AIEPmora an NLP Knowledge Representation and Retrieval Platform

A.I Maha Arachchi, Attanayake A.M.S.S.A.U.B, Phillips G.L.L and Vithanagama S.
Department of Computer Science and Engineering, University of Moratuwa, Sri Lanka.

Abstract-**This paper discusses AIEPmora; a natural language processing system designed to maintain a conversation with a human user. The architecture is much similar to an automated chat bots with several overhauls to integrate varying knowledge bases. The system is designed to consume plain text so that knowledge can be added without worrying about its structure or organization. It does not require link to be present in the text and can manage various type of wh questions plus human like expressions like "hi" , "good morning". This paper presents a high level architecture of AIEPmora and how its component integrates to create a human like chat bot.**

Index Terms—**Chat Bots, Knowledge Representation, Knowledge Retrieval, NLP**

I. INTRODUCTION

Today in the digital era, text has become the primary medium of representing and transmitting information . People's lives are saturated with textual information and there is an increasing demand to develop a method to help them manage and make sense of information overload [1].

In corporate environment there is a massive amount of link free information like emails, text files and documents trapped in repositories which cannot be searched. Building a search algorithm like that of internet search engine is difficult as link ranking methods cannot be directly applied on link free information.A classic example is frequently asked questions [2] where you have a good structural representation of knowledge without proper links. Corporate email client is another source of information where both structure and inter links are not available. Although enterprise search engine exist they don't provide chat bot like features and solely depends on keyword search [3], [4], [5].

Currently existing humanoid chat bots like Alice Bot [6] requires a well structured Knowledge Base (KB). Alice Bot depends on an ontology structure called AIML (Artificial Intelligence Markup Language) and can be used to represent quite a complex knowledge. Converting of day today information like emails to a well structured KB is a daunting task or practically impossible. AIML depends on pattern matching and most of statistical based NLP processing techniques cannot be directly applied on an AIML KB. The inference engines of modern automated chat bots are based on pattern matching and regular expression and their answers are somewhat limited to small set of patterns. In most cases when an answer cannot be generated the system settle down to a general answers like "what do you mean" or "ah tell me again". The knowledge base is very rigid and cannot accommodate ever changing information and user based. Although AIML is a restrictive representation it does have its advantages like fast response time and answering to general expressions like "hi", "whats the color of sky" or "how is the day".

The aim of AIEPmora project is to provide an easy to use chat bot where a general user can query questions much like a human. The system provides meaningful answers by searching its knowledge bases. AIEPmora converges final results at various steps to pin point an answer.

The system compromise of 4 main units. 1) Query engine, 2) Two structured knowledge bases 3) Unstructured knowledge base with an indexer 4) NLP algorithm engine. These components will be discussed in detail in the coming sections.

AIEPmora relies on a simple text query for all the inputs and displays the matching sentences in a simple text output. This simple text input, output makes it easy to integrate with widely available information retrieval platforms. This also allows sound recognition and text readers to be directly incorporated to create a human like conversation engine. A conversation can be carried out in a human language and the system gives meaningful answers. When an expression like "hi" or "good morning" is raised the system answers much like a human.

We thought hogging into a structural KB is a limiting factor and add a plain text consuming engine where day today knowledge can be easily included. A novel KB switch is added to select among different type of KBs available. The KB switch allows intelligently select structural and unstructured KBs based on patterns it has learned previously.

II. ARCHITECTURE

A high level architecture of the system is shown in Fig. 1.

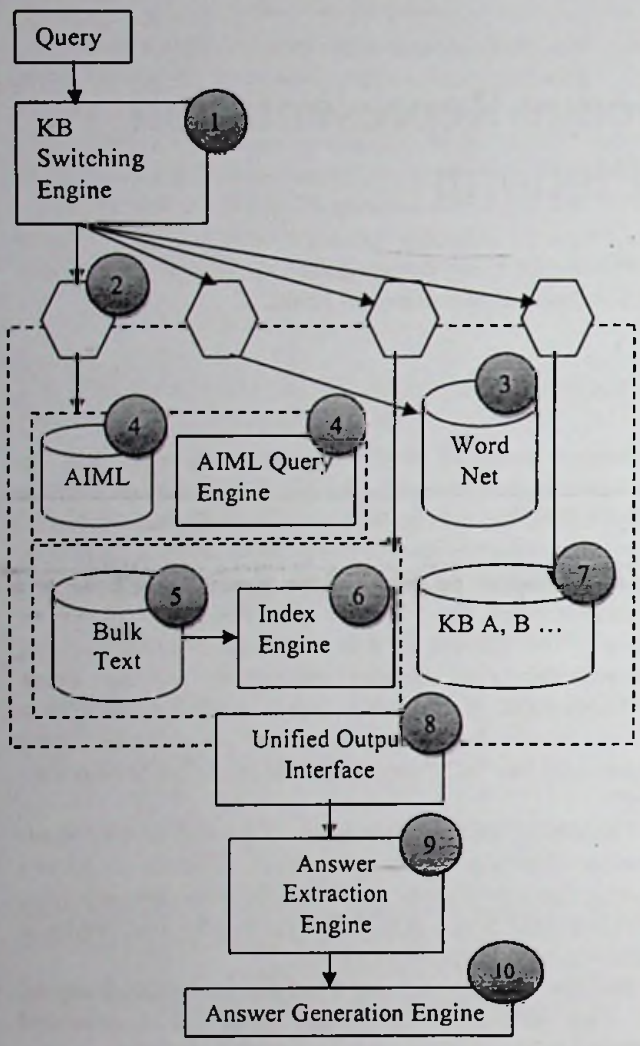

Fig. 1.

1) Usual chat bots maintain only one type of KB. This results in several limitations as some questions can not to be answered through an all in one general KB. For instance a dictionary is suitable to answer questions like "What is the meaning of fortification?" but may not be the ideal KB for a questions like "Who is the president of united state in 1995". Sometimes answers should be extracted from resources like internet and there should be a mechanism to switch between different KBs. KB switching engine runs several algorithms to analyze the input query alone with some context analysis to identify and select the best KB to answer the question. It relies on a learning mechanism to identify query patterns and switch KB according to previously learned patterns. The internal algorithms and working of KB switching engine is discussed separately in another paper [1].
2) Each KB is plugged using a general interface so that KB switching engine can select KBs irrespective of their structure or organization.
3) Questions related to definitions are queried into WordNet (Princeton University WordNet) [7]. WordNet is well structured to select definitions and finding object types. For instance finding the meaning of the word "amalgamate" or resolving that a car is a passenger vehicle and a passenger vehicle is a general vehicle can be done using WordNet.
4) To answer human expressions that have no specific meaning, an AIML [8] based query engine is utilized. For instance to answer expressions like "ah' "ouch", "hoo hoo" can be done with AIML. It also helps in answering some general questions like "Who is your father?" or "How is the weather like?"

   AIML is an XML based language to represent Artificial Intelligence. AIML describes a class of data objects called AIML objects and partially describes the behavior of computer programs that process them. Main components of an AIML document could be identified as categories, patterns and templates.
5) A Chabot is useless unless it can answer a wide variety of questions. Most of modern chat bots can't consume plain text. We see this as a limitation and integrated a plain text consuming engine to AIEPmora.

To answer questions like "Who is the president of United State in 1995?" or "What is the first Benz car with 4000cc Engine?" or "Who is the father of Bill Gates?" you need to have intensive knowledge base. We assume that such knowledge cannot be limited to a specific structure and can only be represented with plain text. Beside all most of such information are available in plain text format.

To carry out the research and to have large amount of textual information we created an automated crawler and downloaded over 0.2 million pages from Wikipedia [9]. Most of NLP context analysis cannot be directly applied on HTML mixed text so we created some parsers and converted HTML into bulk text removing unwanted tags. The bulk text gave us context rich pure text. AIEPmora kept some important tags like <H1> <H2> and <b> so that during index scoring it can define which documents ranks high. High scores are given to large text sizes. A density parsing algorithm is used to remove high html tag density areas while preserving low html tag density areas where bulk text resides as in Fig 2. During the process some information (Especially those at boundaries) will be lost due to density parsing. Once the text is extracted it will be indexed using an advance indexer.

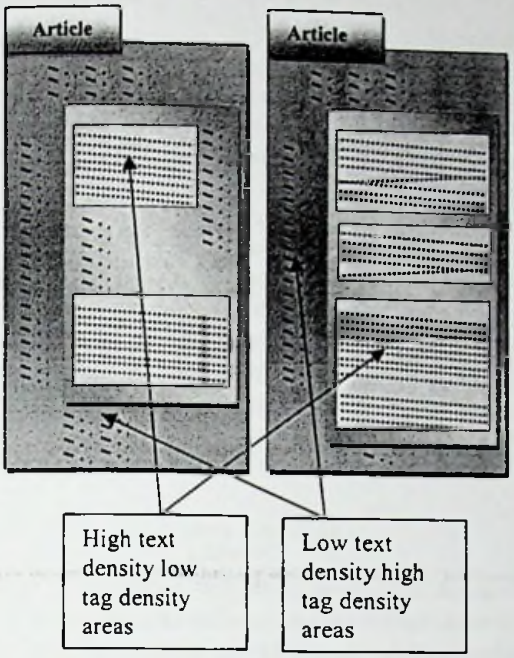

Fig. 2.

6) Information which is not searchable or retrievable has no apparent usage. Indexing is the process of maintaining some meta-data, of a large bulk of information, so that when it is necessary to search for some particular information it is possible to quickly navigate and locate the information needed. Document is the unit of storage inside index and it will be stored with several Meta data like title, security_info, keywords and file_path, modified_date etc. Note that document will not be stored as it is [10].

For instance you can search "keywords: Bill Clinton" so that documents with Bill Clinton in the keywords field will be returned. Fields allow capturing the context of the document in the way suitable for that document. For instance document of an email message will have fields like from, to, received date etc.

III. Index Search Procedure

Each document is represented as a vector in an n dimensional vector space [11].When a search query comes another search vector is created and is positioned on the vector space. Documents closer to the search query vector get a high score during ranking and are returned as the search results. For the purpose of visualization let us consider a situation where there are only 3 documents in the index. (In reality there can be N document in the index which we can't easily visualize when N>3). Each axis in Fig. 3 corresponds to a document. The graph shows two search queries; one in green color (solid line) and the other in red (dash line).

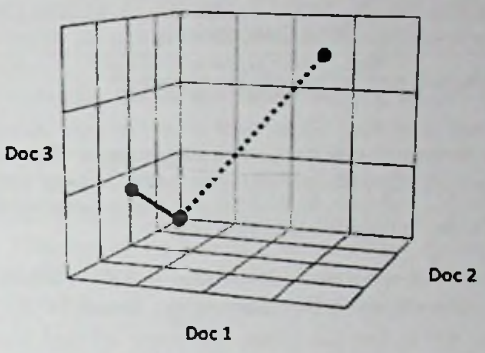

Fig. 3.

All the vector representations of the documents are normalized, so that documents with high textual content won't always get high scores. Similarly search query vectors are normalized so that search queries with higher number of words are comparable with those with less.

Once the vectors are created the Cosine between the search query vector and document vector is calculated (Fig. 4). The one with the lowest Cosine gets the highest score.

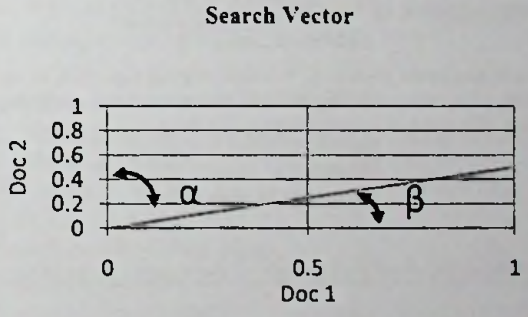

Fig. 4.

$$Cos(\theta) = \frac{V_1 . V_2}{|v_1| * |v_2|} \qquad (1)$$

For calculation of $|v_1|$ or $|v_2|$ *Euclidean distance* is used. Clearly in the example, $\cos\beta < \cos\alpha$,There for the corresponding search vector scores Doc 1 being more relevant to the search than Doc 2. Although the theory works as above, in real implementation the scoring mechanism is developed to prevent noise, and to add search time and index time term boosting.

Score for query q and for document $d = (\text{OverLap}(q, d) * \text{QueryNormalizeFactor}(q)) * \left(\sum_{v\ term\ t \in query\ q}\left(TF(t,d) * IDF(t)^2 * Boost(t) *\right.\right.$

$$ITB(t,d)))$$ (2)

Term Descriptions

$$\text{OverLap}(q,d) = \frac{Num\ of\ over\ lap\ terms\ in\ q\ and\ d}{Total\ number\ of\ terms\ in\ q}$$ (3)

The value depends on how many search query terms are found in the document. If the document contains a large number of query terms the function returns a high value. On the other hand if the document contains a few query terms the function returns a low value.

$$\text{QueryNormalizeFactor}(q) = \frac{1}{\sqrt{SSW}}$$ (4)

$$SSW = Boost(q)^2 * \sum_{\forall\ term\ t\ \in query\ q} (IDF(t) * Boost(t))^2$$ (5)

Normalize the search query allowing all search queries to be comparable. SSW is Sum of Squared Weights.

$$Boost(x)$$

Returns a real number associated with term/field/ document x. Each term/field/document is associated with a factor which can be used as a multiplication factor. This allows certain terms to be artificially boosted and give more weights during scoring.

$$IDF(t) = 1 + log_e \frac{Total\ number\ of\ documents}{Number\ of\ documents\ containing\ term\ t}$$ (6)

Inverse Document Frequency. This helps in giving high marks to rare terms. If only a few documents contain a rare term t, then term t gets a high value. If lots of documents contain the term t, term t gets a low value.

$$TF(t,d) = \sqrt{Number\ of\ times\ term\ t\ appears\ in\ document\ d}$$ (7)

This allows document d with a high number of term t to have a high score.

$$ITB(t,d) = Boost(d) * \frac{\prod_{\forall\ field\ f\ in\ d\ containing\ term\ t} Boost(f)}{\sqrt{Number\ of\ term\ t\ in\ the\ fields}}$$ (8)

Index time Term Boosting. This allows different terms, fields or documents to be boosted during indexing time.

The indexer supports common search techniques implemented using query syntax. The query syntax can handle Boolean operations, Boosting, Proximity Search, and Wildcard search.

1) Boolean Search (AND, NOT, OR)

For instance searching for Bill AND Microsoft (Will return documents containing both words Bill and Microsoft) Bill OR Microsoft (Will return documents with Bill or Microsoft)

2) Boosting

This allows certain terms to be boosted during search. For instance if you want to boost the term Colombo by 10 times in the search "Colombo Sri Lanka" you can search with a query Colombo^10 Sri Lanka

3) Proximity Search

Allow to check whether certain words appear in some proximity to the other. "Network sustaining"~5 (Will search for documents where sustaining occurs within 5 words (eliminating stop words) after Network)

4) Wildcard search

You can use the wild card * to refer any matching string. Net* (Will match documents containing the letters Net (in order) followed by any number of characters)

IV. Answer Extractor

Answer extraction is the process of extracting answers from several candidate sentences return after searching knowledge bases [10]. An outline of the main extraction algorithm is given below.

Input : user question
Output: sentence containing the answer to the question

```
extract_answer(question)
{
string [ ] words_in_question;
string [ ] tags_for_words;
string [ ] selected_words;
string [ ] answers;

question ← convert_to_lowercase(question);
words_in_question ← split_the_question(question);
tags_for_words ← tag_the_words (words_in_question);

if(tag of a word in the question is important)
selected_words ← word    //here, which part of speech
        // does that word belong to is considered
if (a word in the selected_words is a name )
        remove that word from the selected_words;
        add it to removed_words;

    selected_words ← stem(selected_words);

    for each word in selected_words
        synonyms ← get_the_synonyms(word);

    for each word's synonyms
        if(a sentence contains a word from the
synonyms)
            answers ← sentence;

    resolve_pronouns(answers);
}
```

A. Ambiguity Resolution and Predicate Engine

The predicate engine depends on set of context sensitive causes and single output.

Cause 1, Cause 2, Cause 3 ... → Output A

Once trained, the engine can give statistically meaningful output for occurrence of several context sensitive causes not trained before. The algorithm utilizes Bayesian mathematics to analyze context. The predicate engine can be use to find whether a given word is a "Name" or a "Location". It can also be used to find whether a given word is a name of a boy or a girl.

B. Word Stemming and reverse Stemming

The queried questions can have words like "worried", "married", "cooked", "gone" etc, but matching documents may represent them with words like "worry", "marry", "cook", "cooking", "go" etc. To stem the words and match the relevant occurrences Porter Stemming algorithm is utilized [12].

During sentence extraction you sometimes need to reverse stem the word. For example the search query may have a word like "go", but the document may contains words like "gone" or "went". English Language has spelling patterns which can be readily utilized to reverse stem the words. For example consider the sentence

"Where did Abraham Lincoln study?"

Knowing that "did" is a past tense can be utilized to find the reverse stem of the word "study" and look for words like "studied" and "studying". Before applying reverse stemming on words they should be first qualified to be a verb from POS tagging.

C. Synonyms Matching

Consider the query "how many pilots died in the 2005 space shuttle crash?". Here the relevant document may not have a word called pilot. Instead it may contain some words such as aviator, aeronaut, airman, flier or flyer. This shows the importance of using synonyms for the search. As matching synonyms can give so many output documents synonym match is not done in indexer level. It's done only at final stages of answer extraction.

D. Pronoun Resolution

A problem can arise when the expected answer contains pronouns which refer to the main subject described in the question. To understand this problem we shall consider the following user question,

User: Where did Bill Gates study?

The answer for this question can be hidden in the document in the format,

"He enrolled at Harvard College in the fall of 1973 intending to get a pre-law degree, but did not have a definite study plan"

The pronoun 'he' in the above sentence actually refers to Bill Gates. But there is no way for the indexer to resolve this. The original proper noun referred by the pronoun depends on previous sentences and how the pronoun is occurred in the document. First candidate sentences are checked for pronouns using POS tags of the words. Pronouns have the tag "PRP" and "PRPS". "PRP" stands for pronouns like he, she, it, they and "PRPS" stands for pronouns like his, her, their. If the sentence contains pronouns, sentences before the candidate sentence and the words in the same sentence before the pronoun are considered. In our approach, we try to resolve the pronouns he, his, she and her. Previous sentences and the preceding section of the candidate sentence are checked for any names. When a name is found (names have the tag NNP and NNS), it is tried to resolve as a male or female name. This is done using a trained predicate engine trained to identify the gender.

The output of the indexer can contain several sentences with matching proper nouns. To get more convergent result proper nouns are removed using a predicate engine. Predicate engine automatically analyzes word context and determines which words have more probability to be proper nouns. To train the engine set of cases are given along with their preferred outcomes. For example if we are interested in finding names of people the engine can be trained as follows.

daniel cjay Jackson → NAME_YES
matthew haydon → NAME_YES
kyle mills →SHE
bull dog → NAME_NO
fat bully→ NAME_NO
my father→ NAME_NO
John Mike→ HE

when you want to resolve whether the word set "daniel cjay bull" is a name or something else the engine automatically calculate relevant context giving a probability value from 0 to 1.

V. Answer Generation

The responsibility of the Answer Generator is to identify the exact answer from those sentences selected and generate the answer in Natural Language. From the set of trained question patterns, the answer generator tries to match the user question with the trained questions template. If there is a matching template the answer generator generates an answer based on the POS tags of the answer pattern and the sentence containing the answer.

Pattern teaching is done by inserting a question and an answer in natural language. The pattern for the question and

the answer is first normalized. This normalization is done in two ways. First it is done for POS tags and then for word tokens.

Normalization is done because similar pattern of sentences should be recognized as the same. Consider a name of a person. It can be a one word name like "Einstein" or it can be a multi word name, like "Albert Einstein". Since all these words refer to a name of a person, it should be normalized giving only one tag and a token. Similarly, this process should be applied for other word sets too. As an example "big Bill" and "big bad Bill" are considered as similar patterns. Words like "and", "a", "an" etc should be omitted in normalization process, since they do not make a sufficient difference in the meaning of the sentences.

Consider the following question and answer as a teaching pattern:

"Who is mother of Maggie?", "The mother of Maggie is *" (* marks the exact answer expected).

After normalizing, the normalized POS tags are "WP VBZ NN NNP" and "NN NNP VBZ SYM".

Since the question is of type "Who", the answer should be a name. Therefore the token "SYM "should be replaced with the tag "NNP". This yields two patterns for this answer.

The normalized tokens are "Who, is, mother, Maggie" and "mother, Maggie, is, *".

After normalizing, the reference to the similar words of the answer in the question are marked using an Integer array. For the above example, the words "mother", Maggie" and "is" can be found in the question. The reference (the number of the token of the similar word in the question) is marked using an integer array of "3, 4, 2, null". Since the first token is taken as 1, "null" is marked as 0. Thus the integer array should be "3, 2, 1, 0".

When searching for an answer for a particular question, the Answer Generator first selects similar normalized POS tag patterns in the taught patterns. Then it searches for answer patterns in the selected sentences in the document for each taught question patterns selected. When it finds a similar pattern, it checks whether the references for the words in the question are correct.

Consider the example question "Who is the father of Albert Einstein?".

After normalizing it would be "WP VBZ NN NNP" and tokens "Who", "is", "father", "Albert Einstein".

Since the tags are similar to above teaching pattern it would be selected too.

Imagine the answer within the documents is, "The father of Albert was Hermann Einstein".

After normalizing it would be "NN NNP VBZ NNP" and tokens "father", "Albert", "was" and "Hermann Einstein".

Since it is similar to answer pattern in the above teaching example, those tokens will be selected and checked for references. "Father" can be found in the 3rd token of the normalized token of the question. When comparing nouns, synonyms are taken from the WordNet are also used to avoid ambiguity. "Albert" is similar to "Albert Einstein" since it is a name, and name can have multiple words. It is found in the 4th token. Tokens "is" and "was" will be compared after stemming since they are verbs. Last one "Hermann Einstein" will not be searched since the reference integer array has the value 0 for the respective token.

Therefore these tokens are selected for the answer. For generating the answer in natural language there is an additional string array kept in the pattern teaching process. All the tokens are added to this string array, but all tokens which are selected to be normalized, would be replaced by the value "null". For the above teaching pattern (The mother of Maggie is *) the string array is "The", "null", "of", "null", "null", "null".

In natural language answer generating, the null values in this array would be replaced by the tokens selected for the answer. For the above example, it should be "The", "father", "of", "Albert", "was", "Hermann Einstein" after the replacement is done.

It is then converted to a string as follows,

"The father of Albert was Hermann Einstein".

After identifying the answer, it is generated in Natural Language. These are some examples of how it is done.

Where do you live? In Paris. When will Ben have lunch? At 1pm.

Who did she meet? She met Ram.

Why hasn't Tara done it? Because she can't.

To increase the performance, whenever a sentence is found having a score above certain level, it is proceeded to generate an answer. If the answer can be generated from it, the search can be stopped without consuming much time. If the system can't generate the answer from the sentences, it will output the sentence with the highest score, directly extracted by the answer extractor.

ACKNOWLEDGMENT

AIEPmora team would like to thank Dr. Shehan Perera and Dr. Ravindra Koggalage,who supervised the AIEPmora project. Special thanks to our department head Mrs. Mrs. Vishaka Nanayakkara and all the staff members of CSE department of university of Moratuwa..

REFERENCES

[1] Lui H. ,Singh P, "ConceptNet – a practical commonsense reasoning tool-kit", BT Technology journal,

October 2004
[2] Zebra , Available: http://www.indexdata.dk/zebra/
[3] Zebra, Available: http://www.indexdata.dk/zebra/
[4] Wumpus, Available: http://www.wumpus-search.org/
[5] Terrier, Available: http://ir.dcs.gla.ac.uk/terrier/
[6] Alicebot, Available: http://www.alicebot.org
[7] G. A. Miller, et al. (1990). "Introduction to WordNet: An On-line Lexical Database". Int J Lexicography 3(4): pp.235-244.
[8] Wallace R., Bush N. (2001) "AIML Specification 1.0.1": [Online] Available: http://www.alicebot.org/TR/2001/WD-aiml/
[9] G. A. Miller, et al. (1990). "Introduction to WordNet: An On-line Lexical Database". Int J Lexicography 3(4): pp.235-244.
[10] Singhal A.(2001). "Modern Information Retrieval: A Brief Overview". Bulletin of the IEEE Computer Society Technical Committee on Data Engineering 24(4):35-43.
[11] Salton, G., McGill, M. J. 1983 "Introduction to modern information retrieval". McGraw-Hill
[12] C.J. van Rijsbergen, S.E. Robertson and M.F. Porter, 1980. New models in probabilistic information retrieval. London: British Library. (British Library Research and Development Report, no. 5587).