In [1]:
```
import nltk,random
#function returning the last letter of a word
def gender_features(word):
    return {'last_letter': word[-1]}
```

In [86]:
```
gender_features('Shrek')
```

Out[86]:
```
{'last_letter': 'k'}
```

In [87]:
```
from nltk.corpus import names# set of male and female names:how to do our own sets?
labeled_names = ([(name, 'male') for name in names.words('male.txt')] + [(name, 'female') for name in names.words('female.txt')]
```

In [88]:
```
from sklearn.model_selection import train_test_split
featuresets = [(gender_features(n), gender) for (n, gender) in labeled_names]
train_set, test_set = train_test_split(featuresets, shuffle=True,test_size=.2)
print(len(train_set))
print(len(test_set))
```

```
6355
1589
```

In [89]:
```
classifier = nltk.NaiveBayesClassifier.train(train_set)# we need other steps like feauture selection etc...
print(nltk.classify.accuracy(classifier, test_set))
```

```
0.7595972309628697
```

In [90]:
```
print(classifier.classify(gender_features('Alan')))
print(classifier.classify(gender_features('Stacey')))
```

```
male
female
```

In [91]:
```
classifier.show_most_informative_features(5)
```

```
Most Informative Features
             last_letter = 'k'              male : female =     38.0 : 1.0
             last_letter = 'a'            female : male   =     35.8 : 1.0
             last_letter = 'f'              male : female =     19.6 : 1.0
             last_letter = 'v'              male : female =     16.2 : 1.0
             last_letter = 'd'              male : female =      9.9 : 1.0
```

In [11]:
```
from nltk.classify import apply_features
train_set, test_set = train_test_split(labeled_names, shuffle=True,test_size=.2)
train_set = apply_features(gender_features, train_set)
test_set = apply_features(gender_features, test_set)
```

1.2. Choosing the Right Features

In [94]:
```
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 [95]:

```
gender_features2('Walter')
```

Out[95]:

```
{'count(a)': 1,
 'count(b)': 0,
 'count(c)': 0,
 'count(d)': 0,
 'count(e)': 1,
 'count(f)': 0,
 'count(g)': 0,
 'count(h)': 0,
 'count(i)': 0,
 'count(j)': 0,
 'count(k)': 0,
 'count(l)': 1,
 'count(m)': 0,
 'count(n)': 0,
 'count(o)': 0,
 'count(p)': 0,
 'count(q)': 0,
 'count(r)': 1,
 'count(s)': 0,
 'count(t)': 1,
 'count(u)': 0,
 'count(v)': 0,
 'count(w)': 1,
 'count(x)': 0,
 'count(y)': 0,
 'count(z)': 0,
 'first_letter': 'w',
 'has(a)': True,
 'has(b)': False,
 'has(c)': False,
 'has(d)': False,
 'has(e)': True,
 'has(f)': False,
 'has(g)': False,
 'has(h)': False,
 'has(i)': False,
 'has(j)': False,
 'has(k)': False,
 'has(l)': True,
 'has(m)': False,
 'has(n)': False,
 'has(o)': False,
 'has(p)': False,
 'has(q)': False,
 'has(r)': True,
 'has(s)': False,
 'has(t)': True,
 'has(u)': False,
 'has(v)': False,
 'has(w)': True,
 'has(x)': False,
 'has(y)': False,
 'has(z)': False,
 'last_letter': 'r'}
```

In [96]:

```
featuresets = [(gender_features2(n), gender) for (n, gender) in labeled_names]
train_set, test_set = train_test_split(featuresets, shuffle=True,test_size=.2)
classifier = nltk.NaiveBayesClassifier.train(train_set)
print(nltk.classify.accuracy(classifier, test_set))
```

```
0.7627438640654499
```

In [97]:

```
train_names, test_names = train_test_split(labeled_names, shuffle=True,test_size=.3)
devtest_names, test_names = train_test_split(test_names, shuffle=True,test_size=.3333)
```

In [98]:

```
[print(len(s)) for s in [train_names,devtest_names,test_names]]
```

```
5560
1589
795
```

Out[98]:

```
[None, None, None]
```

In [99]:

```
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))
```

```
0.7602265575833858
```

In [100]:

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

In [101]:

```
for (tag, guess, name) in sorted(errors):
    print('correct={:<8} guess={:<8s} name={:<30}'.format(tag, guess, name))
```

```
correct=female   guess=male     name=Adriaens
correct=female   guess=male     name=Aigneis
correct=female   guess=male     name=Alex
correct=female   guess=male     name=Alexis
correct=female   guess=male     name=Alis
correct=female   guess=male     name=Alisun
correct=female   guess=male     name=Alleen
correct=female   guess=male     name=Allyson
correct=female   guess=male     name=Alyson
correct=female   guess=male     name=Anne-Mar
correct=female   guess=male     name=Ashlen
correct=female   guess=male     name=Betteann
correct=female   guess=male     name=Bev
correct=female   guess=male     name=Blair
correct=female   guess=male     name=Bo
correct=female   guess=male     name=Brear
correct=female   guess=male     name=Britt
correct=female   guess=male     name=Brittan
correct=female   guess=male     name=Brooks
```

In [103]:
```
def gender_features(word):
     return {'suffix1': word[-1:],
             'suffix2': word[-2:]}
```

In [104]:
```
train_set = [(gender_features(n), gender) for (n, gender) in train_names]
devtest_set = [(gender_features(n), gender) for (n, gender) in devtest_names]
classifier = nltk.NaiveBayesClassifier.train(train_set)
print(nltk.classify.accuracy(classifier, devtest_set))
#we need more on feature selection, chisquare etc...accuracy chart
```

```
0.7853996224040277
```

1.3 Document Classification

In [108]:
```
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)
```

In [110]:
```
all_words = nltk.FreqDist(w.lower() for w in movie_reviews.words())
word_features = list(all_words)[:2000]
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 [111]:
```
featuresets = [(document_features(d), c) for (d,c) in documents]
train_set, test_set = train_test_split(featuresets, shuffle=True,test_size=.2)
# train_set, test_set = featuresets[100:], featuresets[:100]
classifier = nltk.NaiveBayesClassifier.train(train_set)
```

In [116]:
```
print(nltk.classify.accuracy(classifier, test_set))
classifier.show_most_informative_features(10)
```

```
0.6725
Most Informative Features
        contains(hudson) = True              neg : pos    =      7.8 : 1.0
            contains(et) = True              pos : neg    =      7.6 : 1.0
          contains(taxi) = True              pos : neg    =      7.6 : 1.0
     contains(stupidity) = True              neg : pos    =      7.4 : 1.0
        contains(debate) = True              pos : neg    =      7.3 : 1.0
           contains(ivy) = True              neg : pos    =      6.4 : 1.0
  contains(confidential) = True              pos : neg    =      6.3 : 1.0
         contains(whore) = True              neg : pos    =      5.7 : 1.0
          contains(lang) = True              pos : neg    =      5.6 : 1.0
        contains(freely) = True              pos : neg    =      5.6 : 1.0
```

1.4 Part-of-Speech Tagging

In [3]:
```
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
```

In [4]:
```
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', 'is', 'on', 'l', 'g', 'and', 'ng', 'er', 'as', 'ing', 'h', 'at', 'es', 'or', 'r
e', 'it', '``', 'an', "''", 'm', ';', 'i', 'ly', 'ion', 'en', 'al', '?', 'nt', 'be', 'hat', 'st', 'his', 'th', 'll', 'le', 'ce', 'by', 'ts', 'me', 've', "'", 'se', 'ut', 'was', 'for',
'ent', 'ch', 'k', 'w', 'ld', '`', 'rs', 'ted', 'ere', 'her', 'ne', 'ns', 'ith', 'ad', 'ry', ')', '(', 'te', '--', 'ay', 'ty', 'ot', 'p', 'nce', "'s", 'ter', 'om', 'ss', ':', 'we', 'ar
e', 'c', 'ers', 'uld', 'had', 'so', 'ey']
```

In [9]:
```
def pos_features(word):
     features = {}
     for suffix in common_suffixes:
         features['endswith({})'.format(suffix)] = word.lower().endswith(suffix)
     return features
```

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

In [7]:
```
# size = int(len(featuresets) * 0.1)
# train_set, test_set = featuresets[size:], featuresets[:size]
train_set, test_set = train_test_split(featuresets, shuffle=False,test_size=.1)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-7-12cc5a8dda35> in <module>()
      1 # size = int(len(featuresets) * 0.1)
      2 # train_set, test_set = featuresets[size:], featuresets[:size]
----> 3 train_set, test_set = train_test_split(featuresets, shuffle=False,test_size=.1)

NameError: name 'train_test_split' is not defined
```

```
In [124]: classifier = nltk.DecisionTreeClassifier.train(train_set)
          nltk.classify.accuracy(classifier, test_set)
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-124-0c836654950f> in <module>()
----> 1 classifier = nltk.DecisionTreeClassifier.train(train_set)
      2 nltk.classify.accuracy(classifier, test_set)

/usr/local/lib/python3.5/dist-packages/nltk/classify/decisiontree.py in train(labeled_featuresets, entropy_cutoff, depth_cutoff, support_cutoff, binary, feature_values, verbose)
    159         # Refine the stump.
    160         tree.refine(labeled_featuresets, entropy_cutoff, depth_cutoff-1,
--> 161                     support_cutoff, binary, feature_values, verbose)
    162 
    163         # Return it

/usr/local/lib/python3.5/dist-packages/nltk/classify/decisiontree.py in refine(self, labeled_featuresets, entropy_cutoff, depth_cutoff, support_cutoff, binary, feature_values, verbo
se)
    201                 self._decisions[fval] = DecisionTreeClassifier.train(
    202                     fval_featuresets, entropy_cutoff, depth_cutoff,
--> 203                     support_cutoff, binary, feature_values, verbose)
    204         if self._default is not None:
```

```
In [ ]: classifier.classify(pos_features('dog'))
```

```
In [34]: print(classifier.pseudocode(depth=4))
```

```
if endswith(the) == False:
  if endswith(,) == False:
    if endswith(s) == False:
      if endswith(.) == False: return 'NP-HL'
      if endswith(.) == True: return '.'
    if endswith(s) == True:
      if endswith(was) == False: return 'NNS'
      if endswith(was) == True: return 'BEDZ'
  if endswith(,) == True: return ','
if endswith(the) == True: return 'AT'
```

1.5 Exploiting Context

```
In [36]: 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
```

```
In [37]: pos_features(brown.sents()[0], 8)
```

```
Out[37]: {'prev-word': 'an', 'suffix(1)': 'n', 'suffix(2)': 'on', 'suffix(3)': 'ion'}
```

```
In [38]: 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) )
```

```
In [39]: # size = int(len(featuresets) * 0.1)
         # train_set, test_set = featuresets[size:], featuresets[:size]
         train_set, test_set = train_test_split(featuresets, shuffle=False,test_size=.1)
         classifier = nltk.NaiveBayesClassifier.train(train_set)
         nltk.classify.accuracy(classifier, test_set)
```

```
Out[39]: 0.7771479713603818
```

```
In [40]: 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)
```

In [41]:

```
tagged_sents = brown.tagged_sents(categories='news')
# size = int(len(tagged_sents) * 0.1)
# train_sents, test_sents = tagged_sents[size:], tagged_sents[:size]
train_sents, test_sents = train_test_split(tagged_sents, shuffle=False,test_size=.1)
tagger = ConsecutivePosTagger(train_sents)
print(tagger.evaluate(test_sents))
0.79796012981
```

```
0.7870028904614771
```

Out[41]:

```
0.79796012981
```

In [ ]:

```
```