Answer Analysis And Evaluation

Mahima Rao
(1MS15CS063)
Computer Science and Engineering
RIT, Bangalore, India

Kanaka Samagna
(1MS15CS051)
Computer Science and Engineering
RIT, Bangalore, India

Kevin George
(1MS15CS056)
Computer Science and Engineering
RIT, Bangalore, India

Anagha Vembar
(1MS15C013)
Computer Science and Engineering
RIT, Bangalore, India

Abstract **- Automatic evaluation of answers is key to providing teachers an efficient way to examine the student's answers. In line with this, this paper studies the possibility of automatically determining the correctness of answers depending on a given answer key. The proposed approach provides three basic algorithms and further provides an introduction to the powerful tools available for text analysis. The implementation of the final algorithm incorporates natural language processing tools and vectorization models to map the similarities.**

1. INTRODUCTION

Computer Assisted Assessment(CAA) is when computers are used in the assessment of student learning. The idea of using computers to assist learning process has transformed the field of learning as well as teaching. The process of evaluating innumerous number of papers manually can be tedious, cumbersome and time consuming.

With the advancement of technology in natural language processing (NLP) and text analysis this process can be automated. Developing procedures for computers to understand answers written in natural language is an interesting and challenging problem. Text Analysis or Natural Language Processing (NLP) is a way for computers to understand, analyze, and derive meaning from human language in a smart and useful way. Such mechanisms allow teachers to obtain evaluations to answers in an efficient way. A Q&A system normally is a computer program, which queries a structured database, an unstructured dataset or a document to compare the submitted answer and the answer key.

The main contributions of the paper are as follows:

- Answer analysis using sentence tagging and keyword(noun) comparison
  The similarity is compared by taking an average of the similarities present in the sentences of the answer key and answers provided by the students.
- Answer analysis using Naïve Bayes classifier and web scraping
  Naïve Bayes classifier is the algorithm used for text classification and web scraping is used for extracting data from websites for training.
- Answer analysis using bag of words, tf_idf model,Gensim and NLTK.
  Tf–idf weighting schemes are used as a central tool in scoring and ranking a document's relevance. Gensim is used to handle large text collections, using data streaming and efficient incremental algorithms. NLTK libraries provide various packages and methods to carry out natural language processing.

2. PROBLEM STATEMENT

In teaching and learning, variation among individuals is bound to exist. Professors approach their jobs in different ways. Likewise, each student learns differently and master different amounts of material in their courses, so the answers thus provided will also be varied. Measuring the performance will be the main problem as there is only one answer key provided and the automated evaluator machine should understand the importance of the relevant keywords and the underlying meaning of the answer. The analyzer must further evaluate and assign appropriate marks by comparing the similarity of the answer key provided and student's answer.

3. KEY IDEA

In this paper, we put forth the design, analysis and implementation details along with some preliminary results to build a system that integrates the process of automated answer paper analysis with minimal teacher involvement. This method not only automates the traditional classroom scenario but also overcomes its inherent shortcomings and fallacies. The basic idea is to incorporate natural language processing tools and

vectorization methods to the map the similarity between the answer entered by the student to the answer key provided.

4. BACKGROUND INFORMATION ABOUT TEXT ANALYSIS

4.1. Text analysis

Text analysis is about parsing texts to extract machine-understandable information from them. The purpose of text analysis is to create sets of structured data out of heaps of unstructured, heterogeneous documents

4.2. NLTK

4.2.1. Tokenizing

Tokenizing is the process of splitting sentences or words into entities based on certain rules. A sentence is tokenized into words.When you tokenize a paragraph,sentences are the tokens. This can be achieved in python programming by using a simple split()function. This can also be achieved by using a split by period, space, and then a capital letter. Regular expressions can also be used to split.There are a few drawbacks to using regular expressions like salutations and concatenated words cannot be split properly. These problems can be solved by using NLTK.tokenize module.

```
from nltk.tokenize import sent_tokenize, word_tokenize
print(sent_tokenize(EXAMPLE_TEXT))
```

4.2.2. Stemming and Lemmatization

Stemming is a normalizing method. In order to normalize sentences and shorten the lookup, and we using stemming.Other than when tense of a sentence is involved many variations of words carry the same meaning. In order to match variations of words that mean the same thing we use this method of normalization. Porter stemmer is one of the most popular and efficient stemming algorithms that can be used to implement this.

```
from nltk.stem import PorterStemmer
.
ps = PorterStemmer()
ps.stem(w)
```

A very similar operation to stemming is called lemmatizing. The major difference between these two are,lemmas are actual words whereas stemming can often create non-existent words. Therefore the root stem,cannot be found in the dictionary,but a lemma can be found. Sometimes you will wind up with a very similar word, but sometimes, you will wind up with a completely different word. This is done by:

```
from nltk.stem import WordNetLemmatizer

lemmatizer = WordNetLemmatizer()
print(lemmatizer.lemmatize("word"))
```

The result of lemmatizer.lemmatize() is the normalized version of a word.lemmatize() takes a part of speech parameter, "pos", which defines which category it belongs to.The default is "noun".

4.2.3. Parts of speech tagging

Speech tagging is the process of labelling words in a sentence as nouns, adjectives, verbs,adverbs etc.It can also label by tense, and more. It is one of the most powerful tools in the NLTK module. To implement this a tokenizer, called the PunktSentenceTokenizer.This tool is capable of unsupervised machine learning, so you can actually train it on any body of text that you use.

```
from nltk.tokenize import PunktSentenceTokenizer
.
custom_sent_tokenizer =
PunktSentenceTokenizer(train_text)
tokenized =
custom_sent_tokenizer.tokenize(sample_text)
```

The output should be a list of tuples, where the first element in the tuple is the word, and the second is the part of speech tag

4.2.4. Stop words

The basic idea of NLP is to do some form of processing or analysis,where the machine can understand what the text means or implies.This can be implemented by pre-processing the data which filters out useless data. These useless words (data)in NLP,are referred to as stop words. Stop words are words that contain no meaning, and we want to remove them.This can be implemented by storing a list of words that are considered to be stop words according to your requirement. NLTK has a set of words which are considered as stop words, you can access it via the NLTK corpus with:

```
from nltk.corpus import stopwords
stop_words = set(stopwords.words('english'))
```

The above function stopwords.words() returns a set of stopwords in the English language and we can filter out stop words from out tokenized sentenced by using a for loop and thereby obtaining a set of tokenized words free of stopwords

4.2.5. Naïve Bayes

Text classification is a major part of text analysis, which can be implemented using different algorithms. Here we use Naïve Bayes algorithm which uses supervised machine learning that splits the present existing data into training and testing set.After it is trained, we test the model by giving it some testing data and check for results.The result of this is based on how the model will categorize data.

Training of the classifier is done by:

```
classifier =
nltk.NaiveBayesClassifier.train(training_set)
```

Testing the classifier by:

```
print("Classifier accuracy)
percent:",(nltk.classify.accuracy(classifier,
testing_set))*100)
```

This gives the result based on what the machine learnt based on the training set.We can also retrieve only the important features using this function call:

```
classifier.show_most_informative_features(15)
```

4.3. Tf_idf model

4.4.

Tf-idf is an acronym for term frequency–inverse document frequency, it is based on numerical statistics that shows how important a word is in a document or corpus. The Tf_idf value increases as the number of times a word in a document increases, but it is often an offsetof the frequency of the words of the corpus, which helps adjust for the fact some words appear more frequently than others.

Tf_idf can be successfully implemented for stop-words filtering in various subject fields, including text summarization and classification.

4.5. Gensim

Python uses an open-source topic modelling and vector space toolkit. It uses SciPy , Cython and NumPy to increase its performance .Gensim is used to handle large text collections, using data efficient incremental algorithms and data streaming which makes it different from other scientific software packages that occur only in-memory processing and Target batch.

Gensim is usually implemented with word2vec , document2vec and Tf_idf.

4.6. Bag of words

The bag-of-words model is a simplifying representation used in information retrieval and NLP . In this model, a text is represented as the bag of its words, by ignoring word order and grammar while not changing multiplicity. The bag-of-words model is generally used in methods of document classification.

One of the main uses of Bag-of-words model is feature generation. After transforming the text into a bag of words, we calculate measures to characterize the text. The most common type of feature calculated from the Bag-of-words model is term frequency, namely, the number of times a term appears in the text.

4.7. Web scraping

Web harvesting ,Web Scraping or web data extraction is data scraping used for extracting data from websites . Web scraping software accesses the World Wide Web using the Hypertext Transfer Protocol, or through a web browser.

Web scraping a web page involves fetching it and extracting from it. Therefore, web crawling is a main component of web scraping, to fetch pages for later processing. Once fetched, extraction can take place.Web scraping parses,searches or reformats the contents of a page. The main idea is to take something out of a page, to make use of it for various other purposes.It can be implemented using the Beautifulsoup API as well.

Pseudo code:

```
input(Url To Scrape from);
html -> get(html for web)
Convert html to string
Declare list 1
for each character in htmlString:
  Append position of '<' to list 1
Declare list 2
for each character in htmlString :
  Append position of '>' to list 2
n1->length(list1)
n2->length(list2)
for i in range 0 to n1:
  if(if String between list1[i] and list2[i] is of
length > 5 and not contains : or ;):
      print(print String between list[i] and list[2])
```

5. BUILDING UP TO OUR ALGORITHM

5.1. Approach 1

1. Take the input answer and the answer key and store in sent1 and sent2 variables
2. tokenize1=word_tokenize(sent1)
3. tokenize2=word_tokenize(sent2)
4. Obtain the basic English stop words
   stop_words = set(stopwords.words('english'))
5. Filter the sentences by removing the stop_words
   filtered_sentence1 = [w for w in tokenize1 if not w in stop_words]
   filtered_sentence2 = [w for w in tokenize2 if not w in stop_words]
6. Tag the filtered sentence
7. tagged_sentence1=nltk.pos_tag(filtered_sentence1)
   tagged_sentence2=nltk.pos_tag(filtered_sentence2)
8. convert the word and tag as key and value pairs
9. dict_1=dict(tagged_sentence1)
   dict_2=dict(tagged_sentence2)
10. for w in dict_1.keys()
11. Assign the tag values to x
    x=dict_1[w]
12. x=x.lower()
13. Take the first letter into consideration x=x[0]
14. Check if(x=='n') to take into consideration only nouns
    try:
    w=w+"."+x+".01"

print(w)
k=wordnet.synset(w)
print(k)
except nltk.corpus.reader.wordnet.WordNetError:
print("ERROR")

15. Carry out the step 6. For dict_2 as well
16. Check if(x==x1=='n') to ensure only nouns are being compared
    a. carry out similarity comparisons between the sysnsets and assign to f
    b. increment count to keep track of the number of comparisons
    c. add the similarity rating to a total ,total=total+f
17. Display the total/count to get the average value

5.2. Approach 2

1. Scrape the answers from the web from the appropriate website and store in an arbitrary file
2. Open the arbitrary file open("scrape1.txt") and append the text in the file to sent1
3. To train the model the wrong answers must also be used so another document containing the wrong answers open("wrong.txt") and append the text in the file to sent2
4. sent3 consists of the answer inputted
5. create documents = [(words, "correct")for words in word_tokenize(sent1)+(words, "wrong")for wordss in word_tokenize(sent3)]
6. assign all_words = []
7. for w in word_tokenize():
   all_words.append(w.lower())

8. obtain all_words = nltk.FreqDist(all_words)
9. create features word_features = list(all_words.keys())
10. def find_features(document):
    words = set(document)
    features = {}
    for w in word_features:
    features[w] = (w in words)

    return features
11. Create training_set and testing_set from
12. featuresets = [(find_features(rev),category) for (rev, category) in documents]
13. classifier = nltk.NaiveBayesClassifier.train(training_set) print("Classifier accuracy percent:",(nltk.classify.accuracy(classifier, testing_set))*100)
14. for every word in sent3 calculate classifier.classify(find_features(sent2[i])
15. Obtain the final value by taking into consideration the classification

5.3. Approach 3

1. Create the raw documents list containing the answer key or answer keys
2. Assign gen_docs to [[w.lower() for w in word_tokenize(text)]for text in raw_documents]
    a. Create a vocabulary dictionary ,dictionary =gensim.corpora.Dictionary(gen_docs)
3. Create the bag of words ,corpus = [dictionary.doc2bow(gen_doc) for gen_doc in gen_docs]
4. Create the tf_idf model ,tf_idf = gensim.models.TfidfModel(corpus)
5. Obtain similarity,sims = gensim.similarities.Similarity("location_of_project",tf_idf[corpus],num_features=len(dictionary))
6. Input the answer ,u_inp=str(input("QUESTION TO BE ASKED"))
7. query_doc = [w.lower() for w in word_tokenize(u_inp)]
8. query_doc_bow = dictionary.doc2bow(query_doc)
9. query_doc_tf_idf = tf_idf[query_doc_bow]
10. depending on the requirement 12. Or 13 is carried out
11. obtain the marks accorsingly to display ,if only one key is considered ie,
12. raw_document=["","answer key"]
    s1=list((sims[query_doc_tf_idf]))
    print(s1)

    if(s1[1]>0.9):
    print(5)
    elif(s1[1]>0.8):
    print(4)
    elif(s1[1]>0.7):
    print(3)
    elif(s1[1]>0.6):
    print(2)
    elif(s1[1]>0.5):
    print(1)
    else:
    print(0)
13. obtain the marks to display if only one key is considered ie,raw_document=["answer key 1","answer key 2"," answer key 3"…]
    s1=list((sims[query_doc_tf_idf]))
14. Display the marks depending on the max similarity value to the answer keys given in s1.

6. IMPLEMENTATION

The proposed algorithm is implemented using python:

```
import gensim
raw_documents = ["","ANSWER KEY"];
from nltk.tokenize import word_tokenize
gen_docs = [[w.lower() for w in word_tokenize(text)]
            for text in raw_documents]
dictionary = gensim.corpora.Dictionary(gen_docs)
corpus = [dictionary.doc2bow(gen_doc) for gen_doc in
gen_docs]
tf_idf = gensim.models.TfidfModel(corpus)
print(tf_idf)
sims = gensim.similarities.Similarity('LOCATION OF
DOCUMENT'',tf_idf[corpus],num_features=len(dictionary)
)
u_inp=str(input("QUESTION TO BE ASKED"))
query_doc = [w.lower() for w in word_tokenize(u_inp)]
query_doc_bow = dictionary.doc2bow(query_doc)
query_doc_tf_idf = tf_idf[query_doc_bow]
s1=list((sims[query_doc_tf_idf]))
print("THE MARKS OBTAINED IS :")
#here we are considering a 5 marker question

if(s1[1]>0.9):
    print(5)
elif(s1[1]>0.8):
    print(4)
elif(s1[1]>0.7):
    print(3)
elif(s1[1]>0.6):
    print(2)
elif(s1[1]>0.5):
    print(1)
else:
    print(0)
```

7. MATHEMATICAL DISCUSSION

$$\mathbf{tf}(t, d) = \frac{f_d(t)}{\max_{w \in d} f_d(w)}$$

$$\mathbf{idf}(t, D) = \ln\left(\frac{|D|}{|\{d \in D : t \in d\}|}\right)$$

$$\mathbf{tfidf}(t, d, D) = \mathbf{tf}(t, d) \cdot \mathbf{idf}(t, D)$$

$$\mathbf{tfidf}'(t, d, D) = \frac{\mathbf{idf}(t, D)}{|D|} + \mathbf{tfidf}(t, d, D)$$

$f_d(t) :=$ frequency of term t in document d

$D :=$ corpus of documents

$$\text{posterior probability} = \frac{\text{conditional probability} \cdot \text{prior probability}}{\text{evidence}}$$

$$P(\omega_j \mid \mathbf{x}_i) = \frac{P(\mathbf{x}_i \mid \omega_j) \cdot P(\omega_j)}{P(\mathbf{x}_i)}$$

Let

- $\mathbf{x}_i$ be the feature vector of sample i, $i \in \{1, 2, \ldots, n\}$,
- ω_j be the notation of class j, $j \in \{1, 2, \ldots, m\}$,
- and $P(\mathbf{x}_i \mid \omega_j)$ be the probability of observing sample $\mathbf{x}_i$ given that is belongs to class ω_j.

8. COMPARISON AND LIMITATIONS

Algorithm 1 is a primitive algorithm. The results obtained by this algorithm are not satisfactory as it only removes the stops words and compares the keywords. It doesn't not consider the weightage of the keywords nor does it consider the verbs and connecting words thereby not understanding the underlying meaning of the statement.

Algorithm 2 is a different approach where we train the model by using data obtained by scraping the web and classifying this data set using Naïve Bayes algorithm. The limitation to this approach is the possibility of large number of false positives due to unnecessary keywords present on the websites can lead to a skewed and biased classifier.The Naïve Bayes algorithm is more of a classifier not an analyzer.

Algorithm 3 is the final approach which gives a satisfactory result. One drawback that can be pointed out is that the evaluation may not be up to the mark of certain evaluators due to their rigid requirements and also allocation of threshold values may vary from question to questions.

ALGORITHM	TYPE OF LEARNING	ACCURACY IDEALLY OBTIANED	TRAINING COST	TYPE OF CLASSSIFIER
Algorithm 1	Supervised	50%	High training cost	-
Algorithm 2	Supervised	75%	High training cost	Linear classifier
Algorithm 3	Unsupervised	93%	No prior training	Vector space classifier

9. ALTERNATIVE APPROACHES AND OVERCOMING LIMITATIONS

a. Word2vec

Word2vec is a cluster of models that are related that are mainly used to generate word embeddings. These models are shallow neural networks that consist of 2 layers. Word2vec takes a large corpus of text as it's input and produces a vector space, usually of several hundred dimensions, with each distinct word in the corpus being assigned a corresponding vector that can be added, subtracted and manipulated like any other vector in space. Word vectors are placed in the vector space in such a way that words that share common contexts in the corpus are located close to to one another in the space.

Word2Vec is very useful in the analysis of unstructured data. This tool learns the relations that the terms have with each other, based on the frequencies with which the terms occur together. These patterns get encoded into a matrix.

b. Tensorflow

TensorFlow is an open source software library for dataflow programming for a variety of tasks. TensorFlow works by creating a graph. A graph has a set of nodes and edges between the nodes. The edges on TensorFlow graphs are multidimensional data arrays and they are called as Tensors.

During text analysis, there are two types of data that are dealt with - categorical data and continuous data. For categorical data, the values cannot be outside a fixed set of values whereas for continuous data, the values are not constricted to a fixed set of values. In order to perform prediction of data, both of these types of data are required. Every tensor has a dimension attached to it. The number of dimensions is called the rank of the tensor. TensorFlow is used extensively for deep learning.

c. Google Dialogflow

Dialogflow (formerly Api.ai) is a tool that develops human–computer interaction technologies based on natural language conversations.The SDK's contain voice recognition, natural language understanding, and text-to-speech. Dialogflow

provides a web interface to build and test conversation scenarios by creating chatbots.

d. IBM Watson

IBM developed a system that is capable of answering questions that are asked in Natural Language. With a few simple API calls, you can use Watson Natural Language Understanding to automate NLP and machine learning capabilities within your application. Concepts are analyzed using text extraction.

10. CONCLUSION

In this paper, a novel approach for the evaluation and analysis of answers is presented. The algorithm 3 is built up to an implementational level augmenting the first and second algorithms with more powerful tools and libraries. The algorithm implemented is also very flexible and can be modified as per the evaluator's requirements. It doesn't require previous training data and can work on the answer key provided itself.

The approach developed is very advantageous and the outputs provided are satisfactory with minimal implementation errors.

However, stronger tools, API 's and higher-level approaches can be implemented to get an optimal solution, as shown above. Various AI and machine learning products like Dialogflow , Tensorflow , IBM Watson are used for higher implementational level applications.

REFERENCES

https://pythonprogramming.net/tokenizing-words-sentences-nltk-tutorial/

http://www.ijmlc.org/vol5/523-C013.pdf

https://www2.le.ac.uk/Members/rjm1/talent/book/c3p2.html

https://www.analyticsvidhya.com/blog/2015/10/beginner-guide-web-scraping-beautiful-soup-python/

http://www.tfidf.com/

https://radimrehurek.com/gensim/

https://media.readthedocs.org/pdf/gensim/stable/gensim.pdf

https://machinelearningmastery.com/gentle-introduction-bag-words-model/

https://www.ibm.com/cloud/watson-natural-language-understanding

https://sourcedexter.com/what-is-tensorflow/

https://sourcedexter.com/tensorflow-text-classification-python/

https://www.linkedin.com/pulse/short-introduction-using-word2vec-text-classification-mike

https://dialogflow.com/docs/getting-started/basics

https://www.ibm.com/watson/products-services/

http://sebastianraschka.com/Articles/2014_naive_bayes_1.html