CSCI-8450

2018 Spring

Yuchen Wang

You will find Chapter 6 posted at:

<http://www.nltk.org/book/ch06.html>

1. exercise 2.  Design at least 5 features and report what these features capture. Additionally, use three classifiers, namely, nltk.NaiveBayesClassifier, nltk.DecisionTreeClassifier, nltk.MaxentClassifier. Compare the performance of the three classifiers by analyzing the accuracy. Report the accuracy of each classifier built using all of the features that you designed.

☼ 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?

Code:

*# for exercise 2***def** gender\_features(name):  
 features = {}  
 features[**"firstletter"**] = name[0].lower()  
 features[**"lastletter"**] = 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

**def** exercise2():  
 **from** nltk.corpus **import** names  
 names = ([(name, **'male'**) **for** name **in** names.words(**'male.txt'**)] + [(name, **'female'**) **for** name **in** names.words(**'female.txt'**)])  
 random.shuffle(names)  
 test, devtest, training = names[:500],names[500:1000],names[1000:]  
  
 featuresets = [(gender\_features(n), gender) **for** (n, gender) **in** names]  
 train\_set, test\_set = featuresets[:500], featuresets[500:1500]  
  
  
 *# naviebayes* classifier1 = nltk.NaiveBayesClassifier.train(train\_set)  
 print (**"naive bayes vs train\_set "**,nltk.classify.accuracy(classifier1, train\_set))  
 print(**"naive bayes vs test\_set "**,nltk.classify.accuracy(classifier1, test\_set))  
 print (classifier1.show\_most\_informative\_features(5))  
  
 *# decision tree* classifier2 = nltk.DecisionTreeClassifier.train(train\_set)  
 print(**"decision tree vs train\_set "**, nltk.classify.accuracy(classifier2, train\_set))  
 print(**"decision tree vs test\_set "**, nltk.classify.accuracy(classifier2, test\_set))  
  
 *# Maxent* algorithm = nltk.classify.MaxentClassifier.ALGORITHMS[0]  
 classifier3 = nltk.MaxentClassifier.train(train\_set, algorithm, trace=0,max\_iter=5)  
 print(**"maxent vs train\_set "**, nltk.classify.accuracy(classifier3, train\_set))  
 print(**"maxent vs test\_set "**, nltk.classify.accuracy(classifier3, test\_set))

Result:

Exercise 2

naive bayes vs train\_set 0.812

naive bayes vs test\_set 0.734

Most Informative Features

lastletter = 'a' female : male = 44.0 : 1.0

lastletter = 'd' male : female = 10.7 : 1.0

count(w) = 1 male : female = 8.9 : 1.0

has(w) = True male : female = 8.9 : 1.0

lastletter = 'm' male : female = 7.5 : 1.0

None

decision tree vs train\_set 0.966

decision tree vs test\_set 0.745

maxent vs train\_set 0.664

maxent vs test\_set 0.666

1. exercise 4. To report, pick any 5 features out of the computed 30 and describe their relevance.

☼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?

Code:

**def** exercise4():  
 **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)  
  
 all\_words = nltk.FreqDist(w.lower() **for** w **in** movie\_reviews.words())  
 word\_features = list(all\_words.keys())[: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  
  
 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(30)

Result:

Exercise 4

0.78

Most Informative Features

contains(unimaginative) = True neg : pos = 7.8 : 1.0

contains(schumacher) = True neg : pos = 7.5 : 1.0

contains(suvari) = True neg : pos = 7.1 : 1.0

contains(mena) = True neg : pos = 7.1 : 1.0

contains(atrocious) = True neg : pos = 7.1 : 1.0

contains(singers) = True pos : neg = 6.3 : 1.0

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

contains(justin) = True neg : pos = 5.9 : 1.0

contains(poorly) = True neg : pos = 5.8 : 1.0

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

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

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

contains(wasted) = True neg : pos = 5.3 : 1.0

contains(ugh) = True neg : pos = 5.1 : 1.0

contains(welles) = True neg : pos = 5.1 : 1.0

contains(underwood) = True neg : pos = 5.1 : 1.0

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

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

contains(ridiculous) = True neg : pos = 4.9 : 1.0

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

contains(explores) = True pos : neg = 4.5 : 1.0

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

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

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

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

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

contains(unfunny) = True neg : pos = 4.3 : 1.0

contains(seymour) = True pos : neg = 4.3 : 1.0

contains(positions) = True pos : neg = 4.3 : 1.0

contains(sexist) = True neg : pos = 4.3 : 1.0

1. exercise 7. Design at least 5 features and report what these features capture. Report the accuracy of your classifier. Place your classifier code into the report.

◑ The dialog act classifier assigns labels to individual posts, without considering the context in which the post is found. However, dialog acts are highly dependent on context, and some sequences of dialog act are much more likely than others. For example, a ynQuestion dialog act is much more likely to be answered by a yanswer than by a greeting. Make use of this fact to build a consecutive classifier for labeling dialog acts. Be sure to consider what features might be useful. See the code for the consecutive classifier for part-of-speech tags in [1.7](http://www.nltk.org/book/ch06.html#code-consecutive-pos-tagger) to get some ideas.

Code:

*# exercise 7***def** extract\_features(post):  
 features = {}  
 **for** word **in** nltk.word\_tokenize(post.text):  
 features[**'contains({})'**.format(word.lower())] = **True  
 return** features  
**def** fpost\_list(posts):  
 fposts = []  
 **for** p **in** posts:  
 fposts.append((extract\_features(p), p.get(**'class'**)))  
 **return** fposts  
  
**def** pos\_features(sentence, i, history): *# [\_consec-pos-tag-features]* features = {  
 **"suffix(1)"**: sentence[i][-1:],  
 **"suffix(2)"**: sentence[i][-2:],  
 **"suffix(3)"**: sentence[i][-3:],  
 **"suffix(4)"**: sentence[i][-4:],  
 **"suffix(5)"**: sentence[i][-5:],  
 }  
 **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): *# [\_consec-pos-tagger]* **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)

**def** exercise7():  
 *# Design at least 5 features and report what these features capture.  
 # Report the accuracy of your classifier. Place your classifier code into the report* 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()  
 print (tagger.evaluate(test\_sents))

Result:

Exercise 7

0.8665739452943904

The feature that I capture is the suffix of a word. For example, we have a word like “sunday”. The five features are “s”, “su”, “sun”, “sund”, “sunda”. For those five features, I will also get the tag for each feature.

1. exercise 0 (0 is a dummy number in this case). Word features can be very useful for performing document classification, since the words that appear in a document give a strong indication about what its semantic content is. However, many words occur very infrequently, and some of the most informative words in a document may never have occurred in our training data. One solution is to make use of a lexicon, which describes how different words relate to one another. Using WordNet lexicon, augment the movie review document classifier presented in Chapter 6 to use the following two features on the intersection of words appearing in a document to classify and words appearing in “word\_features”:
   1. Make a binary feature which reports “KNOWN” if the word is found in WordNet (i.e. wn.synsets is non-empty) and “UNK” if it is not found.
   2. Make a lemma name feature. Select the first synset from wn.synsets and choose the first lemma name from synset.lemma\_names as the appropriate lemma. Report “UNK” if it is not found.

Report the accuracy of your classifier: use nltk.NaiveBayesClassifier, your **test** set should contain the first 100 instances in documents defined as follows:

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)]

The remaining instances in documents should be part of your **training** set.

How does this accuracy compare to the accuracy of the classifier trained on the original feature set from the book? (Note that accuracy may not improve.) Why do you think you observe the behavior you observe?

Code:

**def** exercise0():  
  
 documents = [(list(movie\_reviews.words(fileid)), category)  
 **for** category **in** movie\_reviews.categories()  
 **for** fileid **in** movie\_reviews.fileids(category)]  
 random.shuffle(documents)  
  
 **def** feature1(document):  
 document\_words = set(document)  
 features = {}  
 **for** word **in** document\_words:  
 **if** len(wn.synsets(word)) != 0:  
 features[**"{}"**.format(word)] = **"<KNOWN>"  
 else**:  
 features[**"{}"**.format(word)] = **"<UNK>"  
 return** features  
  
 **def** feature2(document):  
 document\_words = set(document)  
 features = {}  
 **for** word **in** document\_words:  
 synsets = wn.synsets(word)  
 **if** len(synsets) != 0:  
 lemma = synsets[0].lemma\_names()[0]  
 features[**"{lemmaNameIs}"**] = lemma  
 **else**:  
 features[**"{}"**.format(word)] = **"<UNK>"  
 return** features  
  
 feature\_dict = {**"feature1"**: feature1, **"feature2"**: feature2}  
 feature\_list = sorted(feature\_dict.keys())  
  
 **for** key **in** feature\_list:  
 featuresets = [(feature\_dict.get(key)(d), c) **for** (d, c) **in** documents]  
 train\_set, test\_set = featuresets[100:], featuresets[:100]  
 classifier = nltk.NaiveBayesClassifier.train(train\_set)  
 print(**"{} has accuracy {}"**.format(key, nltk.classify.accuracy(classifier, test\_set)))

Result:

Exercise 0

feature1 has accuracy 0.7

feature2 has accuracy 0.72

We can see that using lemma name, the accuracy is a little bit higher.

1. (Extra Credit 20 points) exercise 9. Design at least 5 features and explain them. Use nltk.NaiveBayesClassifier. Report the accuracy of your classifier built using all of the features that you designed. Use show\_most\_inforamtive\_feautures(5) functionality from the classifier to inspect the individual feature performance. Which of your features seem to be most influential? Note: http://www.nltk.org/howto/corpus.html#other-corpora provides a little more information on ppattach corpora than the textbook. Section 4 of the publication posted at <https://works.bepress.com/yuliya_lierler/55/> starts by describing the dataset by Ratnaparkhi et al. (1994). This is exactly the dataset included in ppattach in NLTK.

Code:

*# for exercise 9***def** noun\_features(inst):  
 features = {}  
 features[**'noun1'**] = inst.noun1  
 **return** features

**def** exercise9():  
 print(**'Extra Credit'**)  
  
 **from** nltk.corpus **import** ppattach  
  
 training\_ppattach\_corpus = ppattach.attachments(**'training'**)  
 noun\_ppattach\_corpus = [inst **for** inst **in** training\_ppattach\_corpus **if** inst.attachment == **'N'**]  
  
 features = [(noun\_features(inst), inst.prep) **for** inst **in** noun\_ppattach\_corpus]  
 cutoff = int(len(features) / 4)  
 train\_set, test\_set = features[:cutoff], features[cutoff:]  
  
 *# Naive Bayes Classifier* classifier1 = nltk.NaiveBayesClassifier.train(train\_set)  
  
 *# Decision Tree Classifier* classifier2 = nltk.DecisionTreeClassifier.train(train\_set)  
  
 print(**"Naive Bayes classifier"**)  
 print(**"Accuracy"**, nltk.classify.accuracy(classifier1, test\_set))  
 print(**"team"**, classifier1.classify({**'noun1'**: **'team'**}), **"researchers"**)  
 print(**"Decision Tree classifier"**)  
 print(**"Accuracy"**, nltk.classify.accuracy(classifier2, test\_set))  
 print(**"team"**, classifier2.classify({**'noun1'**: **'team'**}), **"researchers"**)  
  
 print(**"5 features:"**)  
 print (classifier1.show\_most\_informative\_features(5))

Result:

Exercise 9

Extra Credit

Naive Bayes classifier

Accuracy 0.5738127377592342

team of researchers

Decision Tree classifier

Accuracy 0.43514541661553563

team of researchers

5 features:

Most Informative Features

noun1 = 'stake' in : of = 41.9 : 1.0

noun1 = 'interest' in : of = 19.3 : 1.0

noun1 = 'offer' for : of = 17.5 : 1.0

noun1 = '%' of : for = 16.8 : 1.0

noun1 = 'million' in : for = 16.0 : 1.0