Technical Report for iDMlib

Eric, Jia Guo

April 21, 2010

Document History		
Date	Author	Content
2010-04-19	Eric Xing	Write an intial format
2010-04-21	Jia Guo	Add API usage of KPE.

Abstract

This document presents the technical report for the project iDMLib in iZENEsoft. iDMLib aims to build a core library that integrates the various basic data mining algorithms and components that are developed in iZENEsoft to maximize the resusability.

Contents

1 Introduction

2 Duplicate Detection

2.1 API usage

3 Key Phrase Extraction

Key Phrase Extraction(KPE) is used to extract meaningful phrases from a large amount of documents. Also contains a lot of Named Entities in it. See Section 4.

3.1 API usage

Some test cases like `test/keyphrase-extraction/t_KPE.cpp` can be refered.

- Following code snippet demonstrate the defination of the ID to String mapping class.

```
//This is the IDManager declaration
class TestIDManager : public boost::noncopyable
{
public:
//init the IDManager with the working directory.
TestIDManager(const std::string& path);

~TestIDManager();

//convert string to id, also storage the id to string mapping.
bool getTermIdByTermString(const wiselib::UString& ustr, uint32_t& termId);

//convert string to id with string's POS tag
//, also storage the id to string mapping.
bool getTermIdByTermString(const wiselib::UString& ustr, char pos, uint32_t& termId);

//convert id to string.
bool getTermStringByTermId(uint32_t termId, wiselib::UString& ustr);

//put the id to string mapping into storage.
void put(uint32_t termId, const wiselib::UString& ustr);

//tell us whether one termId is a single phrase
//, like Korean compound  nouns.
bool isKP(uint32_t termId);

//To analyze the sentence and then give term id list.
//Generally, LA is required.
void getAnalysisTermIdList(const wiselib::UString& str, std::vector<uint32_t>& termIdList);

//To analyze the sentence and then give
//term string list, term id list, POS tag info list, term position list.
//Generally, LA is required.
void getAnalysisTermIdList(
const wiselib::UString& str
, std::vector<wiselib::UString>& termList
, std::vector<uint32_t>& idList
, std::vector<char>& posInfoList
, std::vector<uint32_t>& positionList);

//flush the storage.
void flush();

void close();

};
```

- Following code snippet demonstrate the definition of output class. We support three kinds of output.
  1. Keyphrases only.
  2. Keyphrases with documents frequncy(DF).
  3. Keyphrases with its documents support(doc ID list).

```
//the IDManager we used here.
TestIDManager idManager("./");

//declaration of the callback function due to the different output type.
//KP only
KPE_NONE<TestIDManager>::function_type callback1;
//KP with DF
KPE_DF<TestIDManager>::function_type callback2;
//KP with doc id list
KPE_ALL<TestIDManager>::function_type callback3;
```

The prototypes of *callback1*, *callback2* and *callback3* shows below:

```
typedef std::pair<uint32_t, uint32_t> id2count_t;
typedef wiselib::UString string_type;
//callback1
/**
* @param str label string
* @param score label score
*/
void output(const string_type& str, uint8_t score);

//callback2
/**
* @param str label string
* @param df document frequency
* @param score label score
*/
void output(const string_type& str, uint32_t df, uint8_t score);

//callback3
/**
* @param str label string
* @param id2countList doc id, freq in doc pair
* @param score label score
*/
void output(const string_type& str, const std::vector<id2count_t>& id2countList, uint8_t score);
```

One can use boost::function and boost::bind to specify the callback function.

```
callback1 = boost::bind(...);
```

- Following code snippet demonstrate the initialization of the module.

```
TestIDManager idManager("./id");

//KP with doc id list
KPE_ALL<TestIDManager>::function_type callback = boost::bind(..);
//init with IDManager and the callback function
//, also give a working dir.
KPE_ALL<TestIDManager> kpe( &idManager, callback, "./working");
//load resource used to scoring the phrases
//We can run KPE without any resource, but the result will be worse.
kpe.load("idmlib/resource/kpe");
```

- Following code snippet demonstrate the usage of KPE. We support three kinds of input
  1. ID list with their POS tagging and position.
  2. String list with their POS tagging and position.
  3. Raw article(which must be the most common usage).

```
//insert ID list.
std::vector<uint32_t> termList;
std::vector<char> posList;
std::vector<uint32_t> positionList;
//.... some processing
uint32_t docId = 1;
kpe.insert(termList,posList,positionList, docId);

//insert string list.
std::vector<wiselib::UString> termList;
std::vector<char> posList;
std::vector<uint32_t> positionList;
//.... some processing
uint32_t docId = 1;
kpe.insert(termList,posList,positionList, docId);

//insert raw article
wiselib::UString article;
//.... some processing
uint32_t docId = 1;
kpe.insert(article, docId);
//....
//close the KPE and begin doing extraction then call the callback function.
kpe.close();
```

4 Name Entity Classification

Name Entity Classification module is used to classify a string(supposed to be a noun phrase) to following classes:

- PEOP, represent for people, for examples: 黄昆, 哈里逊, 组织代表, 克里斯蒂安八世

- ORG, represent for ogranization, for examples: 北师大, 北京地质学院, 黑手党
- LOC, represent for location, for examples: 奥地利共和国, 九龙塘铁路站
- OTHER, noun phrase but not name entity, for examples: 巴士路线, 政治风气
- NOISE, non-noun phrase, for examples: 直线传播, 分析和解决

Note that this module is not a module of Name Enity Recognition, which can recognize name entity from a text. The difference is that the input string of Name Entity Classification is supposed to be a noun phrase, while the input string of Name Entity Recognition may be a sentence, paragrahp or a whole article.

4.1 API usage

The usage of this module is very simple, some test cases like `test/nec/xxx.cpp` can be refered.

- Following code snippet demonstrate the usage of the module training.

```
vector<NameEntity> entities;
// load entities
...

// set model path
string path = "model_path";
NameEntityManager neMgr(path);

// training
neMgr.train(entities);
```

- Following code snippet demonstrate the usage of the module prediction.

```
// set model path
string path = "model_path";
NameEntityManager neMgr(path);
// load nec models
neMgr.loadModels();

vector<NameEntity> entities;
NameEntity entity;
// load entities
...

// prediction
neMgr.predict(entities);
// or
// neMgr.predict(entity);
```

5 Document Classification

5.1 API usage