IR manager designing document.

Jia Guo

September 14, 2010

Document History		
Date	Author	Content
2010-09-14	Jia Guo	Draft, described the basic sub components of IR manager and how to provide the neccessary data to mining framework

Contents

1 Introduction

IR manager is performed as the basic IR toolkit in iZENEsoft. IR toolkit is a library that support research and development of information retrieval and data mining software. This **IR manager** will be used as the core component of our enterprise search engine: **SF1-R**.

2 IR manager and mining framework

While we talking about the API provided by IR manager, we need to consider on the requirement of mining framework and all possible data needed by it.

1. We're now sure that the mining framework should based on **IR manager**, all mining algorithms used the data sources in **IR manager**. This made our all mining modules like plugins which is more comfortable for developers, and also make the whole software more flexiable.

2. I don't think that use forward and inverted index only for mining algorithms is a good idea. In normal IR toolkits, for example Lucene which is the most popular one, have the feature that storage string content for any property. For "TITLE" property, we can do such operation:

```
ir_manager->insert( 1, "TITLE", "text of Title",
 PROPERTY_IS_INDEX.YES, PROPERTY_IS_STORAGE.YES );
```

   which will build inverted index for this title and also save the string content on local disk. Then we can get the TITLE property by

```
ir_manager->get_property(1, "TITLE" );
```

   This method will be used widely, for example display that property on web pages. So I think this feature is a must have and very easy-to-use one in any IR toolkit include ours.

3. Another reason is, take an example of similar image search algorithm. In this mining feature, we need to get the image url for every documents. So once we have the IR manager, what we should do for that is:

```
std::string img_url = ir_manager->get_property( docid, "ImgURL" );
iise_manager->insert( docid, img_url);
```

   Such mining algorithms can not be implemented by inverted or forward index only.

4. So my suggestion is, keep the **IR manager** has the properties storage ability and also the basic inverted index building, document-manager can be removed as **IR manager** can do the document storage itself. All mining tasks used the data sources from **IR manager** only, the algorithm will decide to use either storaged content or inverted index for input or both of them by itself. The algorithm may also do some text analysis(LA) due to the its own requirement.

5. Data mining performance comparasion, use forward and inverted index only as **A**, use above architecture as **B**. and we take TG(KPE) as example, and we also assume there's only one text property TITLE. All things they should do :

   - A.1 build normal inverted index for TITLE
   - A.2 storage TITLE in ir-manager(now it is managed by document manager)
   - A.3 run specific LA for mining usage in property TITLE.raw.
   - A.4 build forward index for TITLE.raw from A.3, and we also need term's attribute(like POS tag) in it
   - A.5 KPE iterated on all forward index in TITLE.raw.
   -
   - B.1 build normal inverted index for TITLE (the same with A.1)
   - B.2 storage TITLE in ir-manager (the same with A.2)
   - B.3 KPE iterate on TITLE property for all documents in ir manager (the same with A.5)

- B.4 run specific LA for KPE usage in property TITLE. (the same with A.3)

Maybe the time used in B.3 ¿ A.5, but the forward index(A.4) in not needed in B. So I'm sure that the time used in B ¡ A.

However, all of these steps are the input phase of KPE(or other algorithms, the same), less than 10% of the total mining time. So it will not affect the total mining time a lot. And in development view, B is better, that **IR manager** is more user friendly and easy to use.

3 IR manager and wildcard search

[TODO]

4 Detail design

From the above disscussion, **IR manager** is consist of these sub modules:

1. IRAnalyzer, one optional LA per property.
2. IRIDManager, be charge of generation term id and doc id.
3. IRDocument, actually a property name to property value map
4. IRPropertySetting, some settings for every properties, like the LA setting, whether to index or not, whether to storage this property on disk or not.
5. IndexManager, wrap it inside IRManager.
6. PropertyStorageManager, just like the DocumentManager before, but the name is more clear.

4.1 Example code fragments

For this IR manager, assume we have 3 properties: DATE, TITLE, CONTENT.

```
IRAnalyzer analyzer; //has the IRPropertySetting in it
//no index on DATE, but storage this property on disk.
analyzer.AddProperty( 1, "DATE", NULL, false, true);
//do index on TITLE with common_la, also storage it.
analyzer.AddProperty( 2, "TITLE", common_la, true, true);
analyzer.AddProperty( 3, "CONTENT", common_la, true, true);
//init the IRManager
IRManager manager(path, analyzer);
//init some document to index it.
IRDocument doc;
doc.InsertPropery("DATE", "20100101");
doc.InsertPropery("TITLE", "this is the title");
doc.InsertPropery("CONTENT", "this is the content");
manager.AddDocument(doc);
//....
```

4.2 Class relationship

Figure 1: classes of IR manager