Chapter 6

In [3]:
```
import nltk
```

In [2]:
```
from nltk import *
```

In [ ]:

1. Using Naive Bayes classifier described in this chapter, and any features you can think of, build the best name gender classifier you can. Begin by splitting the Names Corpus into three subsets: 500 words for the test set, 500 words for the dev-test set, and the remaining 6900 words for the training set. Then, starting with the example name gender classifier, make incremental improvements. Use the dev-test set to check your progress. Once you are satisfied with your classifier, check its final performance on the test set.

In [16]:
```
def gender_features(word):
    return {'suffix1': word[-1:], 'suffix2': word[-2:]}
```

In [7]:
```
def gender_features2(name):
    features = {}
    features["first_letter"] = name[0].lower()
    features["last_letter"] = name[-1].lower()
    for letter in 'abcdefghijklmnopqrstuvwxyz':
        features["count({})".format(letter)] = name.lower().count(letter)
        features["has({})".format(letter)] = (letter in name.lower())
    return features
```

In [17]:
```
from nltk.corpus import names
labeled_names = ([(name, 'male') for name in names.words('male.txt')] +
                 [(name, 'female') for name in names.words('female.txt')])
```

In [18]:
```
test_names = labeled_names[:500]
devtest_names = labeled_names[500:1000]
train_names = labeled_names[1000:]
```

In [19]:
```
train_set = [(gender_features(n), gender) for (n, gender) in train_names]
devtest_set = [(gender_features(n), gender) for (n, gender) in devtest_names]
test_set = [(gender_features(n), gender) for (n, gender) in test_names]
classifier = nltk.NaiveBayesClassifier.train(train_set)
print(nltk.classify.accuracy(classifier, devtest_set))
```

```
0.696
```

2. Using the movie review document classifier discussed in Chapter 6- Section 1.3 (constructing a list of the 2500 most frequent words as features and use the first 150 documents as the test dataset) , generate a list of the 10 features that the classifier finds to be most informative. Can you explain why these particular features are informative? Do you find any of them surprising?

In [30]:
```
from nltk.corpus import movie_reviews
import random
documents = [(list(movie_reviews.words(fileid)), category)
             for category in movie_reviews.categories()
             for fileid in movie_reviews.fileids(category)]
random.shuffle(documents)
```

In [31]:
```
all_words = nltk.FreqDist(w.lower() for w in movie_reviews.words())
word_features = list(all_words)[:2500]
```

```python
def document_features(document):
    document_words = set(document)
    features = {}
    for word in word_features:
        features['contains({})'.format(word)] = (word in document_words)
    return features
```

In [33]:
```python
featuresets = [(document_features(d), c) for (d,c) in documents]
train_set, test_set = featuresets[150:], featuresets[:150]
classifier = nltk.NaiveBayesClassifier.train(train_set)
```

In [34]:
```python
print(nltk.classify.accuracy(classifier, test_set))
```

```
0.8266666666666667
```

In [35]:
```python
classifier.show_most_informative_features(10)
```

```
Most Informative Features
   contains(outstanding) = True              pos : neg    =     11.3 : 1.0
        contains(finest) = True              pos : neg    =      8.0 : 1.0
         contains(damon) = True              pos : neg    =      7.9 : 1.0
         contains(mulan) = True              pos : neg    =      7.7 : 1.0
        contains(seagal) = True              neg : pos    =      7.3 : 1.0
   contains(wonderfully) = True              pos : neg    =      7.3 : 1.0
    contains(schumacher) = True              neg : pos    =      6.9 : 1.0
          contains(lame) = True              neg : pos    =      5.9 : 1.0
         contains(flynt) = True              pos : neg    =      5.7 : 1.0
      contains(lebowski) = True              pos : neg    =      5.0 : 1.0
```

These make sense as some include last names of a-list actors and positive adjectives. One that does surprise me is lame as that is usually not used to describe a positive review.

3. Select one of the classification tasks described in this chapter, such as name gender detection, document classification, part-of-speech tagging, or dialog act classification. Using the same training and test data, and the same feature extractor, build three classifiers for the task: a decision tree, a naive Bayes classifier, and a Maximum Entropy classifier. Compare the performance of the three classifiers on your selected task.

In [78]:
```python
sents = nltk.corpus.treebank_raw.sents()
tokens = []
boundaries = set()
offset = 0
for sent in sents:
    tokens.extend(sent)
    offset += len(sent)
    boundaries.add(offset-1)
```

In [79]:
```python
def punct_features(tokens, i):
    return {'next-word-capitalized': tokens[i+1][0].isupper(),
            'prev-word': tokens[i-1].lower(),
             'punct': tokens[i],
             'prev-word-is-one-char': len(tokens[i-1]) == 1}
```

In [80]:
```python
featuresets = [(punct_features(tokens, i), (i in boundaries))
               for i in range(1, len(tokens)-1)
               if tokens[i] in '.?!']
```

In [81]:
```python
size = int(len(featuresets) * .1)
```

In [82]:
```python
train_set,test_set = featuresets[size:], featuresets[:size]
```

In [87]:
```python
nbc_classifier = nltk.NaiveBayesClassifier.train(train_set)
```

In [89]:
```
nltk.classify.accuracy(nbc_classifier, test_set)
```

Out[89]:
```
0.936026936026936
```

In [90]:
```
me_classifier = nltk.MaxentClassifier.train(train_set)
```

```
  ==> Training (100 iterations)

      Iteration    Log Likelihood    Accuracy
      ---------------------------------------
             1          -0.69315        0.610
             2          -0.42328        0.891
             3          -0.31744        0.944
             4          -0.26043        0.944
             5          -0.22517        0.943
             6          -0.20121        0.945
             7          -0.18373        0.945
             8          -0.17027        0.945
             9          -0.15951        0.943
            10          -0.15063        0.943
            11          -0.14314        0.970
            12          -0.13671        0.972
            13          -0.13112        0.972
            14          -0.12619        0.973
            15          -0.12182        0.973
            16          -0.11790        0.973
            17          -0.11436        0.973
            18          -0.11116        0.973
            19          -0.10824        0.973
            20          -0.10556        0.973
            21          -0.10310        0.973
            22          -0.10083        0.973
            23          -0.09873        0.973
            24          -0.09678        0.973
            25          -0.09496        0.973
      Training stopped: keyboard interrupt
         Final          -0.09326        0.973
```

In [91]:
```
dc_classifier = nltk.DecisionTreeClassifier.train(train_set)
```

In [93]:
```
nltk.classify.accuracy(dc_classifier, test_set)
```

Out[93]:
```
0.9730639730639731
```

4. Identify the NPS Chat Corpus, which was demonstrated in Chapter 2, consists of posts from instant messaging sessions. These posts have all been labeled with one of 15 dialogue act types, such as "Statement," "Emotion," "ynQuestion", and "Continuer." We can therefore use this data to build a classifier that can identify the dialogue act types for new instant messaging posts. Build a simple feature extractor that checks what words the post contains. Construct the training and testing data by applying the feature extractor to each post and create a Naïve Bayes classifier. Please print the accuracy of this classifier. Please use NPS Chat Corpus as our dataset and use 8% data as our test data.

In [29]:
```
posts = nltk.corpus.nps_chat.xml_posts()[:10000]
```

In [30]:
```
def dialogue_act_features(post):
    features = {}
    for word in nltk.word_tokenize(post):
        features['contains({})'.format(word.lower())] = True
    return features
```

In [31]:
```
featuresets = [(dialogue_act_features(post.text), post.get('class'))
                for post in posts]
```

In [32]:
```
size = int(len(featuresets) * 0.08)
```

In [33]:
```
train_set, test_set = featuresets[size:], featuresets[:size]
```

In [34]:
```
classifier = nltk.NaiveBayesClassifier.train(train_set)
```

In [35]:
```
print(nltk.classify.accuracy(classifier, test_set))
```

```
0.68125
```

5.Given the following confusion matrix, please calculate: a) Accuracy Rate; b) Precision; c) Recall; d) F-Measure.

```
No   Yes
```

No 104 33 Yes 13 50

In [1]:
```
true_pos, false_pos, true_neg, false_neg, = 50, 13, 104, 33
num_obs = true_pos + false_pos + true_neg + false_neg
```

In [5]:
```
accuracy_rate = (true_pos +true_neg) / num_obs
precision = true_pos/ (true_pos + false_pos)
recall = true_pos / (true_pos + false_neg)
```

In [11]:
```
 print('Accuracy Rate: %s\nPrecision: %s\nRecall: %s' %(accuracy_rate, precision, recall))
```

```
Accuracy Rate: 0.77
Precision: 0.7936507936507936
Recall: 0.6024096385542169
```

Chapter 7

6. Write a tag pattern to match noun phrases containing plural head nouns in the following sentence: "Many researchers discussed this project for two weeks." Try to do this by generalizing the tag pattern that handled singular noun phrases too. Please 1) pos-tag this sentence 2) write a tag pattern (i.e. grammar); 3) use RegexpParser to parse the sentence and 4) print out the result containing NP (noun phrases).

In [71]:
```
grammar = "NP:{<DT>?<CD>?<JJ>*<NNS>}"
cp=nltk.RegexpParser(grammar)
document = "Many researchers discussed this project for two weeks."
sentences = nltk.sent_tokenize(document)
sentences = [nltk.word_tokenize(sent) for sent in sentences]
sentences = [nltk.pos_tag(sent) for sent in sentences]
sentence = [('Many', 'JJ'), ('researchers', 'NNS'), ('discussed', 'VBD'),
        ('this', 'DT'), ('project', 'NN'), ('for', 'IN'), ('two', 'CD'), ('weeks', 'NNS'), ('.', '
result = cp.parse(sentence)
print(result)
```

```
(S
  (NP Many/JJ researchers/NNS)
  discussed/VBD
  this/DT
  project/NN
  for/IN
  (NP two/CD weeks/NNS)
  ./.)
```

7. Write a tag pattern to cover noun phrases that contain gerunds, e.g. "the/DT receiving/VBG end/NN", "assistant/NN managing/VBG editor/NN". Add these patterns to

the grammar, one per line. Test your work using some tagged sentences of your own devising.

In [8]:
```
grammar = "Gerunds:{<DT>?<NN>?<VBG><NN>}"
sentence = [("the","DT"), ("receiving","VBG"), ("end","NN"), ("assistant","NN"), ("managing","VBG"
cp = nltk.RegexpParser(grammar)
tree = cp.parse(sentence)
for subtree in tree.subtrees():
    if subtree.label()=='Gerunds': print (subtree)
```

```
(Gerunds the/DT receiving/VBG end/NN)
(Gerunds assistant/NN managing/VBG editor/NN)
```

8. Use the Brown Corpus and the cascaded chunkers that has patterns for noun phrases, prepositional phrases, verb phrases, and clauses to print out all the verb phrases in the Brown corpus.

In [84]:
```
grammar = r"""
  NP: {<DT|JJ|NN.*>+}
  PP: {<IN><NP>}
  VP: {<VB.*><NP|PP|CLAUSE>+$}
  CLAUSE: {<NP><VP>}
  """
```

In [86]:
```
from nltk.corpus import brown
brown = nltk.corpus.brown
cp = nltk.RegexpParser(grammar)
for sent in brown.tagged_sents():
    tree = cp.parse(sent)
    for subtree in tree.subtrees():
        if subtree.label()=='VP' :print(subtree)
```

```
(VP Ask/VB-HL (NP jail/NN-HL deputies/NNS-HL))
(VP revolving/VBG-HL (NP fund/NN-HL))
(VP Issue/VB-HL (NP jury/NN-HL subpoenas/NNS-HL))
(VP Nursing/VBG-HL (NP home/NN-HL care/NN-HL))
(VP pay/VB-HL (NP doctors/NNS-HL))
(VP nursing/VBG-HL (NP homes/NNS))
(VP Asks/VBZ-HL (NP research/NN-HL funds/NNS-HL))
(VP Regrets/VBZ-HL (NP attack/NN-HL))
(VP Decries/VBZ-HL (NP joblessness/NN-HL))
(VP Underlying/VBG-HL (NP concern/NN-HL))
(VP bar/VB-HL (NP vehicles/NNS-HL))
(VP loses/VBZ-HL (NP pace/NN-HL))
(VP hits/VBZ-HL (NP homer/NN-HL))
(VP attend/VB-HL (NP races/NNS-HL))
(VP follows/VBZ-HL (NP ceremonies/NNS-HL))
(VP Noted/VBN-HL (NP artist/NN-HL))
(VP Cites/VBZ-HL (NP discrepancies/NNS-HL))
(VP calls/VBZ-HL (NP police/NNS-HL))
(VP held/VBN-HL (NP key/NN-HL))
(VP grant/VB-HL (NP bail/NN-HL))
(VP Held/VBD-HL (NP candle/NN-HL))
(VP Expresses/VBZ-HL (NP thanks/NNS-HL))
(VP Gets/VBZ-HL (NP car/NN-HL number/NN-HL))
(VP Attacks/VBZ-HL (NP officer/NN-HL))
(VP oks/VBZ-HL (NP pact/NN-HL))
(VP report/VB-HL (NP gains/NNS-HL))
(VP Pulling/VBG-HL (NP strings/NNS-HL))
(VP United/VBN-TL-HL (NP States/NNS-TL-HL defense/NN-HL))
(VP Betting/VBG-HL (NP men/NNS-HL))
(VP brings/VBZ-HL (NP numbness/NN-HL))
(VP Questions/VBZ-HL (NP shelters/NNS-HL))
(VP Marketing/VBG-HL (NP meat/NN-HL))
(VP Taxing/VBG-HL (NP improvements/NNS-HL))
(VP Praises/VBZ-HL (NP exhibit/NN-HL))
(VP aid/VB (NP international/JJ law/NN))
```

```
(VP retarded/VBN-HL (NP children/NNS-HL))
(VP tormented/VBN-HL (NP span/NN-HL))
(VP dashed/VBN-HL (NP hope/NN-HL))
(VP open/VB-HL (NP program/NN-HL))
(VP Fleeting/VBG-HL (NP glimpse/NN-HL))
(VP fragmented/VBN-HL (NP Society/NN-TL-HL))
(VP locking/VBG-HL (NP bars/NNS-HL))
(VP fire/VB (NP standard/JJ))
(VP Canned/VBN-HL (NP cocktail/NN-HL frankfurters/NNS-HL))
(VP whipped/VBN (NP Salt/NN Paprika/NN))
(VP Barbecued/VBN-HL (NP frankfurters/NNS-HL))
(VP Changing/VBG-HL (NP colors/NNS-HL))
(VP Measuring/VBG-HL (NP armhole/NN-HL))
(VP Backstitching/VBG-HL (NP seam/NN-HL))
(VP Weaving/VBG-HL (NP seam/NN-HL))
(VP drilling/VBG-HL (NP tools/NNS-HL))
(VP drilling/VBG-HL (NP operations/NNS-HL))
(VP Adjoining/VBG-HL (NP areas/NNS-HL))
(VP chinning/VBG-HL (NP bar/NN-HL))
(VP fattening/VBG-HL (NP rations/NNS-HL))
(VP marketing/VBG-HL (NP methods/NNS-HL))
(VP marketing/VBG-HL (NP management/NN-HL))
(VP feeding/VBG-HL (NP facilities/NNS-HL))
(VP Paid/VBN-HL (NP vacations/NNS-HL))
(VP Eating/VBG-HL (NP facilities/NNS-HL))
(VP farming/VBG-HL (NP methods/NNS-HL))
(VP Nourishing/VBG (NP meals/NNS))
(VP save/VB-HL (NP teeth/NNS-HL))
(VP helps/VBZ-HL (NP families/NNS-HL))
(VP Printed/VBN-HL (NP material/NN-HL))
(VP Printed/VBN-HL (NP material/NN-HL))
(VP Printed/VBN-HL (NP material/NN-HL))
(VP Printed/VBN-HL (NP material/NN-HL))
(VP planning/VBG-HL (NP process/NN-HL))
(VP applying/VBG-HL (NP conditions/NNS-HL))
(VP Encouraging/VBG-HL (NP self-help/NN-HL))
(VP stressing/VBG-HL (NP self-help/NN-HL))
(VP nearing/VBG-HL (NP self-sufficiency/NN-HL))
(VP Advertising/VBG-HL (NP program/NN-HL))
(VP Planning/VBG-HL (NP division/NN-HL))
(VP financing/VBG-HL (NP adjustments/NNS-HL))
(VP distributing/VBG-HL (NP funds/NNS-HL))
(VP Matching/VBG-HL (NP requirements/NNS-HL))
(VP prepared/VBN (NP shelter/NN))
(VP Increased/VBN-HL (NP efficiency/NN-HL))
(VP shifting/VBG-HL (NP styles/NNS-HL))
(VP broadcasting/VBG-HL (NP station/NN-HL))
(VP cleaning/VBG-HL (NP process/NN-HL))
(VP frozen/VBN-HL (NP sections/NNS-HL))
(VP Staining/VBG-HL (NP technique/NN-HL))
(VP Concluding/VBG-HL (NP remarks/NNS-HL))
(VP modernizing/VBG-HL (NP societies/NNS-HL))
(VP teaching/VBG-HL (NP methods/NNS-HL))
(VP teaching/VBG-HL (NP methods/NNS-HL))
(VP bargaining/VBG-HL (NP issues/NNS-HL))
(VP
  selecting/VBG-HL
  (NP mail/NN-HL questionnaire/NN-HL method/NN-HL))
(VP mailing/VBG-HL (NP lists/NNS-HL))
(VP distributed/VBN-HL (NP cost/NN-HL analysis/NN-HL))
(VP define/VB-HL (NP input/output/NN-HL control/NN-HL system/NN-HL))
(VP related/VBN-HL (NP materials/NNS-HL))
(VP ionizing/VBG-HL (NP radiation/NN-HL))
(VP seen/VBN (NP that/DT Af/NN))
(VP
  Chipping/VBG
  (NP mechanism/NN)
  (PP of/IN (NP cohesive/JJ failure/NN)))
(VP cracking/VBG-HL (NP mechanism/NN-HL))
(VP Processing/VBG-HL (NP urethanes/NNS-HL))
```

```
(VP coupled/VBN-HL (NP image/NN-HL intensifiers/NNS-HL))
(VP following/VBG (NP morning/NN))
(VP whining/VBG (NP voice/NN))
(VP convinced/VBN (PP of/IN (NP that/DT)))
```

9. The bigram chunker scores about 90% accuracy. Using bigram_chunker.tagger.tag(postags) to examine the results and study its errors. Then experiment with trigram chunking. Are you able to improve the performance any more?

In [98]:
```python
from nltk.corpus import conll2000
```

In [99]:
```python
cp = nltk.RegexpParser("")
```

In [100…
```python
class BigramChunker(nltk.ChunkParserI):
    def __init__(self, train_sents):
        train_data = [[(t, c) for w, t, c in nltk.chunk.tree2conlltags(sent)]
                      for sent in train_sents]
        self.tagger = nltk.BigramTagger(train_data)

    def parse(self, sentence):
        pos_tags = [pos for (word, pos) in sentence]
        tagged_pos_tags = self.tagger.tag(pos_tags)
        chunktags = [chunktag for (pos, chunktag) in tagged_pos_tags]
        conlltags = [(word, pos, chunktag) for ((word, pos), chunktag)
                     in zip(sentence, chunktags)]
        return nltk.chunk.conlltags2tree(conlltags)
```

In [101…
```python
class TrigramChunker(nltk.ChunkParserI):
    def __init__(self, train_sents):
        train_data = [[(t, c) for w, t, c in nltk.chunk.tree2conlltags(sent)]
                      for sent in train_sents]
        self.tagger = nltk.TrigramTagger(train_data)

    def parse(self, sentence):
        pos_tags = [pos for (word, pos) in sentence]
        tagged_pos_tags = self.tagger.tag(pos_tags)
        chunktags = [chunktag for (pos, chunktag) in tagged_pos_tags]
        conlltags = [(word, pos, chunktag) for ((word, pos), chunktag)
                     in zip(sentence, chunktags)]
        return nltk.chunk.conlltags2tree(conlltags)
```

In [102…
```python
test_sents = conll2000.chunked_sents('test.txt', chunk_types=['NP'])
```

In [103…
```python
train_sents = conll2000.chunked_sents('train.txt', chunk_types=['NP'])
```

In [104…
```python
bigram_chunker = BigramChunker(train_sents)
```

In [105…
```python
print(bigram_chunker.evaluate(test_sents))
```

```
ChunkParse score:
    IOB Accuracy:  93.3%%
    Precision:     82.3%%
    Recall:        86.8%%
    F-Measure:     84.5%%
```

In [56]:
```python
trigram_chunker = TrigramChunker(train_sents)
```

In [58]:
```python
print(trigram_chunker.evaluate(test_sents))
```

```
ChunkParse score:
    IOB Accuracy:  93.3%%
    Precision:     82.5%%
```

```
Recall:         86.8%%
F-Measure:      84.6%%
```

10. Explore the Brown Corpus to print out all the FACILITIES (one of the commonly used types of name entities).

In [11]:

```python
for sent in brown.tagged_sents():
    tree = nltk.ne_chunk(sent)
    for subtree in tree.subtrees():
        if subtree.label() == 'FACILITY': print(subtree)
```

```
(FACILITY Raymondville/NP)
(FACILITY White/JJ-TL)
(FACILITY White/JJ-TL)
(FACILITY White/JJ-TL)
(FACILITY White/JJ-TL)
(FACILITY White/JJ-TL)
(FACILITY White/JJ-TL)
(FACILITY Kremlin/NP)
(FACILITY White/JJ-TL)
(FACILITY White/JJ-TL)
(FACILITY White/JJ-TL House/NN-TL)
(FACILITY White/JJ-TL House/NN-TL)
(FACILITY Franklin/NP-TL)
(FACILITY Kremlin/NP)
(FACILITY Franklin/NP-TL Square/NN-TL)
(FACILITY Pennsylvania/NP-TL Avenue/NN-TL)
(FACILITY Jenks/NP-TL Street/NN-TL)
(FACILITY White/JJ-TL)
(FACILITY White/JJ-TL)
(FACILITY White/JJ-TL)
(FACILITY White/JJ-TL)
(FACILITY White/JJ-TL)
(FACILITY White/JJ-TL)
(FACILITY White/JJ-TL)
(FACILITY White/JJ-TL House/NN-TL)
(FACILITY White/JJ-TL)
(FACILITY White/JJ-TL)
(FACILITY White/JJ-TL)
(FACILITY White/JJ-TL)
(FACILITY White/JJ-TL)
(FACILITY White/JJ-TL)
(FACILITY White/JJ-TL)
(FACILITY White/JJ-TL)
(FACILITY White/JJ-TL)
(FACILITY Pensacola/NP)
(FACILITY White/JJ-TL Sox/NPS-TL)
(FACILITY Caltech/NP)
(FACILITY White/JJ-TL House/NN-TL)
(FACILITY White/JJ-TL)
(FACILITY Caracas/NP)
(FACILITY White/JJ-TL)
(FACILITY White/JJ-TL House/NN-TL)
(FACILITY White/JJ-TL)
(FACILITY White/JJ-TL)
(FACILITY White/JJ-TL Way/NN-TL)
(FACILITY White/JJ-TL)
(FACILITY Kremlin/NP)
(FACILITY Kremlin/NP)
(FACILITY Madison/NP-TL Square/NN-TL Garden/NN-TL)
(FACILITY Boron/NP)
(FACILITY Rome/NP)
(FACILITY Rome/NP)
(FACILITY Grafton/NP)
(FACILITY Bari/NP)
(FACILITY Bari/NP)
(FACILITY Northfield/NP)
(FACILITY Baltimore/NP)
(FACILITY Hilo/NP)
```

```
(FACILITY Hilo/NP)
(FACILITY White/JJ-TL)
(FACILITY White/JJ-TL)
(FACILITY White/JJ-TL House/NN-TL)
(FACILITY White/JJ-TL House/NN-TL)
(FACILITY White/JJ-TL)
(FACILITY White/JJ-TL House/NN-TL)
(FACILITY White/JJ-TL)
(FACILITY White/JJ-TL)
(FACILITY White/JJ-TL)
(FACILITY White/JJ-TL)
(FACILITY White/JJ-TL House/NN-TL)
(FACILITY White/JJ-TL House/NN-TL)
(FACILITY White/JJ-TL House/NN-TL)
(FACILITY White/JJ-TL House/NN-TL)
(FACILITY White/JJ-TL)
(FACILITY White/JJ-TL)
(FACILITY White/JJ-TL)
(FACILITY Kremlin/NP)

---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-11-f991cb928b20> in <module>
      1 for sent in brown.tagged_sents():
----> 2     tree = nltk.ne_chunk(sent)
      3     for subtree in tree.subtrees():
      4         if subtree.label() == 'FACILITY': print(subtree)

~\anaconda3\lib\site-packages\nltk\chunk\__init__.py in ne_chunk(tagged_tokens, binary)
    184         chunker_pickle = _MULTICLASS_NE_CHUNKER
    185     chunker = load(chunker_pickle)
--> 186     return chunker.parse(tagged_tokens)
    187 
    188 

~\anaconda3\lib\site-packages\nltk\chunk\named_entity.py in parse(self, tokens)
    124         Each token should be a pos-tagged word
    125         """
--> 126         tagged = self._tagger.tag(tokens)
    127         tree = self._tagged_to_parse(tagged)
    128         return tree

~\anaconda3\lib\site-packages\nltk\tag\sequential.py in tag(self, tokens)
     61         tags = []
     62         for i in range(len(tokens)):
---> 63             tags.append(self.tag_one(tokens, i, tags))
     64         return list(zip(tokens, tags))
     65 

~\anaconda3\lib\site-packages\nltk\tag\sequential.py in tag_one(self, tokens, index, history)
     81         tag = None
     82         for tagger in self._taggers:
---> 83             tag = tagger.choose_tag(tokens, index, history)
     84             if tag is not None:
     85                 break

~\anaconda3\lib\site-packages\nltk\tag\sequential.py in choose_tag(self, tokens, index, history)
    646         # higher than that cutoff first; otherwise, return None.
    647         if self._cutoff_prob is None:
--> 648             return self._classifier.classify(featureset)
    649 
    650         pdist = self._classifier.prob_classify(featureset)

~\anaconda3\lib\site-packages\nltk\classify\maxent.py in classify(self, featureset)
    139 
    140     def classify(self, featureset):
--> 141         return self.prob_classify(featureset).max()
    142 
    143     def prob_classify(self, featureset):

~\anaconda3\lib\site-packages\nltk\classify\maxent.py in prob_classify(self, featureset)
```

```
    144         prob_dict = {}
    145         for label in self._encoding.labels():
--> 146             feature_vector = self._encoding.encode(featureset, label)
    147 
    148             if self._logarithmic:

~\anaconda3\lib\site-packages\nltk\classify\maxent.py in encode(self, featureset, label)
    578 
    579             # Otherwise, we might want to fire an "unseen-value feature".
--> 580             elif self._unseen:
    581                 # Have we seen this fname/fval combination with any label?
    582                 for label2 in self._labels:

KeyboardInterrupt: 
```