown word - tag = 'N'
    if output_li_baseline[i] == '':
        output_li_baseline[i] = 'N'

    if output_li_baseline[i] != test_li_tags[i]:
        num_errors_baseline += 1

    if i == 0:  # Accounting for the 1st word in the test document for the Viterbi
        di_transition_probs = dict2_tag_follow_tag_.get('.', [])  # Get value for key '.' o
    else:
        di_transition_probs = dict2_tag_follow_tag_.get(output_li[i - 1], [])

    di_emission_probs = dict2_word_tag.get(test_li_words[i], '')

    # If unknown word - tag = 'N'
    if di_emission_probs == '':
        output_li[i] = 'N'
    else:
        max_prod_prob = 0
        counter_trans = 0
        counter_emis = 0
        prod_prob = 0
        while counter_trans < len(di_transition_probs) and counter_emis < len(di_emission_p
            tag_tr = di_transition_probs[counter_trans][0]
            tag_em = di_emission_probs[counter_emis][0]
            if tag_tr < tag_em:
                counter_trans += 1
            elif tag_tr > tag_em:
                counter_emis += 1
            else:
                prod_prob = di_transition_probs[counter_trans][1] * di_emission_probs[count
                if prod_prob > max_prod_prob:
                    max_prod_prob = prod_prob
                    output_li[i] = tag_tr
                    print("i=", i, " and output=", output_li[i])
                counter_trans += 1
                counter_emis += 1

        if output_li[i] == '':  # In case there are no matching entries between the transit
            output_li[i] = max(di_emission_probs, key=itemgetter(1))[0]

    if output_li[i] != test_li_tags[i]:
        num_errors += 1
```

In [13]:
```
print("Fraction of errors (Baseline) :",(num_errors_baseline/num_words_test))
print("Fraction of errors (Viterbi):",(num_errors/num_words_test))

print("Tags suggested by Baseline Algorithm:", output_li_baseline)

print("Tags suggested by Viterbi Algorithm:", output_li)

print("Correct tags:",test_li_tags)
```

```
Fraction of errors (Baseline) : 0.7303252885624344
Fraction of errors (Viterbi): 0.7303252885624344
Tags suggested by Baseline Algorithm: ['N', 'N', 'N', 'N', 'N', 'N', 'w', 'N', 'N',
'N', 'N', 'N', 'N', 'N', 'w', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'w', 'N',
'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N',
'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'w', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N',
'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N',
'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'w', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N',
'N', 'N', 'N', 'N', 'N', 'w', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N',
'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'w',
'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N',
'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N',
'N', 'N', 'N', 'N', 'N', 'N', 'N', 'w', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N',
'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'w', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N',
'N', 'N', 'w', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'w',
'N', 'N', 'N', 'N', 'N', 'N', 'N', 'w', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N',
'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'w', 'N', 'N', 'N',
'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'w', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N',
```

In [ ]:

In [ ]:

In [ ]: