HOMEWORK 6

By- Syed Zeeshan Ali [alisz@mail.uc.edu](mailto:alisz@mail.uc.edu)

2.  Using any of the three classifiers 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. How does the performance on the test set compare to the performance on the dev-test set? Is this what you'd expect?

#2

import nltk

#import nltk

#Define function to return last letter of the word as a feature

def gender\_features1(word):

return {'last\_letter': word[-1]}

#Import names

from nltk.corpus import names

#get names labelled as male and female

labeled\_names = ([(name, 'male') for name in names.words('male.txt')] +[(name, 'female') for name in names.words('female.txt')])

import random #random shuffle

random.shuffle(labeled\_names)

#training name 1001 till last

train\_names = labeled\_names[1000:]

#Dev test second 500 names ranging from 501 to 1000 names

devtest\_names = labeled\_names[500:1000]

#test names first 500 names

test\_names = labeled\_names[:500]

#Get train\_set which has feature and gender for each name

train\_set = [(gender\_features1(n), gender) for (n, gender) in train\_names]

#Get devtest\_set which has feature and gender for each name

devtest\_set = [(gender\_features1(n), gender) for (n, gender) in devtest\_names]

#Get test\_set which has feature and gender for each name

test\_set = [(gender\_features1(n), gender) for (n, gender) in test\_names]

#Train NaiveBayes classifier on train\_set

classifier = nltk.NaiveBayesClassifier.train(train\_set)

#print accuracy on dev\_test

print(nltk.classify.accuracy(classifier, devtest\_set))

#Define a function which uses last two letters that is it has more features than the last function

def gender\_features2(word):

return {'suffix1': word[-1:],'suffix2': word[-2:]}

#Divide train,dev test and test set same as above but using gender\_features2 function

train\_set = [(gender\_features2(n), gender) for (n, gender) in train\_names]

devtest\_set = [(gender\_features2(n), gender) for (n, gender) in devtest\_names]

test\_set = [(gender\_features2(n), gender) for (n, gender) in test\_names]

#training naivebayes classifier on the train set

classifier = nltk.NaiveBayesClassifier.train(train\_set)

#print accuracy on dev test

print(nltk.classify.accuracy(classifier, devtest\_set))

#Increase in classification accuracy

print(nltk.classify.accuracy(classifier, test\_set))

**OUTPUT:**

Accuracy of the classifier on dev\_test with gender\_features1 function which returns last letter



Accuracy of the classifier on dev\_test with gender\_features2 function which returns last 2 letter

Since the accuracy has improved on dev\_test we will now test the accuracy on test\_set



3. The Senseval 2 Corpus contains data intended to train word-sense disambiguation classifiers. It contains data for four words: hard, interest, line, and serve. Choose one of these four words, and load the corresponding data:

Using this dataset, build a classifier that predicts the correct sense tag for a given instance. See the corpus HOWTO at http://nltk.org/howto for information on using the instance objects returned by the Senseval 2 Corpus.

#3

#importing senseval package

from nltk.corpus import senseval

#getting instance of interest.pos, senseval has four different instance

instances = senseval.instances('interest.pos')

#getting 10% of the instances in size variable

size = int(len(instances) \* 0.1)

#Using size variable first 10% i.e. 236 in train\_set and rest of the 90% in train\_set

train\_set, test\_set = instances[size:], instances[:size]

#train naivebayes on train\_set

classifier = nltk.NaiveBayesClassifier.train(train\_set)

#Defining a function to return sense feature

def sense\_features(left,word,right):

return {'prefix': left[-1:]}

#Since senseval objects are not iterateable directly

#We will use below method to iterate on it and create training and then testing set

train=[] #declare train variable

#For each train\_Set

for inst in train\_set:

p = inst.position

left = ' '.join(w for (w,t) in inst.context[p-2:p])

word = ' '.join(w for (w,t) in inst.context[p:p+1])

right = ' '.join(w for (w,t) in inst.context[p+1:p+3])

senses = ' '.join(inst.senses)

#print((left, word, right, senses))

l=sense\_features(left,word,right)

train.append((l,senses)) #append left last letter and sense to train

#Training NaiveBayes Classifier on train

classifier = nltk.NaiveBayesClassifier.train(train)

test=[] #Declare test variable

#Iterating over senseval test\_set to get left word and right

for inst in test\_set:

p = inst.position

left = ' '.join(w for (w,t) in inst.context[p-2:p])

word = ' '.join(w for (w,t) in inst.context[p:p+1])

right = ' '.join(w for (w,t) in inst.context[p+1:p+3])

senses = ' '.join(inst.senses)

#print((left, word, right, senses))

l=sense\_features(left,word,right)

test.append((l,senses)) #append left last letter and sense to test

#accuracy of the classifier on test set

print(nltk.classify.accuracy(classifier, test))

**OUTPUT:**



4. Using the movie review document classifier discussed in this chapter, generate a list of the 30 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?

#4

#import movie\_reviews

from nltk.corpus import movie\_reviews

#get category and fileid in documents

documents = [(list(movie\_reviews.words(fileid)), category)

for category in movie\_reviews.categories()

for fileid in movie\_reviews.fileids(category)]

random.shuffle(documents) #random shuffle

#Frequency distribution of movie review words

all\_words = nltk.FreqDist(w.lower() for w in movie\_reviews.words())

#first 2000 in word\_features variable

word\_features = list(all\_words)[:2000]

#function to return contained words as feature

def document\_features(document):

document\_words = set(document)

features = {}

for word in word\_features:

features['contains({})'.format(word)] = (word in document\_words)

return features

#getting document features in a feature set variable

featuresets = [(document\_features(d), c) for (d,c) in documents]

#training and testing split

train\_set, test\_set = featuresets[100:], featuresets[:100]

#Train naive bayes classifier on train\_set

classifier = nltk.NaiveBayesClassifier.train(train\_set)

#print the accuracy on test\_set

print(nltk.classify.accuracy(classifier, test\_set))

#30 most informative features

classifier.show\_most\_informative\_features(30)

**OUTPUT:**

Surprising thing is negative ratio seems to be more than positive in the below output

classifier.show\_most\_informative\_features(30)

Most Informative Features

contains(schumacher) = True neg : pos = 12.2 : 1.0

contains(justin) = True neg : pos = 9.6 : 1.0

contains(welles) = True neg : pos = 8.3 : 1.0

contains(unimaginative) = True neg : pos = 7.6 : 1.0

contains(sexist) = True neg : pos = 7.6 : 1.0

contains(mena) = True neg : pos = 6.9 : 1.0

contains(suvari) = True neg : pos = 6.9 : 1.0

contains(shoddy) = True neg : pos = 6.9 : 1.0

contains(turkey) = True neg : pos = 6.7 : 1.0

contains(atrocious) = True neg : pos = 6.5 : 1.0

contains(singers) = True pos : neg = 6.4 : 1.0

contains(awful) = True neg : pos = 6.2 : 1.0

contains(ridiculous) = True neg : pos = 6.0 : 1.0

contains(unravel) = True pos : neg = 5.7 : 1.0

contains(bronson) = True neg : pos = 5.6 : 1.0

contains(poorly) = True neg : pos = 5.5 : 1.0

contains(ugh) = True neg : pos = 5.4 : 1.0

contains(wasted) = True neg : pos = 5.2 : 1.0

contains(surveillance) = True neg : pos = 5.0 : 1.0

contains(underwood) = True neg : pos = 5.0 : 1.0

contains(oops) = True neg : pos = 5.0 : 1.0

contains(kudos) = True pos : neg = 4.8 : 1.0

contains(miscast) = True neg : pos = 4.7 : 1.0

contains(waste) = True neg : pos = 4.7 : 1.0

contains(everyday) = True pos : neg = 4.7 : 1.0

contains(explores) = True pos : neg = 4.6 : 1.0

contains(groan) = True neg : pos = 4.6 : 1.0

contains(martian) = True neg : pos = 4.6 : 1.0

contains(bland) = True neg : pos = 4.4 : 1.0

contains(inject) = True neg : pos = 4.4 : 1.0

5. 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. How do you think that your results might be different if you used a different feature extractor?

#5

#Training data , dev test and testing data

train\_names = labeled\_names[1000:]

devtest\_names = labeled\_names[500:1000]

test\_names = labeled\_names[:500]

#Defining gender features function

def gender\_features(word):

return {'suffix1': word[-1:],'suffix2': word[-2:]}

#Spltting in training, dev test and test set

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]

#Naive Bayes CLassifier training

classifier = nltk.NaiveBayesClassifier.train(train\_set)

#Decision Tree Training

classifier2 = nltk.DecisionTreeClassifier.train(train\_set)

#Maximum Entropy Training

classifier3=nltk.classify.MaxentClassifier.train(train\_set)

#Naive Baise

print("Naive Bayes")

print("dev\_test",nltk.classify.accuracy(classifier, devtest\_set))

print("test",nltk.classify.accuracy(classifier, test\_set))

print("==================")

#Decision Tree

print("Decision Tree")

print("dev\_test",nltk.classify.accuracy(classifier2, devtest\_set))

print("test",nltk.classify.accuracy(classifier2, test\_set))

print("==================")

#Maximum Entropy

print("Max Entropy")

print("dev\_test",nltk.classify.accuracy(classifier3, test\_set))

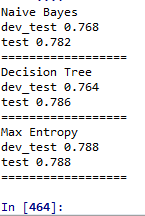
print("test",nltk.classify.accuracy(classifier3, test\_set))

print("==================")

**OUTPUT**:

Max Entropy is winning as of now in the three classifier but if we changed the feature extractor such that all test data is already covered by the training data entropy will be low in that case. Because

0 <Entropy< 1, 0 if no new words, so the higher no, of different words in test set the higher the entropy will be.



6. The synonyms *strong* and *powerful* pattern differently (try combining them with *chip* and *sales*). What features are relevant in this distinction? Build a classifier that predicts when each word should be used.

#6

import nltk #import nltk

#returninga feature

def d\_feature(w):

return {'left': w[-1:]}

#Train\_words on correct usage

train\_words=['strong sales','powerful chip']

#All combinations of these four

words\_all=['strong chip','strong sales','powerful chip','powerful sales']

train\_words1=[] #Train words variable

#Preprocessing data before we can use it for feaure extraction

for line in train\_words:

f1,f2=line.split(' ')

train\_words1.append((f1,f2))

all\_words1=[] #All words variable

#Preprocessing data before we can use it for feaure extraction

for line in words\_all:

f1,f2=line.split(' ')

all\_words1.append((f1,f2))

#All set having feature and word

all\_set=[(d\_feature(n), wor) for (n, wor) in all\_words1]

#Train set having feature and word

tr\_set=[(d\_feature(n), wor) for (n, wor) in train\_words1]

#Traing classifier

classifier = nltk.NaiveBayesClassifier.train(tr\_set)

#Here it will only classify correct combination that;s why 0.5 accuracy

print(nltk.classify.accuracy(classifier, all\_set))

**OUTPUT:**

|  |  |  |
| --- | --- | --- |
| **FIRST WORD** | **SECOND WORD** | **USAGE** |
| Strong | Sales | **CORRECT** |
| Powerful | Chip | **CORRECT** |
| Powerful | Sales | **WRONG** |
| Strong | Chip | **WRONG** |

We will train our classifier on correct usage so when try to classify all combination we will get 0.5 accuracy, cause our feature set will recognize correct usage but it won’t recognize the incorrect usage.

