Using NLTK for text classification - Akash Gupta

This note demonstrates various Python functions and primarily the **NLTK** library to carry out text classification tasks.

```
import nltk
import re
import pprint

# inputs are converted into feature sets which are then fed to the ML algorithm
# along with training labels. Then for testing, again inputs are converted into
# extracted feature sets which are fed to the model to get the labels.
```

```
# modeling differences in gender names. Decide what features are important and encode them.

def gender_features(word):
    return {'last letter': word[-1]}
gender_features('shrek')     ## output of this is a feature set
```

```
{'last letter': 'k'}
```

```
from nltk.corpus import names
import random

names1 = ([(name,'male') for name in names.words('male.txt')] +
         [(name,'female') for name in names.words('female.txt')])
random.shuffle(names1)
```

```
# training a naive bayes classifier

featuresets = [(gender_features(n),g) for (n,g) in names1]
train_set, test_set = featuresets[500:], featuresets[:500]
classifier = nltk.NaiveBayesClassifier.train(train_set)

classifier.classify(gender_features('neo'))
classifier.classify(gender_features('trinity'))

print(nltk.classify.accuracy(classifier, test_set))
```

```
0.77
```

```
# compute likelihood ratios

classifier.show_most_informative_features(5)
```

```
Most Informative Features
             last letter = 'k'              male : female =     42.5 : 1.0
             last letter = 'a'            female : male   =     36.1 : 1.0
             last letter = 'p'              male : female =     21.0 : 1.0
             last letter = 'f'              male : female =     15.3 : 1.0
             last letter = 'v'              male : female =     10.5 : 1.0
```

```
# demonstrating overfitting with a very detailed feature extraction. training not generalizing
# well to new data refers to the problem of overfitting

def gender_features2(name):
    features = {}
    features['first letter'] = name[0].lower()
```

```
    features['last letter'] = name[-1].lower()
    letters = 'abcdefghijklmnopqrstuvwxyz'
    for l in letters:
        features['count(%s)'% l] = name.lower().count(l)
        features['has(%s)'% l] = (l in name.lower())
    return features

gender_features2('John')

featuresets = [(gender_features2(n),g) for (n,g) in names1]
train_set, test_set = featuresets[500:],featuresets[:500]
classifier = nltk.NaiveBayesClassifier.train(train_set)
print(nltk.classify.accuracy(classifier,test_set))
```

```
0.752
```

```
# for properly evaluating a model we create a development set - divide it into training set
# and dev-test set. training trains the model and dev-test does error analysis. test set is the
# final evalutation system

train_names = names1[1500:]
devtest_names = names1[500:1500]
test_names = names1[:500]

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

```
0.751
```

```
# generate list of error classifier makes in devtest

errors = []

for (name, tag) in devtest_names:
    guess = classifier.classify(gender_features(name))
    if guess != tag:
        errors.append((tag, guess, name))

# make changes to featureset based on the errors generated

for (tag, guess, name) in sorted(errors):
    print("correct=%-8s guess=%-8s name=%-30s"%(tag,guess,name))

len(errors)
```

```
correct=female   guess=male     name=Agnes
correct=female   guess=male     name=Aimil
correct=female   guess=male     name=Alis
correct=female   guess=male     name=Alisun
correct=female   guess=male     name=Allis
correct=female   guess=male     name=Allison
correct=female   guess=male     name=Ann
```

```
correct=female   guess=male     name=Arden
correct=female   guess=male     name=Aryn
correct=female   guess=male     name=Barb
correct=female   guess=male     name=Bert
correct=female   guess=male     name=Bren
correct=female   guess=male     name=Bridgett
correct=female   guess=male     name=Carlen
correct=female   guess=male     name=Carmel
correct=female   guess=male     name=Carmen
correct=female   guess=male     name=Caro
correct=female   guess=male     name=Carroll
correct=female   guess=male     name=Caryl
correct=female   guess=male     name=Cathryn
correct=female   guess=male     name=Charmian
correct=female   guess=male     name=Christean
correct=female   guess=male     name=Christen
correct=female   guess=male     name=Chrysler
correct=female   guess=male     name=Coral
correct=female   guess=male     name=Coralyn
correct=female   guess=male     name=Dagmar
correct=female   guess=male     name=Daniel
correct=female   guess=male     name=Daryl
correct=female   guess=male     name=Del
correct=female   guess=male     name=Dell
correct=female   guess=male     name=Dido
correct=female   guess=male     name=Dionis
correct=female   guess=male     name=Doralynn
correct=female   guess=male     name=Doris
correct=female   guess=male     name=Drew
correct=female   guess=male     name=Eilis
correct=female   guess=male     name=Elisabet
correct=female   guess=male     name=Elizabet
correct=female   guess=male     name=Emmalynn
correct=female   guess=male     name=Eran
correct=female   guess=male     name=Eryn
correct=female   guess=male     name=Estel
correct=female   guess=male     name=Flower
correct=female   guess=male     name=Gail
correct=female   guess=male     name=Gert
correct=female   guess=male     name=Gertrudis
correct=female   guess=male     name=Grethel
correct=female   guess=male     name=Gwendolin
correct=female   guess=male     name=Ivett
correct=female   guess=male     name=Jacklin
correct=female   guess=male     name=Jean
correct=female   guess=male     name=Jenifer
correct=female   guess=male     name=Jerrilyn
correct=female   guess=male     name=Jillian
correct=female   guess=male     name=Joell
correct=female   guess=male     name=Joyan
correct=female   guess=male     name=Juliet
correct=female   guess=male     name=Kaitlynn
correct=female   guess=male     name=Karlen
correct=female   guess=male     name=Karylin
```

```
correct=female   guess=male     name=Katheryn
correct=female   guess=male     name=Kathleen
correct=female   guess=male     name=Kathlin
correct=female   guess=male     name=Kathryn
correct=female   guess=male     name=Kellyann
correct=female   guess=male     name=Kial
correct=female   guess=male     name=Kiersten
correct=female   guess=male     name=Kit
correct=female   guess=male     name=Kris
correct=female   guess=male     name=Kristan
correct=female   guess=male     name=Krystal
correct=female   guess=male     name=Lamb
correct=female   guess=male     name=Laural
correct=female   guess=male     name=Lauren
correct=female   guess=male     name=Lauryn
correct=female   guess=male     name=Lillian
correct=female   guess=male     name=Lois
correct=female   guess=male     name=Loreen
correct=female   guess=male     name=Lou
correct=female   guess=male     name=Lust
correct=female   guess=male     name=Margalo
correct=female   guess=male     name=Margaret
correct=female   guess=male     name=Margit
correct=female   guess=male     name=Marigold
correct=female   guess=male     name=Marilyn
correct=female   guess=male     name=Marin
correct=female   guess=male     name=Maris
correct=female   guess=male     name=Marj
correct=female   guess=male     name=Maryellen
correct=female   guess=male     name=Marys
correct=female   guess=male     name=Mehetabel
correct=female   guess=male     name=Mel
correct=female   guess=male     name=Mellisent
correct=female   guess=male     name=Michal
correct=female   guess=male     name=Michell
correct=female   guess=male     name=Morgan
correct=female   guess=male     name=Noell
correct=female   guess=male     name=Noellyn
correct=female   guess=male     name=Peg
correct=female   guess=male     name=Pen
correct=female   guess=male     name=Persis
correct=female   guess=male     name=Quinn
correct=female   guess=male     name=Rosalind
correct=female   guess=male     name=Rosario
correct=female   guess=male     name=Roselyn
correct=female   guess=male     name=Rozamond
correct=female   guess=male     name=Ruthann
correct=female   guess=male     name=Sal
correct=female   guess=male     name=Sallyann
correct=female   guess=male     name=Sara-Ann
correct=female   guess=male     name=Sherill
correct=female   guess=male     name=Sib
correct=female   guess=male     name=Suellen
correct=female   guess=male     name=Terri-Jo
```

```
correct=female   guess=male     name=Tess
correct=female   guess=male     name=Wynn
correct=male     guess=female   name=Allah
correct=male     guess=female   name=Anatoly
correct=male     guess=female   name=Anthony
correct=male     guess=female   name=Antony
correct=male     guess=female   name=Archy
correct=male     guess=female   name=Aube
correct=male     guess=female   name=Aubrey
correct=male     guess=female   name=Barde
correct=male     guess=female   name=Barrie
correct=male     guess=female   name=Benji
correct=male     guess=female   name=Bentley
correct=male     guess=female   name=Bjorne
correct=male     guess=female   name=Bradley
correct=male     guess=female   name=Carleigh
correct=male     guess=female   name=Case
correct=male     guess=female   name=Chance
correct=male     guess=female   name=Chrissy
correct=male     guess=female   name=Clarke
correct=male     guess=female   name=Claude
correct=male     guess=female   name=Clemente
correct=male     guess=female   name=Clemmie
correct=male     guess=female   name=Cy
correct=male     guess=female   name=Danny
correct=male     guess=female   name=Davie
correct=male     guess=female   name=Derby
correct=male     guess=female   name=Dominique
correct=male     guess=female   name=Drake
correct=male     guess=female   name=Duffie
correct=male     guess=female   name=Durante
correct=male     guess=female   name=Dwaine
correct=male     guess=female   name=Eddy
correct=male     guess=female   name=Elisha
correct=male     guess=female   name=Elroy
correct=male     guess=female   name=Ely
correct=male     guess=female   name=Erny
correct=male     guess=female   name=Fletch
correct=male     guess=female   name=Freddy
correct=male     guess=female   name=Garry
correct=male     guess=female   name=Garvy
correct=male     guess=female   name=Georgia
correct=male     guess=female   name=Georgy
correct=male     guess=female   name=Gerome
correct=male     guess=female   name=Giffy
correct=male     guess=female   name=Graehme
correct=male     guess=female   name=Granville
correct=male     guess=female   name=Griffith
correct=male     guess=female   name=Guthry
correct=male     guess=female   name=Henri
correct=male     guess=female   name=Henrie
correct=male     guess=female   name=Henrique
correct=male     guess=female   name=Hersch
correct=male     guess=female   name=Hezekiah
```

```
correct=male     guess=female   name=Hilary
correct=male     guess=female   name=Irvine
correct=male     guess=female   name=Isa
correct=male     guess=female   name=Isadore
correct=male     guess=female   name=Jackie
correct=male     guess=female   name=Jake
correct=male     guess=female   name=Jamie
correct=male     guess=female   name=Jay
correct=male     guess=female   name=Jeremie
correct=male     guess=female   name=Jerry
correct=male     guess=female   name=Jessee
correct=male     guess=female   name=Jodie
correct=male     guess=female   name=Jody
correct=male     guess=female   name=Johnnie
correct=male     guess=female   name=Joseph
correct=male     guess=female   name=Juanita
correct=male     guess=female   name=Judah
correct=male     guess=female   name=Kenny
correct=male     guess=female   name=Lawrence
correct=male     guess=female   name=Lazare
correct=male     guess=female   name=Leroy
correct=male     guess=female   name=Leslie
correct=male     guess=female   name=Lyle
correct=male     guess=female   name=Maurise
correct=male     guess=female   name=Merry
correct=male     guess=female   name=Mike
correct=male     guess=female   name=Moore
correct=male     guess=female   name=Murdoch
correct=male     guess=female   name=Myke
correct=male     guess=female   name=Obadiah
correct=male     guess=female   name=Ole
correct=male     guess=female   name=Ollie
correct=male     guess=female   name=Orbadiah
correct=male     guess=female   name=Partha
correct=male     guess=female   name=Pate
correct=male     guess=female   name=Pattie
correct=male     guess=female   name=Penny
correct=male     guess=female   name=Pierce
correct=male     guess=female   name=Randy
correct=male     guess=female   name=Rawley
correct=male     guess=female   name=Reggy
correct=male     guess=female   name=Reilly
correct=male     guess=female   name=Rickie
correct=male     guess=female   name=Ricky
correct=male     guess=female   name=Roderich
correct=male     guess=female   name=Rodrique
correct=male     guess=female   name=Ronny
correct=male     guess=female   name=Roscoe
correct=male     guess=female   name=Sascha
correct=male     guess=female   name=Saundra
correct=male     guess=female   name=Scotty
correct=male     guess=female   name=Serge
correct=male     guess=female   name=Sheffie
correct=male     guess=female   name=Sherlocke
```

```
correct=male     guess=female   name=Shorty
correct=male     guess=female   name=Si
correct=male     guess=female   name=Skelly
correct=male     guess=female   name=Solly
correct=male     guess=female   name=Spense
correct=male     guess=female   name=Stacy
correct=male     guess=female   name=Stanley
correct=male     guess=female   name=Stanly
correct=male     guess=female   name=Sunny
correct=male     guess=female   name=Teddy
correct=male     guess=female   name=Tedie
correct=male     guess=female   name=Terrence
correct=male     guess=female   name=Thaine
correct=male     guess=female   name=Timmy
correct=male     guess=female   name=Torey
correct=male     guess=female   name=Torry
correct=male     guess=female   name=Tremaine
correct=male     guess=female   name=Uri
correct=male     guess=female   name=Vasily
correct=male     guess=female   name=Wallace
correct=male     guess=female   name=Wash
correct=male     guess=female   name=Wye
correct=male     guess=female   name=Yancy
correct=male     guess=female   name=Yehudi
correct=male     guess=female   name=Zacharie
correct=male     guess=female   name=Zeke
```

249

```python
# we see that sometimes last two letters can be indicative of the gender. 'ch' ending tends to
# be associated with male even though 'h' is associated more with female. Adjust featureset

def gender_features(word):
    return {'suffix 1': word[-1:], 'suffix 2': word[-2:]}

# rebuild the classifier

train_set = [(gender_features(n),g) for (n,g) in train_names]
devtest_set = [(gender_features(n),g) for (n,g) in devtest_names]
classifier = nltk.NaiveBayesClassifier.train(train_set)
print(nltk.classify.accuracy(classifier, devtest_set))
```

0.763

```python
# categorizing movie reviews as positive or negative

from nltk.corpus import movie_reviews

documents = [(list(movie_reviews.words(fileid)),category) for category
            in movie_reviews.categories() for fileid in movie_reviews.fileids(category)]
```

```
random.shuffle(documents)
```

```
# define as a feature as whether a document contains a certain word or not
# we make a list of 2000 most frequent words and then define a feature extractor that
# check whether that document has the word or not

all_words = nltk.FreqDist(w.lower() for w in movie_reviews.words())
word_features = list(all_words.keys())[:2000]

def document_features(document):
    docwords = set(document)
    features = {}
    for word in word_features:
        features['contains(%s)'%word] = (word in docwords)
    return features

print(document_features(movie_reviews.words('pos/cv957_8737.txt')))
```

```
# now use the defined feature extractor to train a classifier that labels movie reviews

featuresets = [(document_features(d),c) for (d,c) in documents]
train_set, test_set = featuresets[100:], featuresets[:100]
classifier = nltk.NaiveBayesClassifier.train(train_set)
print(nltk.classify.accuracy(classifier,test_set))
classifier.show_most_informative_features(5)
```

```
0.79
Most Informative Features
    contains(schumacher) = True              neg : pos    =      7.3 : 1.0
        contains(shoddy) = True              neg : pos    =      6.9 : 1.0
 contains(unimaginative) = True              neg : pos    =      6.9 : 1.0
     contains(atrocious) = True              neg : pos    =      6.5 : 1.0
        contains(turkey) = True              neg : pos    =      6.3 : 1.0
```

```
# classifying POS by finding which suffixes are most informative

from nltk.corpus import brown
suffix_fdist = nltk.FreqDist()

for word in brown.words():
    word = word.lower()
    suffix_fdist[word[-1:]] += 1
    suffix_fdist[word[-2:]] += 1
    suffix_fdist[word[-3:]] += 1

common_suffixes = [suffix for (suffix,count) in suffix_fdist.most_common(100)]
print(common_suffixes)
```

```
['e', ',', '.', 's', 'd', 't', 'he', 'n', 'a', 'of', 'the', 'y', 'r', 'to', 'in', 'f', 'o', 'ed', 'nd',
```

```
# feature extracter checks a word for these suffixes

def pos_features(word):
    features = {}
    for suffix in common_suffixes:
        features['endswith({})'.format(suffix)] = word.lower().endswith(suffix)
```

```
    return features

# classifier will use this to label the inputs
```

```
# building a decision tree classifier

tagged_words = brown.tagged_words(categories = 'news')
featuresets = [(pos_features(n),g) for (n,g) in tagged_words]

size = int(len(featuresets)*0.1)
```

```
train_set, test_set = featuresets[1:2000], featuresets[:size]
```

```
classifier = nltk.DecisionTreeClassifier.train(train_set)
```

```
nltk.classify.accuracy(classifier,test_set)
classifier.classify(pos_features('cat'))
```

```
'IN'
```

```
# context dependent feature extractor that defines our POS tag classifier. Identity of previous
# word is included as a feature

def pos_features(sentence, i):
    features = {'suffix(1)':sentence[i][-1:],
                'suffix(2)':sentence[i][-2:],
                'suffix(3)':sentence[i][-3:]}
    if i==0:
        features['prev-word'] = '<START>'
    else:
        features['prev-word'] = sentence[i-1]
    return features

pos_features(brown.sents()[0], 8)
```

```
{'suffix(1)': 'n', 'suffix(2)': 'on', 'suffix(3)': 'ion', 'prev-word': 'an'}
```

```
tagged_sents = brown.tagged_sents(categories = 'news')
featuresets = []
for tagged_sent in tagged_sents:
    untagged_sent = nltk.tag.untag(tagged_sent)
    for i,(word,tag) in enumerate(tagged_sent):
        featuresets.append((pos_features(untagged_sent,i),tag))

size = int(len(featuresets)*0.1)
```

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

```
classifier = nltk.NaiveBayesClassifier.train(train_set)
```

```
nltk.classify.accuracy(classifier,test_set)
```

```
0.7891596220785678
```

```
classifier.classify(pos_features(brown.sents()[9],5))
```

```
'AT'
```

```python
# joint classifier or sequence classifier does POS tagging for a given sequence of words in a sentence
# consecutive or greedy classification is employed. first word is tagged based on most likely tag
# then the next word is tagged taking into account the previous tag and so on.
# a history variables keeps an account of what all the tagger has already tagged

def pos_features(sentence, i, history):
    features = {'suffix(1)':sentence[i][-1:]
               ,'suffix(2)':sentence[i][-2:]
               ,'suffix(3)':sentence[i][-3:]}
    if i==0:
        features['prev-word'] = '<START>'
        features['prev-tag'] = '<START>'
    else:
        features['prev-word'] = sentence[i-1]
        features['prev-tag'] = history[i-1]
    return features

class ConsecutivePosTagger(nltk.TaggerI):
    def __init__(self, train_sents):
        train_set = []
        for tagged_sent in train_sents:
            untagged_sent = nltk.tag.untag(tagged_sent)
            history = []
            for i,(word,tag) in enumerate(tagged_sent):
                featureset = pos_features(untagged_sent, i, history)
                train_set.append((featureset,tag))
                history.append(tag)
        self.classifier = nltk.NaiveBayesClassifier.train(train_set)

    def tag(self, sentence):
        history = []
        for i,word in enumerate(sentence):
            featureset = pos_features(sentence, i, history)
            tag = self.classifier.classify(featureset)
            history.append(tag)
        return zip(sentence,history)

tagged_sents = brown.tagged_sents(categories = 'news')
size = int(len(tagged_sents)*0.1)
train_sents, test_sents = tagged_sents[size:],tagged_sents[:size]
tagger = ConsecutivePosTagger(train_sents)
print(tagger.evaluate(test_sents))
```

```
0.7980528511821975
```

```python
# another method is to assign scores to all possible POS tags and choose the sequence
# with the highest score - HMM. It finds the probability distribution of a word over all
# possible tags. These probabilities are combined to form score which determine our final choice
# instead of looking at ALL possible tags for the probability distribution over a word, we look
# at the n most recent tags
```

```python
# sentence segmentation is the classification of punctuation - marking the end of sentences or not
# obtain data that has already been segmented into sentences and convert it to a form
# such that features can be extracted from it.
```

```python
# marking sentence end boundaries.

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)

# boundaries stores the index of the sentence end token
```

```python
# specify features to decide if puncuation is used to mark sentence boundary

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

```python
# select punctuation marks and label whether they are boundary tokens or not

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

```python
# now train and evaluate a punctuation classifier

size = int(len(featuresets)*0.1)
train_set,test_set = featuresets[size:],featuresets[:size]
classifier = nltk.NaiveBayesClassifier.train(train_set)
nltk.classify.accuracy(classifier,test_set)
```

```
0.936026936026936
```

```python
def segment_sentence(words):
    start = 0
    sents = []
    for i, word in enumerate(words):
        if word in '.?!' and classifier.classify(punct_features(words,i))==True:
            sents.append(words[start:i+1])
            start = i+1
    if start < len(words):
        sents.append(words[start:])
    return sents

segment_sentence(['checking','the','.','sent','classifier','.','Now'])
```

```
[['checking', 'the', '.', 'sent', 'classifier', '.'], ['Now']]
```

```python
# identifying dialogue act types - whether person said something as a statement, question, etc.
# first extract basic messaging data from chat corpus using XML extraction

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

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

featuresets = [(dialogue_act_features(post.text), post.get('class')) for post in posts]
size = int(len(featuresets)*0.1)
train_set, test_set = featuresets[size:],featuresets[:size]
classifier = nltk.NaiveBayesClassifier.train(train_set)
nltk.classify.accuracy(classifier,test_set)
classifier.classify(dialogue_act_features(posts[5].text))
```

```
'System'
```

```python
# recognizing textual entailment involves determining if a given piece of text entails
# another piece of text called the hypothesis. True means entailment holds and False means it doesn't
# there is entailment if information found in the hypothesis is found in text and false if information
# in hypothesis does not match the text -- hyp_extra()
# we check the degree of word overlap and degree to which there are words in hypothesis but not in text
# names entitities (nes) are important factors and stopwords can be filtered out.


def rte_features(rtepair):
    extractor = nltk.RTEFeatureExtractor(rtepair)
    features = {}
    features['word overlap'] = len(extractor.overlap('word'))
    features['word_hyp_extra'] = len(extractor.hyp_extra('word'))
    features['ne_overlap'] = len(extractor.overlap('ne'))
    features['ne_hyp_extra'] = len(extractor.hyp_extra('ne'))
    return features

#rtepair = nltk.corpus.rte.pairs(['rte3_dev.xml'])[33]
#extractor = nltk.RTEFeatureExtractor(rtepair)
#print(extractor.text_words)
```

```python
# remember that accuracy measures the percentage of inputs correctly labeled by the classifier

# confusion matrix for unigram tagger
from nltk.corpus import brown
brown_sents = brown.sents(categories = 'news')
brown_tagged_sents = brown.tagged_sents(categories = 'news')
unigram_tagger = nltk.UnigramTagger(brown_tagged_sents)

size = int(len(brown_tagged_sents)*0.9)
train_sents = brown_tagged_sents[:size]
test_sents = brown_tagged_sents[size:]
unigram_tagger = nltk.UnigramTagger(train_sents)

t0 = nltk.DefaultTagger('NN')
t1 = nltk.UnigramTagger(train_sents, backoff = t0)
t2 = nltk.BigramTagger(train_sents, backoff = t1)
```

```python
def tag_list(tagged_sents):
    return [tag for sent in tagged_sents for (word,tag) in sent]
```

```python
def apply_tagger(tagger,corpus):
    return [tagger.tag(nltk.tag.untag(sent)) for sent in corpus]

gold = tag_list(brown.tagged_sents(categories = 'editorial'))
test = tag_list(apply_tagger(t2, brown.tagged_sents(categories = 'editorial')))
```

```python
cm = nltk.ConfusionMatrix(gold,test)
print(cm.pretty_format(sort_by_count = True, show_percents = True, truncate = 9))
```

```
    |                                       N                   |
    |      N      I      A      J           N             V      N |
    |      N      N      T      J      .    S      ,      B      P |
----+--------------------------------------------------------------+
 NN | <11.9%>  0.0%      .   0.2%      .   0.0%      .   0.2%   0.0% |
 IN |   0.0%  <9.0%>     .      .      .   0.0%      .      .      . |
 AT |      .      .  <8.6%>     .      .      .      .      .      . |
 JJ |   1.6%      .      .  <4.0%>     .      .      .   0.0%   0.0% |
  . |      .      .      .      .  <4.8%>     .      .      .      . |
NNS |   1.5%      .      .      .      .  <3.3%>     .      .   0.0% |
  , |      .      .      .      .      .      .  <4.4%>     .      . |
 VB |   0.9%      .      .   0.0%      .      .      .  <2.4%>     . |
 NP |   1.0%      .      .   0.0%      .      .      .      .  <1.9%>|
----+--------------------------------------------------------------+
(row = reference; col = test)
```

```python
# have a look into cross validation methods to imporve score of a classifier

# calculating entropy

import math

def entropy(labels):
    freqdist = nltk.FreqDist(labels)
    probs = [freqdist.freq(l) for l in nltk.FreqDist(labels)]
    return -sum([p*math.log(p,2) for p in probs])

print(entropy(['female','female','male','male']))
```

```
1.0
```