```
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from collections import defaultdict
import sys
import pandas as p


def readfiles(trainfile,testfile):
    traindata = open(trainfile, 'r').readlines()
    trainfeatures = defaultdict(list)


    for i in range(0, len(traindata)):


        if 'TOK' not in traindata[i]:
            Words = []
            Words = traindata[i].split('. ')
            #print(Words)



            if 'NEOS' in Words[1]:
                trainfeatures['Label'].append('No')
            elif 'EOS' in Words[1]:
                trainfeatures['Label'].append('Yes')
            else:
```

```
    trainfeatures['Label'].append('NA')

#Feature1 Words Left to "."
Left = ''
Left = Words[0].split(' ')[1]
trainfeatures['LeftofP'].append(Left)
#print(Left)

#Feature2 Words Right to "."
Right = 'NA'
if '.' in Left:
    Right = Left.split('.')[0]
trainfeatures['RightofP'].append(Right)
#print(Right)

#Feature3 Length of L is less than 3
if(len(Left)<3):
    trainfeatures['LeftLen'].append('Yes')
else:
    trainfeatures['LeftLen'].append('No')


#Feature4 L is Capitalised
if(Left.isnumeric()):
    trainfeatures['CapLeft'].append('NA')
elif(Left.istitle()):
    trainfeatures['CapLeft'].append('Yes')
else:
    trainfeatures['CapLeft'].append('No')
```

```
#Feature5 R is Capitalised
if(Right.isnumeric()):
   trainfeatures['CapRight'].append('NA')
elif(Right.istitle()):
   trainfeatures['CapRight'].append('Yes')
else:
   trainfeatures['CapRight'].append('No')

#Feature6 Length of L is 1
if(len(Left) == 1):
   trainfeatures['LeftLenOne'].append('Yes')
else:
   trainfeatures['LeftLenOne'].append('No')

#Feature7 If L contains '.'
if '.' in Left:
   trainfeatures['PeriodInLeft'].append('Yes')
else:
   trainfeatures['PeriodInLeft'].append('No')

#Feature8 If L contains DoubleQoute'"'
if '"' in Left:
   trainfeatures['QouteInLeft'].append('Yes')
else:
   trainfeatures['QouteInLeft'].append('No')
```

```
testdata = open(testfile, 'r').readlines()
testfeatures = defaultdict(list)

for i in range(0, len(testdata)):

    if '. ' in testdata[i]:
        Words = []
        Words = testdata[i].split('. ')

        if 'NEOS' in Words[1]:
            testfeatures['Label'].append('No')
        elif 'EOS' in Words[1]:
            testfeatures['Label'].append('Yes')
        else:
            testfeatures['Label'].append('NA')

        #Feature1 Words Left to "."
        Left = ''

        Left = Words[0].split(' ')[1]
        testfeatures['LeftofP'].append(Left)

        #Feature2 Words Right to "."
        Right = 'NA'
        if '.' in Left:
            Right = Left.split('.')[0]
```

```
testfeatures['RightofP'].append(Right)

#Feature3 Length of L is less than 3
if(len(Left)<3):
   testfeatures['LeftLen'].append('Yes')
else:
   testfeatures['LeftLen'].append('No')

#Feature4 L is Capitalised
if(Left.isnumeric()):
   testfeatures['CapLeft'].append('NA')
elif(Left.istitle()):
   testfeatures['CapLeft'].append('Yes')
else:
   testfeatures['CapLeft'].append('No')

#Feature5 R is Capitalised
if(Right.isnumeric()):
   testfeatures['CapRight'].append('NA')
elif(Right.istitle()):
   testfeatures['CapRight'].append('Yes')
else:
   testfeatures['CapRight'].append('No')

#Feature6 Length of L is 1
if(len(Left) == 1):
   testfeatures['LeftLenOne'].append('Yes')
else:
   testfeatures['LeftLenOne'].append('No')
```

```
        #Feature7 If L contains '.'
        if '.' in Left:
            testfeatures['PeriodInLeft'].append('Yes')
        else:
            testfeatures['PeriodInLeft'].append('No')

        #Feature8 If L contains DoubleQoute'"'
        if '"' in Left:
            testfeatures['QouteInLeft'].append('Yes')
        else:
            testfeatures['QouteInLeft'].append('No')
```

```
trV=p.DataFrame(trainfeatures,columns=['Label','LeftofP','RightofP','LeftLen','CapLeft','CapRight','LeftLenOne','PeriodInLeft','QouteInLeft'])

tsV=p.DataFrame(testfeatures,columns=['Label','LeftofP','RightofP','LeftLen','CapLeft','CapRight','LeftLenOne','PeriodInLeft','QouteInLeft'])
    #print(trV)
    #print(tsV)
    trV["LeftofP"]=trV.index
    trV["RightofP"]=trV.index
    tsV["LeftofP"]=tsV.index
    tsV["RightofP"]=tsV.index
    trV['Label'] = trV['Label'].map({'Yes': 1, 'No': 0, 'NA':-1})
    tsV['Label'] = tsV['Label'].map({'Yes': 1, 'No': 0, 'NA':-1})
```

```
trV['LeftLen'] = trV['LeftLen'].map({'Yes': 1, 'No': 0})
tsV['LeftLen'] = tsV['LeftLen'].map({'Yes': 1, 'No': 0})
trV['CapLeft'] = trV['CapLeft'].map({'Yes': 1, 'No': 0, 'NA':-1})
tsV['CapLeft'] = tsV['CapLeft'].map({'Yes': 1, 'No': 0, 'NA':-1})
trV['CapRight'] = trV['CapRight'].map({'Yes': 1, 'No': 0, 'NA':-1})
tsV['CapRight'] = tsV['CapRight'].map({'Yes': 1, 'No': 0, 'NA':-1})
trV['LeftLenOne'] = trV['LeftLenOne'].map({'Yes': 1, 'No': 0})
tsV['LeftLenOne'] = tsV['LeftLenOne'].map({'Yes': 1, 'No': 0})
trV['PeriodInLeft'] = trV['PeriodInLeft'].map({'Yes': 1, 'No': 0})
tsV['PeriodInLeft'] = tsV['PeriodInLeft'].map({'Yes': 1, 'No': 0})
trV['QouteInLeft'] = trV['QouteInLeft'].map({'Yes': 1, 'No': 0})
tsV['QouteInLeft'] = tsV['QouteInLeft'].map({'Yes': 1, 'No': 0})



#CORE FEATURES

five=['LeftofP','RightofP','LeftLen','CapLeft','CapRight']



five_Y=trV[['Label']]

five_X=trV[five]



five_test_data_X = tsV[five]
```

```
five_test_data_y = tsV[['Label']]

clf = DecisionTreeClassifier()
clf.fit(five_X,five_Y)

predictions_Five=clf.predict(five_test_data_X)



testaccuracyFive=accuracy_score(five_test_data_y,predictions_Five)

print("Accuracy for 5 core features:",round((testaccuracyFive*100),2),"%")

#CORE FEATURES + 3 EXTRA FEATURES



all_Y=trV[['Label']]

all_X=trV[['LeftofP','RightofP','LeftLen','CapLeft','CapRight','LeftLenOne','PeriodInLeft','QouteInLeft']]


all_test_data_X =
tsV[['LeftofP','RightofP','LeftLen','CapLeft','CapRight','LeftLenOne','PeriodInLeft','QouteInLeft']]

all_test_data_y = tsV[['Label']]
```

```
clf2 = DecisionTreeClassifier()
clf2.fit(all_X,all_Y)


predictionsAll=clf2.predict(all_test_data_X)


testAccuracyAll=accuracy_score(all_test_data_y,predictionsAll)

print("Accuracy for All features:",round((testAccuracyAll*100),2),"%")


 #EXTRA 3 FEATURES
three_Y=trV[['Label']]

three_X=trV[['LeftLenOne','PeriodInLeft','QouteInLeft']]


three_test_data_X = tsV[['LeftLenOne','PeriodInLeft','QouteInLeft']]

three_test_data_y = tsV[['Label']]




clf1 = DecisionTreeClassifier()
clf1.fit(three_X,three_Y)
```

```
predictionsThree=clf1.predict(three_test_data_X)
#print(predictionsThree)


testAccuracyThree=accuracy_score(three_test_data_y,predictionsThree)

#print(three_test_data_y)



print("Accuracy for Three Extra features:",round((testAccuracyThree*100),2),"%")


count=0
sys.stdout = open('SBD.test.out', 'w')
for item in open('SBD.test', 'r'):

   if 'EOS' in item:

      if item.split()[2] == predictionsAll[count]:
         score += 1

         item = item.split()
         item[2] = predictionsAll[count]
         item = line[0] + " " + item[1] + " " + item[2]
         count += 1
         totalcount += 1
```

```
            sys.stdout.write(item)
    sys.stdout.flush()




def main():
    train=sys.argv[1]
    test=sys.argv[2]
    readfiles(train,test)

if __name__ == "__main__":
    main()
```