CSCI 8450 Chapter6: Assignment

Mohan Sai Ambati
Collaborated with Sai Tarun Battula

1. The 5 features used of this classifier is:
   1. last_letter(word)
   2. last4_letters(word)
   3. count_of_vowels(word)
   4. first2_letters(word)
   5. first_letter(word)

   Feature 1 is set as default feature and I incrementally added all the features in 5 steps. I compared accurarcy of all 3 classifiers with dev_set in each step and In step 5 I have compared the accuracy of all 3 classifiers using test_set.

   Step 1:

   Features = (1)
   Naive Bays Classifier :
   Accuracy of dev_set is : 0.772
   Decision Tree Classifier :
   Accuracy of dev_set is : 0.766
   Maxent Classifier :
   Accuracy of dev_set is : 0.772

   Step 2:

   Features = (1,2)
   Naive Bays Classifier :
   Accuracy of dev_set is : 0.77
   Decision Tree Classifier :
   Accuracy of dev_set is : 0.53
   Maxent Classifier :
   Accuracy of dev_set is : 0.77

   Step 3:

   Features = (1,2,3)
   Naive Bays Classifier :
   Accuracy of dev_set is : 0.772
   Decision Tree Classifier :
   Accuracy of dev_set is : 0.528
   Maxent Classifier :
   Accuracy of dev_set is : 0.762

   Step 4:

   Naive Bays Classifier :
   Accuracy of dev_set is : 0.668
   Decision Tree Classifier :
   Accuracy of dev_set is : 0.528
   Maxent Classifier :
   Accuracy of dev_set is : 0.718

   Step 5:

   Naive Bays Classifier :
   Accuracy of dev_set is : 0.67
   Decision Tree Classifier :
   Accuracy of dev_set is : 0.53
   Maxent Classifier :
   Accuracy of dev_set is : 0.726

   Final accuracy of classifiers :
   Naive Bays Classifier : 0.7013248847926268

Mohan Sai Ambati
Collaborated with Sai Tarun Battula

Decision Tree Classifier : 0.6657546082949308
Maxent Classifier : 0.7070852534562212

Naïve Bays and Maxent classifiers predicted with almost same accuracy and they does better than Decision Tree.

2. Accuracy : 0.78
The 5 features I picked are
contains(recognizes) = True pos : neg = 8.1 : 1.0
contains(unimaginative) = True neg : pos = 7.8 : 1.0
contains(schumacher) = True neg : pos = 7.8 : 1.0
contains(turkey) = True neg : pos = 6.5 : 1.0
contains(atrocious) = True neg : pos = 6.4 : 1.0
In whole movie_review corpus word "recognizes" appear 8 times more likely than it doesn't appear. But words "unimaginative", "schumacher" are almost 8 time more likely to be negative than positive. Similary, words "turkey" and "atrocious" are almost 6.5 times more likely to be negative than positive.

3. The features I used in this exercise are:
1. Suffix(1) of post.text() : captures last letter of post.text
2. Suffix(2) of post.text() : captures last 2 letters of post.text
3. Suffix(3) of post.text() : captures last 3 letters of post.text
4. prefix(1) of post.text() : captures first letter of post.text
5. prefix(2) of post.text() : captures first 2 letters of post.text
6. prefix(3) of post.text() : captures first 3 letters of post.text
7. previous post : captures the previous post
8. previous-class : captures the class of previous post

The posts in the corpus doesn't seem to have a particular order. The features asked in the question (previous-class) doesn't have much effect in classification process. So I used suffixes and prefixes to improve the accuracy of classifier.

Accuracy : 0.7622652088589852

Code:
```
def dac_features(post, i, history):
    features = {}
    features["suffix(1)"] = post.text[-1:].lower()
    features["suffix(2)"] = post.text[-2:].lower()
    features["suffix(3)"] = post.text[-3:].lower()
    features["prefix(1)"] = post.text[0:1].lower()
    features["prefix(2)"] = post.text[0:2].lower()
    features["prefix(3)"] = post.text[0:3].lower()
    if i == 0 or len(history) == 0:
        features["prev-post"] = "START"
        features["prev-class"] = "START"
    else:
        features["prev-post"] = history[i - 1].text.lower()
        features["prev-class"] = history.get('class')[i - 1]
    return features


class ConsecutiveDialogTagger():
    def __init__(self, posts):
```

Mohan Sai Ambati
Collaborated with Sai Tarun Battula

```
            train_set = []
            self.refined_set = []
            i = 0
            for post in posts:
                history = []
                featureset = dac_features(post, i, history)
                i = i + 1
                train_set.append((featureset, post.get('class')))
                self.refined_set.append((featureset, post.get('class')))
                history.append(post)
            self.classifier = nltk.NaiveBayesClassifier.train(train_set)

        def getClassifier(self):
            return self.classifier

        def getRefined(self):
            return self.refined_set


    def exercise7():
        train_set = nltk.corpus.nps_chat.xml_posts()[0:7000]
        test_set = nltk.corpus.nps_chat.xml_posts()[7000:]
        dialog_tagger = ConsecutiveDialogTagger(train_set)
        restDialog_tagger = ConsecutiveDialogTagger(test_set)
        print("Accuracy : ",nltk.classify.accuracy(dialog_tagger.getClassifier(),
    restDialog_tagger.getRefined()))
        print(dialog_tagger.getClassifier().show_most_informative_features(5))
```

4. Accuracy : 0.78
   Most Informative Features :
   [('lemma(recognizes)', 'acknowledge'), ('contains(recognizes)', 'KNOWN'), ('lemma(unimaginative)', 'sterile'), ('contains(unimaginative)', 'KNOWN'), ('lemma(turkey)', 'turkey')]
   First I thought this could improve some accuracy but after some observation accuracy is unchanged because we are using same word_features, instead of using True or False we are using other binary text like known and ukw. We are doing same thing either way on a same word_features data. So, accuracy remains the same.

5. Features used are:
   1. Noun1_suffix : captures the last letter of noun1
   2. Noun2_suffix : captures the last letter of noun2
   3. Noun1_prefix : Captures first 3 letters of noun1
   4. Noun2_prefix : captures first 3 letters of noun2
   5. Verb : captures the verb used in that sentence
   6. Special1: True if noun1 and noun2 ends with same letter. (for plural nouns)

   Accuracy : 0.7
   Most Informative Features :
   [('noun1_suffix', '4'), ('noun1_suffix', '%'), ('noun1_suffix', '0'), ('noun2_suffix', '4'), ('noun1_suffix', '5')]
   If we could find the similar sentence patterns in a large text then it could improve the accuracy even more.