In [25]:

```
import nltk
import pandas as pd
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression

train_data = open('task2_lemmas_train')
```

In [26]:

```
i = 0
train_set = []
for line in train_data:
    if i != 0:
        b = line.split(',')
        word = b[1]
        infinitive = b[2].split('+')[0]
        word_form = b[2].split('+')[1][:-1]
        train_set.append((word, infinitive, word_form))
    i += 1
```

In [27]:

```
print(train_set[:10])
```

```
[('vergognerete', 'vergognare', 'V'), ('amnistiavate', 'amnistiare', 'V'),
 ('menomazione', 'menomazione', 'N'), ('sfaldavamo', 'sfaldare', 'V'), ('sfo
dererei', 'sfoderare', 'V'), ('ascondesti', 'ascondere', 'V'), ('edificheres
te', 'edificare', 'V'), ('maschieran', 'maschiare', 'V'), ('transennasser',
 'transennare', 'V'), ('computando', 'computare', 'V')]
```

In [28]:

```
def define_common_part_of_words(w1, w2):
    a = 0
    len_ = min(len(w1), len(w2))
    for i in range(len_):
        if w1[i] != w2[i]:
            break
        i += 1
    res = (w1[a:], w2[a:])
    return res

def define_not_commot_part_of_words(w1, w2):
    a = 0
    len_ = min(len(w1), len(w2))
    for i in range(len_):
        if w1[i] != w2[i]:
            break
        a += 1
    res = (w1[a:], w2[a:])
    return res

import time
def define_set_of_features(word, infinitive):
    feature = {}
    feature['suffix'] = define_not_commot_part_of_words(word, infinitive)[0]
    if ( len(define_common_part_of_words(word, infinitive)[0]) >= 2):
        feature['last'] = define_common_part_of_words(word, infinitive)[0][-1]
    else:
        feature['last'] = ''
        print(word)
        print(infinitive)
        print(define_not_commot_part_of_words(word, infinitive)[0])
        print(define_not_commot_part_of_words(word, infinitive)[1])
        #print(define_common_part_of_words(word, infinitive))
        #time.sleep(1)
        print('\n')
    if( len(define_common_part_of_words(word, infinitive)[0]) >= 2):
        feature['prelast'] = define_common_part_of_words(word, infinitive)[0][-2]
    else:
        feature['prelast'] = ''
#     print(word)
#     print(infinitive)
#     print(define_not_commot_part_of_words(word, infinitive)[0])
#     print(define_not_commot_part_of_words(word, infinitive)[1])
#     print(define_common_part_of_words(word, infinitive))
#     time.sleep(1)
#    print('\n')
    return feature

import time
def define_suffix(w):
    for s in set_of_suffixes:
        if w.find(s) != -1:
#             print(w)
#             print(s)
#             print(w.find(s))
#             print(len(s))
#             print(len(w))
#             time.sleep(1)
            if ( (w.find(s) + len(s)) == (len(w) - 1)):
                 return s

    return ''

def get_word_without_suffix(w, s):
    if s == '':
        return w
    else:
        i = w.index(s)
        return w[:i]
```

In [29]:

```
suffixes = [(define_not_commot_part_of_words(word, infinitive), word_form) for (word, infir
set_of_suffixes = set([s[0][0] for s in suffixes])
set_of_suffixes = [s for s in set_of_suffixes]
set_of_suffixes.sort(key = lambda s: -len(s))
```

In [30]:

```
features_for_defining_word_form = [(define_set_of_features(word, infinitive), word_form)
                       for (word, infinitive, word_form) in train_set]
features_for_defining_infinitive = [(define_set_of_features(word, infinitive), define_not_c
                    for (word, infinitive, word_form) in train_set]
```

In [31]:

```
print(set_of_suffixes)
print(len(set_of_suffixes))
```

```
['siederebbero', 'i-leninismi', 'e-ordinanze', 'siedereste', 'ocerebbero',
 'siederesti', 'siederebbe', 'sistettero', 'ipattuglia', 'siederanno', 'here
bbero', 'nerebbero', 'cotessero', 'cerebbero', 'siederemo', 'siedevate', 'si
edevamo', 'siederete', 'ini-radar', 'e-partito', 'siedevano', 'ocereste', 'o
ceresti', 'e-lavoro', 'maggiori', 'cotevamo', 'maggiore', 'siederai', 'ocess
imo', 'ocerebbe', 'i-quadro', 'herebber', 'siediamo', 'siederei', 'cotevan
o', 'i-chiave', 'oceranno', 'sistette', 'cotevate', 'sarebber', 'oceremmo',
 'rrebbero', 'migliori', 'i-italia', 'erebbero', 'nerebber', 'cerebber', 'e-
estati', 'hereste', 'neremmo', 'heresti', 'rrebber', 'cereste', 'ceresti',
 'nessero', 'cotemmo', 'ocesser', 'heforti', 'nereste', 'cotesse', 'oceret
e', 'cotessi', 'siedano', 'neresti', 'rebbero', 'siedete', 'nessimo', 'siede
va', 'heremmo', 'i-stati', 'i-paese', 'igruppo', 'ceranno', 'ocevate', 'ocev
amo', 'cotiate', 'coteste', 'cotesti', 'cessimo', 'oceremo', 'cotiamo', 'her
ebbe', 'heranno', '-chiave', 'cerebbe', 'he-dati', 'sarebbe', 'erebber', 'sa
ranno', 'siedevi', 'cessero', 'ceremmo', 'ocevano', 'herete', 'cerete', 'nev
ate', 'cotete', 'ereste', 'nevamo', 'cciamo', 'essimo', 'neremo', 'ocendo',
 'ocevan', 'esorti', 'erebbe', 'fosser', 'eranno', 'escano', 'essero', 'oces
ti', 'oceste', 'ociamo', 'cciano', 'ossero', 'ravamo', 'rreste', 'rresti',
 'uppero', 'cevate', 'eforti', 'furono', 'ocemmo', 'edonne', 'uoiono', 'i-sp
ia', 'nevano', 'gliate', 'e-paga', 'ceremo', 'escono', 'nerono', 'eresti',
 'iedevo', 'heremo', 'cotevi', 'cciate', 'nesser', 'cerono', 'ocer\xc3\xa0',
'cesser', 'cevano', 'uoiano', 'rremmo', 'ssiate', 'ocerei', 'rranno', 'cevam
o', 'ocer\xc3\xb2', 'ociate', 'eremmo', 'ravate', 'gliano', 'gliono', 'rebbe
r', 'imorte', 'rrebbe', 'ocerai', 'oceran', 'iedono', 'ocesse', 'ilista', 'c
otevo', 'sarete', 'quero', 'rebbe', 'sarei', 'osser', 'iedon', 'ceron', 'upp
er', 'nesti', 'neste', 'niamo', 'resti', 'scano', 'reste', 'ceran', 'cerai',
'sarai', 'furon', 'siate', 'ciuta', 'cevan', 'heran', 'herai', 'nemmo', 'ner
\xc3\xa0', 'cesse', 'cessi', 'erete', 'cerei', 'venti', 'idero', 'ccian', 'v
ente', 'cendo', 'evamo', 'ocete', 'isero', 'uoian', 'ttero', 'ranno', 'erem
o', 'rrete', 'ocevi', 'rremo', 'ciute', 'ciuto', 'ciuti', 'ggano', 'sar\xc3
\xb2', 'evano', 'ssono', 'ggono', 'saran', 'ssimo', 'fosti', 'cer\xc3\xa0',
 'ngono', 'scono', 'cer\xc3\xb2', 'nente', 'nenti', 'lsero', 'ngano', 'sser
o', 'nesse', 'hiate', 'fosse', 'herei', 'stata', 'ciate', 'hiamo', 'nerei',
 'state', 'her\xc3\xa0', 'niate', 'her\xc3\xb2', 'ciano', 'siete', 'fummo',
 'cemmo', 'sar\xc3\xa0', 'e-gol', 'siamo', 'ner\xc3\xb2', 'escon', 'centi',
 'cente', 'nerai', 'evate', 'foste', 'nessi', 'glian', 'ceste', 'cesti', 'lg
ano', 'lgono', 'remmo', 'esser', 'oceva', 'iclan', 'neron', 'iviri', 'ocev
o', 'ecero', 'ciono', 'ciamo', 'nendo', 'usero', 'nevan', 'escan', 'bbero',
 'cian', 'dero', 'iamo', 'iedo', 'iedi', 'iede', 'vero', 'ieda', 'cion', 'en
te', 'enti', 'ss\xc3\xa8', 'gano', 'lero', 'iano', 'erai', 'etti', 'etto',
 'scan', 'etta', 'ette', 'ngan', 'iono', 'sser', 'lgan', 'nuti', 'nute', 'ie
ni', 'iene', 'nete', 'scon', 'iser', 'otto', 'otta', 'erto', 'erte', 'uori',
'erta', 'cete', 'ider', 'ggon', 'ssan', 'er\xc3\xb2', 'ccio', 'uppe', 'er\xc
3\xa0', 'uppi', 'rr\xc3\xa0', 'ecer', 'rr\xc3\xb2', 'sian', 'sson', 'sero',
 'este', 'esti', 'bber', 'essa', 'esse', 'essi', 'esso', 'osso', 'quer', 'os
si', 'rici', 'osse', 'rice', 'ossa', 'iate', 'erei', 'endo', 'iuto', 'iuti',
'iute', 'iuta', 'esca', 'reta', 'reto', 'lgon', 'vano', 'remo', 'eran', 'can
o', 'cono', 'otti', 'esci', 'rrei', 'otte', 'ccia', 'user', 'ls\xc3\xa8', 'v
amo', 'ngon', 'ggan', 'hino', 'emmo', 'lser', 'uoio', 'rrai', 'rran', 'nev
a', 'vate', 'nevo', 'nevi', 'anno', 'rete', 'ceva', 'erti', 'cevi', 'cevo',
 'glia', 'esce', 'gono', 'esco', 'nero', 'evan', 'tter', 'sco', 'sci', 'sc
e', 'sca', 'ver', 'evi', 'eva', 'der', 'nei', 'ner', 'nes', 'gga', 'ggo', 'q
ui', 'use', 'usa', 'uso', 'usi', 'can', 'lta', 'n\xc3\xa8', 'ito', 'iti', 'i
te', 'lsi', 'tti', 'tto', 'tta', 'tte', 'cei', 'ces', 'sii', 'evo', 'gon',
 'rei', 'ece', 'eci', 'nti', 'mmo', 'que', 'nte', 'ate', 'isi', 'lte', 'lt
i', 'lto', 'ise', 'ono', 'ian', 'ste', 'sta', 'sto', 'sti', 'ran', 'ano', 'r
ai', '\xc3\xa8i', 'nga', 'ies', 'ngo', 'sso', 'ssi', 'sse', 'ita', 'ssa', 'n
in', 'ser', 'sei', 'uoi', 'u\xc3\xb2', 'bbi', 'bbe', 'lgo', 'lga', 'gan', 'f
ui', 'ete', 'r\xc3\xb2', 'ion', 'r\xc3\xa0', 'con', 's\xc3\xa8', 'amo', 'nd
o', 'idi', 'ide', 'son', 'van', 'cia', 'uta', 'ute', 'uti', 'cio', 'uto', 'n
no', 'ino', 'ini', 'c\xc3\xa8', 'ga', 'go', 'to', 'ti', 'ta', 'vi', 'te', 'm
o', 'fu', 'so', 'sa', 'se', 'en', 'ei', 'ro', 'ri', '\xc3\xa8', '\xc3\xac',
 '\xc3\xa0', '\xc3\xb2', 'di', 'de', 'si', 'le', 'ci', 'co', 'ca', 'ce', 'l
i', 'va', 've', 'vo', 'io', 'in', 'ia', 'on', 'hi', 'he', 'ai', 'an', 'ni',
 'no', 'ne', 'a', 's', 'n', 'i', 'o', 'e', '']
538
```

In [32]:

```
# import numpy as np
# a_res = []
# a_tar1 = []
# a_tar2 = []
# for x in features_for_defining_word_form:
#     a_res.append([ x[0]['suffix'], x[0]['last'], x[0]['prelast'] ])
#     a_tar1.append(x[1])

# for x in features_for_defining_infinitive:
#     a_tar2.append(x[1])

# a_res = np.array(a_res)
# a_tar1 = np.array(a_tar1)
# a_tar2 = np.array(a_tar2)
# print (type(a_res))
# print(a_res.shape)
```

In [33]:

```
from sklearn.model_selection import cross_val_score
form_classifier = nltk.NaiveBayesClassifier.train(features_for_defining_word_form)
infinitive_classifier = nltk.NaiveBayesClassifier.train(features_for_defining_infinitive)
from sklearn.linear_model import LogisticRegression

print(nltk.classify.accuracy(form_classifier, features_for_defining_word_form))
print(nltk.classify.accuracy(infinitive_classifier, features_for_defining_infinitive))
form_classifier.show_most_informative_features()
infinitive_classifier.show_most_informative_features()

#algo1 = LogisticRegression(penalty='l2', C=0.5)
#algo1.fit(a_res, a_tar1)

#algo2 = LogisticRegression(penalty='l2', C=0.5)
#algo2.fit(a_res, a_tar2)

#print cross_val_score(form_classifier, matrix, ys, scoring="roc_auc").mean()
#print cross_val_score(infinitive_classifier, matrix, ys, scoring="roc_auc").mean()

import pandas as pd
from sklearn.model_selection import cross_val_score
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.linear_model import LogisticRegression

# def lower_text(text):
#     lowered_text = text.lower()
#     return lowered_text

# data_frame = pd.read_csv("task2_lemmas_train", sep=",", names=['X', 'y'])
# #data_frame["X"] = data_frame.X.apply(lower_text)
# Xs = data_frame.X
# ys = data_frame.y
# #ys = data_frame.y.split('+')[0]
```

```
0.874881995954
0.847800067431
Most Informative Features
                    last = '\xb2'              V : N      =    225.3 : 1.0
                  suffix = 'a'                 A : N      =    221.5 : 1.0
                  suffix = 'e'                 N : V      =    145.7 : 1.0
                    last = 'r'                 V : A      =    143.9 : 1.0
                 prelast = '\xc3'              V : A      =    138.2 : 1.0
                    last = 'n'                 V : A      =    107.1 : 1.0
                  suffix = 'vo'                V : A      =    102.9 : 1.0
                 prelast = 'k'                 N : V      =     65.8 : 1.0
                 prelast = 'a'                 V : A      =     63.0 : 1.0
                  suffix = 'mmo'               V :        =     39.5 : 1.0
Most Informative Features
                  suffix = 'a'                 o : re     =   8034.1 : 1.0
                  suffix = 'ono'               e : re     =   6049.9 : 1.0
                  suffix = 'on'                e : re     =   6009.3 : 1.0
                    last = 'a'                 o : are    =   5057.9 : 1.0
                  suffix = 'ano'             ere : re     =   2466.4 : 1.0
                  suffix = 'an'              ere : re     =   2306.8 : 1.0
                  suffix = 'ente'            ire : re     =   2292.9 : 1.0
                 prelast = 'o'                 e : are    =   2259.1 : 1.0
                  suffix = 'enti'            ire : re     =   2136.5 : 1.0
```

```
            prelast = 'c'                ore : a       =    972.2 : 1.0
```

In [ ]:

```

```

In [34]:

```
test_data = open('task2_lemmas_test')
test_set = []
i = 0
for line in test_data:
    if i != 0:
        b = line.split(',')
        word = b[1]
        test_set.append(word)
    i += 1
print(len(test_set))
```

```
29661
```

In [35]:

```
test_suffixes = [{'suffix': define_suffix(w)} for w in test_set]
test_infinitive = [get_word_without_suffix(w, define_suffix(w)) for w in test_set]
```

In [36]:

```
print(test_suffixes)
print(test_infinitive)
```

```
ix': '\xc3\xb2'}, {'suffix': 'le'}, {'suffix': 'ide'}, {'suffix': 'ssim
o'}, {'suffix': 'vi'}, {'suffix': 'ino'}, {'suffix': 'ai'}, {'suffix': 'es
si'}, {'suffix': 'ini'}, {'suffix': 'erebber'}, {'suffix': 'heremo'}, {'su
ffix': '\xc3\xb2'}, {'suffix': 'an'}, {'suffix': 'ver'}, {'suffix': 'va
n'}, {'suffix': 'ono'}, {'suffix': 'in'}, {'suffix': 'scon'}, {'suffix':
 'reste'}, {'suffix': 'ce'}, {'suffix': 'eran'}, {'suffix': 'ano'}, {'suff
ix': 'ndo'}, {'suffix': 'er\xc3\xa0'}, {'suffix': 'ano'}, {'suffix': 'l
e'}, {'suffix': 'der'}, {'suffix': 'on'}, {'suffix': 'sser'}, {'suffix':
 'eresti'}, {'suffix': 'vamo'}, {'suffix': 'erebber'}, {'suffix': 'ca'},
 {'suffix': 'mmo'}, {'suffix': 'ono'}, {'suffix': 'endo'}, {'suffix':
 'e'}, {'suffix': 'ssero'}, {'suffix': 'eremo'}, {'suffix': 'li'}, {'suffi
x': ''}, {'suffix': 'er\xc3\xb2'}, {'suffix': 'er\xc3\xb2'}, {'suffix': 's
simo'}, {'suffix': 'ste'}, {'suffix': 'vo'}, {'suffix': 'ano'}, {'suffix':
'nerei'}, {'suffix': 'rei'}, {'suffix': 'le'}, {'suffix': 'sse'}, {'suffi
x': 'mo'}, {'suffix': 'rici'}, {'suffix': 'ssimo'}, {'suffix': 'her\xc3\xa
0'}, {'suffix': 'eremo'}, {'suffix': 'van'}, {'suffix': 'iti'}, {'suffix':
'iate'}, {'suffix': 'cerai'}, {'suffix': 'eremo'}, {'suffix': 'eran'}, {'s
uffix': 'eremo'}, {'suffix': 'hi'}, {'suffix': 'hi'}, {'suffix': 'nte'},
 {'suffix': 'va'}, {'suffix': 'an'}, {'suffix': 'ste'}, {'suffix': 'an'},
 {'suffix': 'ate'}, {'suffix': 's'}, {'suffix': 'ssi'}, {'suffix': 'scon
```

In [37]:

```
preds_suffix = [infinitive_classifier.classify(s) for s in test_suffixes]
preds_class = [form_classifier.classify(s) for s in test_suffixes]



#preds_suffix = [algo2.predict(suffix) for suffix in test_suffixes]
#preds_class = [algo1.predict(suffix) for suffix in test_suffixes]
```

In [ ]:

```

```

In [ ]:

```

```

In [38]:

```
answer = []
for i in range(len(preds_suffix)):
    a = test_infinitive[i].rstrip() + preds_suffix[i].rstrip() + '+' + preds_class[i].rstri
    answer.append(a)
```

In [39]:

```
answer_ = pd.read_csv('task2_lemmas_sample_submission')
answer_['Category'] = answer
answer_.to_csv('res.txt', sep=',',index=False)
print(answer_[:4])
```

```
   Id        Category
0   1       gettone+V
1   2  incidentaere+V
2   3     involtare+V
3   4         lirre+V
```

In [ ]:

```

```

In [ ]:

```

```