io.py

class DataStore

data structure to store entries and their labels, can also be replaced with database methods

- read_from_txt(filename) # return list of entries and training labels
- write_to_txt(entries,filename)

app_wrapper.py

class AppWrapper

- initialize(path_to_raw_data) #import data and initialize jieba and classification models
- update_jieba_dict(new_tokens)
- update_classifier(new_labels)
- tokenize(list_of_entries) #returns tokenized result
- classify(list_of_entries) # returns predicted labels
- get_hard_entries(n) # returns n entries to be labeled

octree.py

class OCTree

- build_tree(tokenized_entries, labels) # contruct tree and node labels
- export_to_json()
- compress_all() # returns all the compressed tokens

tokenzier.py

class JiebaInterface

- initialize_model(entries) # process all entries
- add_to_dictionary(new_tokens)
- tokenize(entry) #returns tokenized result of entry

Jieba library

classifier.py

class MultiLabelClassifier

- initialize_model(entries_with_label) # train classifier
- update_model(new_entries, new_labels)
- classify(entry) # returns labels and confidence scores

dependence

get entries and make predictions

return user annotation

pseudo-code for annotator workflow

```
app=AppWrapper()
app.initialize('raw_entry_list.txt')
while NOT_DONE:
    hard_entries=app.get_hard_entries(10)
    token_test = app.tokenize(hard_entries)
    label_test = app.classify(hard_entries)
    DISPLAY_MODEL_PREDICTION(token_test,label_test)
    token_user,label_user =
    GET_USER_ANNOTATIONS(hard_entries)
    app.update_jieba_dict(token_user)
    app.update_classifier(label_user)
```