Evaluation of Semantic Textual Similarity Approaches for Automatic Short Answer Grading

R&D Report

Author:

Ramesh Kumar[*]

Advisors:

Prof. Dr. Paul Plöger[○]

M. Sc. Deebul Nair[•]

January 31, 2018

Bonn-Rhein-Sieg University of Applied Sciences

Master of Autonomous Systems

∗ kumar.kumar@smail.inf.h-brs.de

○ paul.ploeger@h-brs.de

• deebul.nair@mail.inf.h-brs.de

Declaration

I, **Ramesh Kumar**, declare that this research and development project titled, *Evaluation of Semantic Textual Similarity Approaches for Automatic Short Answer Grading* and the work presented is original and is not submitted to any other institution before.

Signature: ________________________________

Date: ________________________________

Acknowledgements

This research and development project would have been incomplete without the guidance and help of some great people. I would like to express my gratitude to M.Sc. Deebul Nair for his countless support throughout the project. I would also like to thank Luis Octavio Arriaga Camargo for guiding me during my work. Thanks to Aleksandar Mitrevski and Santosh George Thoduka for providing a dataset of Mathematics and Robotics Control(MRC) course. Special thanks to Sivasurya Santhanam for reviewing this report.

Lastly, I am really thankful to my parents and friends for their endless love and support.

Abstract

Semantic textual similarity computes the equivalence of two sentences on the basis of its conceptual similarity. It is widely used in natural languages processing tasks such as essay scoring, machine translation, text classification, information extraction, and question answering. This research and development project focuses on one of the applications of semantic textual similarity; automatic short answer grading. Automatic short answer grading(ASAG) system assigns a grade to a response provided by a student by comparing with one or more model answers.

In particular, we selected one of the state-of-the-art short answer grading approaches implemented using Stanford CoreNLP library, re-implemented with the help of two open source libraries; NLTK and Spacy, and evaluated on the Texas dataset. Additionally, an in-house benchmarking ASAG dataset; Mathematics for Robotics and Control(MRC) course dataset, was also collected and used for comparative evaluation. All the implementations were evaluated on metrics such as Pearson correlation coefficient, root mean square error(RMSE), and runtime.

Results on Texas dataset showed that Stanford CoreNLP library has better Pearson correlation coefficient(0.66) and lowest RMSE(0.85) than NLTK and Spacy libraries. While on MRC dataset, all 3 libraries showed the comparative results on Pearson correlation coefficient, RMSE, and runtime.

Contents

List of Figures

List of Tables

Acronyms

Acronym		Meaning
ASAGs	=	Automatic Short Answer Grading Systems
ASAP	=	Automated Student Assessment Prize
ATM	=	Automated Text Marker
C-rater	=	Concept-Rater
LSA	=	Latent Semantic Analysis
MRC	=	Mathematics for Robotics and Control
NER	=	Named Entity Recognition
NLP	=	Natural Language Processing
NLTK	=	Natural Language ToolKit
NP	=	Noun Phrase
POS	=	Part of Speech
RMSE	=	Root Mean Square Error
SemEval	=	Semantic Evaluation
SVD	=	Singular Value Decomposition
T-SNE	=	t-Distributed Stochastic Neighbor Embedding
TF-IDF	=	Term Frequency-Inverse Document Frequency
VP	=	Verb Phrase
word2Vec	=	Word to Vector

Chapter 1

Introduction

Assessment of student's responses plays important role in the educational process, as it determines whether the goal of education is being meet or not. However, time spent on assessing responses manually takes more than instructing them; as [8] shows that almost 30% of British teacher's time is spent only for grading. Beside this, there is also the possibility of errors and unfairness due to bias, fatigue or lack of consistency[9][10] .

With the advancement of technologies such as natural language processing, it is possible to grade electronic student's responses automatically that can save time, cost and maintain accuracy. Furthermore, a difference between multiple choice questions(MCQs) and short answer questions is easy to understand; MCQs can have only one correct option. While short answer questions can have multiple correct answers. On the other hand, the difference between short answer questions and essays become vague. According to Burrows et al. [2], any short answer question should satisfy at least five particular criteria: First: the question should require a response from the student that needs to recall external knowledge rather than the response to be determined by the question. For example, the question should not be like: "How many states does Germany have? 16 or 20?", as the answer is already present in the question. Secondly, the question should require an answer to be in the natural language. Thirdly, the response of question can vary from one sentence to one paragraph. Fourth, assessment of response should be considered on content basis rather than writing style. Lastly, level of openness between open-ended and closed-ended responses should be limited based on the design of question.

In this research and development project, we select one of the state-of-the-art short answer grading approaches implemented using Stanford CoreNLP library, re-implement with the help of two open source libraries; NLTK and Spacy, and evaluate on the Texas dataset. Additionally, an in-house benchmarking ASAG dataset was also collected and used for comparative evaluation. All the implementations were evaluated on metrics such as Pearson correlation coefficient, root mean square error(RMSE), and runtime.

This work is divided into different sections: the first section gives an introduction about Natural Language Processing; its basic concepts and libraries that we use throughout our project. The second section contains the review of the state-of-the-art approaches for automatic short answer grading systems(ASAG) introduced during different eras. The third section gives detailed information about an approach that we evaluate using two libraries on a publicly available dataset and in-house created dataset. The Fourth section provides the description of datasets that we use for this task. The last section contains evaluation and results of datasets using Spacy and NLTK libraries.

Chapter 2

Natural Language Processing

Natural language processing(NLP) is a field of computer science that allows computers to understand natural languages such as English, Chinese, German, Spanish, etcetera. There are numerous applications of natural language processing such as machine translation, spam detection, information extraction, sentiment analysis, text summarization, automatic short answer grading of essays and short answers, and so on. This is possible if machines are able to understand the rules of a natural language using techniques such as syntactic dependencies, constituency dependencies, part of speech, word and sentence segmentation, stemming, lemmatization, and so on. Furthermore, this field has various challenges in solving given applications such as ambiguity in sentences; words can have more than one meaning, idioms have different meanings, non-standard English used in social media; e.g: @Ronaldo, #neversaynever as models are completely trained on data from newspapers or Wikipedia, segmentation issues of sentences; abbreviations such as M.E, Ph.D. have punctuations in it, this may be problematic for computers to perform accurate segmentation [1].

Following section introduce some basic concepts in natural language processing, that are used in later chapters:

2.1 Text pre-processing

Text pre-processing is a process of converting text into standard form so that it can be used for further operations such as sentence similarity, spell checking, etcetera. [1] shows that, before processing any text, at least three common steps are performed:

- Sentence segmentation
- Word tokenization
- Text normalization

Sentence segmentation

It is a process of chopping the text into sentences, called segments or tokens. For a given text, it considers most useful signs such as periods(.), question marks(?) and an exclamation point(!) to divide into sentences. However, there are challenges in segmenting such as periods can represent an abbreviation, decimal point or an email address but not the end of the sentences.

[1] shows that, to determine whether periods are part of a word or sentence separator, sentence tokenization methods use one of following approaches:

- Machine learning(unsupervised and supervised)
- Rule-based

Word tokenization

Word tokenization is a process of splitting the text into word/token. Tokens do not contain the whitespace characters, but punctuations such as periods(.), comma(,), question mark(?), etcetera [1]. Furthermore, there are challenges such as:

- San Francisco is one word, but tokenization process consider as two words.
- Germany's is one word, but tokenization return as two words.
- This needs to be defined according to standard, how we want to tokenize them.
- Tokenization becomes very complicated in languages such as Chinese and Japanese, where words do not have spaces. In that case, input words are compared with dictionary words and tokenized accordingly [11].

Text normalization

To process the natural language text, it is necessary to normalize it by converting characters of words to lowercase, removing stop words, determining lemmatization and stemming of words, or representing it in canonical form. So, that other operations such as string and semantic similarity can be performed easily. Few possible list of operations that can be applied during normalization is discussed in the following section.

Convert characters to lowercase

In this operation, we convert characters of tokenized words as the lower case so that we can use them for operations such as spell checking, lemmatization, stemming, string similarity, etcetera.

Remove stop words

Stop words are those that do not have much meaning such as “the”, “a”, “of”, “in”, “on”, and so on. It is necessary to remove these words to save space and make processing faster. On the other hand, there are some search engines that do not remove stop words as they can be useful for some queries. Such as, when we want to perform an exact match between two pieces of text then removal of stop words can cause a problem [12].

Lemmatization and Stemming

Lemmatization and stemming is a process of reducing inflectional and derivationally related forms of a word in simplest or common form. In other words, it is the process of determining words that have the same roots, regardless of their surface differences. For example, words “going”, “gone”, “goes”, all share same roots, that is “go”. Furthermore, the difference between lemmatization and stemming is that, stemming is a crude heuristic process that performs the basic operation on strings by removing ending of words. While lemmatization use vocabulary and morphological analysis to determine root form and result is always root form of the word present in the dictionary.

However, it depends upon the application, whether to use lemmatization or stemming. For suppose, if the speed of application is important, then we should use stemming, as lemmatization have to search through corpus, while, stemming do simple operations on a string. If we want to return always dictionary words, then lemmatization is helpful [1][13].

2.2 String Similarity

It determines the similarity between two text strings based on string matching regardless of their meaning. Furthermore, these metrics are normally used to check spelling mistakes by comparing two strings. For example, ”lame” and ”same” are considered to be pretty much close. This distance between strings is computed using string similarity algorithms such as minimum edit distance or Levenshtein distance, N-gram, Hamming distance, Jaccard similarity, etcetera [14]. Following section briefly discuss two most common algorithms; minimum edit distance and N-gram. Later on, we will also see how both algorithms can be used for spell-checking.

Minimum Edit Distance

Minimum edit distance is widely known an algorithm for string similarity. It computes the minimum number of operations needed to transform one string into another. To do this, three editing operations are performed:

- Insertion(i)

- Deletion(d)
- Substitution(s)

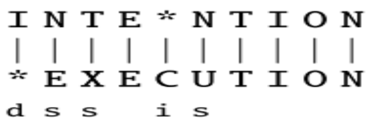


Figure 2.1: [1] Comparison between two strings

Figure 2.1 shows the comparison of two strings “INTENTION” and “EXECUTION”. In order to transform one string into another, we need to consider the cost of each operation. Therefore, if we take the cost of each operation as 1, the distance between two strings is 5.

To find the minimum edit distance, we can consider it as a searching task, where we want to find the shortest path to compute the number of operations required to transform one string to another. So problem formulation can be written as:

- Initial State: Word that is to be transformed.
- Operations: Insertion, deletion, and substitution.
- Goal State: Word that we are trying to get.
- Path Cost: Minimize number of editing operations.

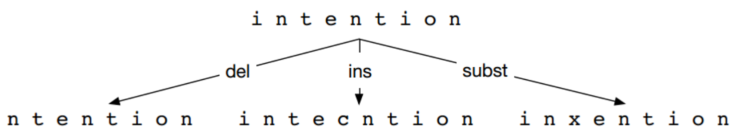


Figure 2.2: [1] Finding the minimum edit distance considered as search problem

The figure 2.2 shows how minimum edit distance is viewed as a search problem. Since space of all possible edits will be very huge, therefore, it will be difficult and time-consuming to explore entire search space. However, many of different paths might end in the same final state(target word), therefore, instead of re-computing all those paths, we can simply store the optimal path to a state during each time, which is possible by using dynamic programming [1].

How Minimum Edit Distance is used for Spell correction?

- Given a sentence, we check each word, whether it is present in English dictionary using the PyEnchant library.

- If it is not, that means the word is misspelled, we take a list of suggested words based on incorrect/misspelled word using the PyEnchant library.
- Compute minimum edit distance between incorrect word and each suggested word present in the list.
- Replace a misspelled word with a suggested word having least edit distance.

N-Grams

It is another most commonly used algorithm for string similarity. N-grams are a sequence of variable characters in a word or sequence of words in a sentence. Furthermore, they are used for the purpose of spell correction, word breaking, and text summarization. N-grams are computed by considering all words in a window and we move one or more word forward, depending on the gram of word/sentence we want. For suppose, given a sentence "The quick brown fox jumps over a lazy dog", if N = 2 (bi-grams), then bi-grams of a sentence can be computed as:

- The quick
- quick brown
- brown fox
- fox jumps
- jumps over
- over a
- a lazy
- lazy dog

To find number of grams in a sentence/word, we can use formula

$$Grams = X - (N - 1) \tag{2.1}$$

Where X = number of words in sentence or number of character in word and N is value of gram [15].

How N-Grams are used for Spell correction?

- Find misspelled words using the PyEnchant library.
- Get list of suggested words based on misspelled word with the help of PyEnchant library.

- Filter list of suggested words within certain minimum edit distance. We do this, because a list of suggested words can be very big, so, we remove words that have highest minimum edit distance.
- Compute N-gram of misspelled word and each suggested word from a list.
- Compute Jaccard coefficient of misspelled word and each suggested word, which is given as:

  $$jaccard - coefficient = \frac{A \cap B}{A \cup B} \tag{2.2}$$

  its value varies between 0 and 1, where 0 represents terms are completely dissimilar and 1 denotes terms are exactly similar.
- Replace suggested word having highest Jaccard coefficient.

2.3 Part of Speech(POS) tagging

Part of speech tagging is a process of determining part of speech tags for each word present in the text. It is also called as sequence labeling problem, as task is to map the sentence x_1, x_2,..., x_n to a sequence of tags y_1, y_2,..., y_n. Furthermore, it gives us information about a word and its neighbors. For suppose, if a word is a noun then it is obvious that its neighbors are adjectives and determiners, as in the majority of cases nouns are preceded by adjectives and determiners. Moreover, it is also useful for finding named entities such as organizations, persons, locations, etcetera present in the text. To compute POS tags of text, first, word tokenization is performed on the given sentence. After that, we use tagging algorithms such as Baseline tagger, Maxtent POS tagger, Maximum Entropy Markov Models(MEMM), or Bidirectional dependencies; where inputs are tokenized words with their possible POS tags and output is a single best tag for each tokenized word. In addition to this, tagging is considered as disambiguation task, as it is possible for a word to have more than one POS tag. Such as word ”book” can be a verb(book that hotel) or a noun(give me that book), and the challenge is to find best possible tag according to context [16]. Lastly, modern language processing techniques for English use Penn Treebank tagset that has 45 POS Tags as shown in table 2.1

Table 2.1: [1] List of possible POS tags in Penn Treebank TagSet

Tag	Description	Example	Tag	Description	Example
CC	coordin. conjunction	and, but, or	**SYM**	symbol	+, %, &
CD	cardinal number	one, two	**TO**	"to"	to
DT	determiner	a, the	**UH**	interjection	ah, oops
EX	existential 'there'	there	**VB**	verb base form	eat
FW	foreign word	mea culpa	**VBD**	verb past tense	ate
IN	preposition/sub-conj	of, in, by	**VBG**	verb gerund	eating
JJ	adjective	yellow	**VBN**	verb past participle	eaten
JJR	adj., comparative	bigger	**VBP**	verb non-3sg pres	eats
JJS	adj., superlative	wildest	**VBZ**	verb 3sg pres	eats
LS	list item marker	1, 2, One	**WDT**	wh-determiner	which, that
MD	modal	can, should	**WP**	wh-pronoun	what, who
NN	noun, sing. or mass	llama	**WP$**	possessive wh-	whose
NNS	noun, plural	llamas	**WRB**	wh-adverb	how, where
NNP	proper noun, sing,	IBM	**$**	dollar sign	$
NNPS	proper noun, plural	Carolinas	**#**	pound sign	#
PDT	predeterminer	all, both	**“**	left quote	‘ or “
POS	possessive ending	's	**”**	right quote	’ or ”
PRP	personal pronoun	I, you, he	(	left parenthesis	[, (, <
RB	adverb	quickly, never	**,**	comma	,
RBR	adverb, comparative	faster	**.**	sentence-final punc.	. ! ?
RBS	adverb, superlative	fastest	**:**	mid-sentence punc.	: ; ... -
RP	particle	up, off			

2.4 Named Entity Recognition(NER)

Named entity recognition is a process of identifying the words present in the text belonging to pre-defined categories such as the name of persons, organizations, locations, currencies, quantities, etcetera [17]. Furthermore, NER is used in many applications such as: question answering systems, textual entailment, and information extraction. It can be helpful for question answering systems, as questions like "who did that...? ", "where is Bonn?", mostly have answers as the name of persons, locations, etcetera. Although, there are lot of challenges such as most of the named entities are not single words rather than group of words such as Hochschule-Bonn-Rhein-Sieg or Commerce Bank, therefore, named entity models specifically Maximum Entropy Markov Model or Conditional Random Fields need to determine whether the group of words belong to same entity or not [18].

2.5 Parse Tree

It represents syntactic structure of text based on context-free grammar. In other words, it is a technique that helps computers to understand the grammatical structure of sentences based on linguistic rules. Furthermore, they are usually formed based on the relation of constituency grammars or dependency grammars [19]. Following section, briefly describes the two types of parse trees:

Constituency Parse Tree

This type of tree breaks down the text into sub-phrases such as Noun phrase(NP), Verb phrase(VP), etcetera. In other words, it tells the relationship between the sub-phrases present in the text [19]. For example, given a sentence: "Four men died in an accident", constituency tree is given in figure 2.3, where we see the relation between sub-phrases.

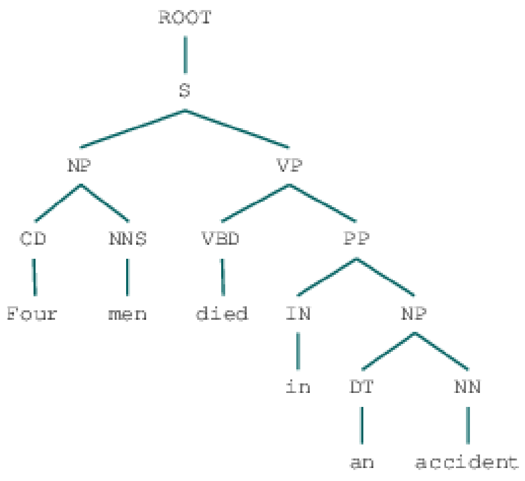


Figure 2.3: Constituency tree shows relationship between sub-phrases of a given sentence

Dependency Parse Tree

This is another type of parse tree that expresses the relationship between the words present in the sentence. Furthermore, each word has its vertex and there is always the child-parent relationship between the words [19]. Figure 2.4 shows dependency tree of sentence: “I read the book”, where our parent word is “read”, and children are “I” and “the book”. In addition to this, “read” and “I” have a relation of a nominal subject(nsubj), whereas “read” and “the book” have a relation of direct object(dobj). Table 2.2 shows some possible dependency relations between words.

Later on, we also use dependency parse tree to determine the similarity between two sentences based on children and parent relations.

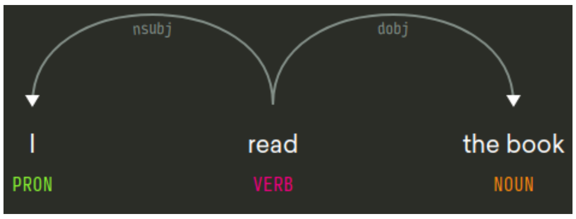


Figure 2.4: Dependency tree generated using Spacy library of a given sentence

Table 2.2: [6] List of some possible dependency relations between words

Tag	Description
acomp	adjectival complement
advcl	adverbial clause modifier
advmod	adverb modifier
agent	agent
amod	adjectival modifier
aux	auxiliary
auxpass	passive auxiliary
cc	coordination
ccomp	clausal complement
conj	conjunct
csubj	clausal subject
dep	dependent
det	determiner
dobj	direct object
iobj	indirect object
neg	negation modifier
nn	noun compound modifier
poss	possession modifier
prep	prepositional modifier
rcmod	relative clause modifier
root	root
tmod	temporal modifier
xcomp	open clausal complement

2.6 Word Embedding

Word embedding is the mapping of words or phrases using a dictionary to a vector of real numbers. This is necessary as most of machine learning algorithms are unable to process raw text, to perform classification, regression, or any other tasks. In addition to this, these vectors are the low-dimensional representation of words built from the large corpus of text, and also maintains the contextual similarity of words [20]. They are useful for many applications including grouping of related words, document clustering, determine similarities between words, used as features for text classification and so on [21].

One way of generating word vectors is to use word2Vec model. It consists of a single hidden layer with all linear neurons and a fully connected output layer that has soft-max activation function. The number of inputs to network corresponds to amount of words in vocabulary for training. The number of hidden neurons are used to set dimension of output word vectors, and size of output layer corresponds to a number of inputs to the input layer. Figure 2.5 shows architecture of Word2Vec model.

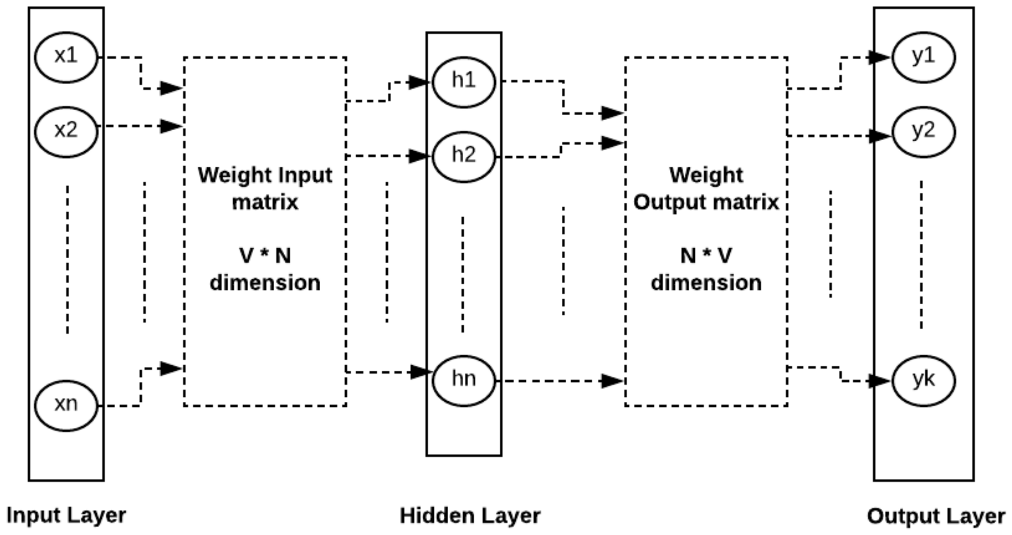


Figure 2.5: [1] Architecture of word2vec model

Let's consider the training corpus having following sentences:

"Tiger saw a cat", "Tiger catch the cat","The cat climbed to hill"

If we sort them alphabetically and consider only unique words, the vocabulary size is 8; [a, cat, catch, climbed, hill, saw, Tiger, the]. So, the number of inputs and outputs of a network is 8. Assume we just want 3 hidden neurons, so our input and output weight matrices are 8*3 and 3*8 respectively. Initially, all weights are initialized randomly, and later on, they are adjusted based on an error between current and desired output. For time being, let's assume we want to learn the relationship between two words; "Tiger" and "cat". So, in terms of word embedding, "Tiger" is known as "context word" and "cat" is considered as "target/desired word". For this example, our input vector; X = $[0, 0, 0, 0, 0, 0, 1, 0]^t$ and desired vector is $[0, 1, 0, 0, 0, 0, 0, 0]^t$. If we compute the output of hidden layer neurons, it looks like:

$$H = X^t W_{input} = \begin{bmatrix} 0.0123 & -0.132 & 0.435 \end{bmatrix}$$

Where X is input and WI is weight input matrix. Since activation function of hidden layer is linear, therefore values for the product between input and weight matrix are considered randomly, just to make things simpler.

Similarly, vector for output layer is written as:

$$HW_{output} = \begin{bmatrix} -0.4533 & 0.2342 & 0.0034 & -0.1503 & 0.2324 & 0.0213 & 0.0139 & -0.2420 \end{bmatrix}$$

Where, H is an output of hidden layer and W_{output} is weight matrix of the output layer. Again, we take values of product randomly.

Since our goal is to show probabilities of words, therefore we use soft-max activation function, so that sum of neurons output in final layer sums up to one. It is given as:

$$y_k = \frac{e^x}{\sum_{n=1}^{V} e^x} \tag{2.3}$$

where, V is vocabulary and x is element in product of HW_{output}

So, probabilities of eight words in corpus look like:

$$\begin{bmatrix} 0.0949 & \mathbf{0.1291} & 0.1144 & 0.1448 & 0.1195 & 0.1117 & 0.1498 & 0.2519 \end{bmatrix}$$

Again, we take values of probabilities randomly, just to explain how things work. Since the probability of target word in bold fonts is less than probabilities of other words, therefore, we compute error between current output and target vector. After that, weights are adjusted based on back-propagation, till error is minimized. So, in this way, we use word2vec model to learn relationships between words [22].

However, there are publicly available trained models for word embeddings, since we follow an approach of Sultan et al. [7], therefore, we use model provided by Baroni et al. [23] trained using word2vec. Where, we get the resultant vector of 400 dimensions for each queried word. The reason word2vec model is employed because it outperforms other word embeddings method such as frequency-based embeddings as shown in Baroni et al. [23].

2.7 Libraries

Libraries are a collection of pre-compiled and non-volatile resources that help programmers to develop desired software. These libraries can include packages, classes, functions, data structures, templates, documentation, etcetera [24]. Every library has its application that helps users in particular area. For Example, NumPy library enables users to perform tasks on multi-dimensional arrays and matrices. Since in our case, we want to perform tasks on human language, therefore there are many available libraries such as Mallet, Open NLP, Stanford CoreNLP toolkit, Spacy, Natural Language Toolkit(NLTK) and so on. Most of these are written in Java programming language, but only Natural Language Toolkit(NLTK) and Spacy are written in Python. Since, for short answer grading, we use IPython Notebook that is completely based on python, therefore, to make our tasks easier, NLTK and Spacy are employed for this research and development project. Furthermore, we also use the spell-checking library, known as PyEnchant, to compare the performance of spell correction using minimum edit distance and n-grams. Following section briefly gives an overview about these libraries:

Natural Language Toolkit(NLTK)

Natural language Toolkit(NLTK) is an open-source and free platform to perform natural languages tasks. It is completely based on Python and has a variety of natural language processing tools, datasets, algorithms, corpora and lexical resources such as WordNet. Since it has dozens of algorithms, different corpora, therefore it is intended for research and educational purposes [25][26]. Furthermore, It supports all tasks for the English language, however, only a few tasks such as stemmer support around 17 languages; Danish, Dutch, German, Finnish, Italian and so on [27].

We use this library as it has already implemented basic natural language concepts such as tokenization, stemming, lemmatization, POS tagger, named entity recognition, stop words, minimum edit distance, n-grams, parse trees, and so on.

Spacy

This is another open-source tool for processing language tasks. It also provides similar functionality for NLP tasks as other libraries, but with more accurate results and speed, like it provides fastest syntactic parser, tagging and fast entity recognition model among all libraries [28][29][30]. In addition to this, it also provides language models for multiple languages such as English, Spanish, German, French, Italian and Dutch, that supports for POS tagger, parser and entity recognition.

If we do the comparison between NLTK and Spacy then unlike NLTK, Spacy is more application-oriented that help users for processing and analyzing large text data rather than for research and educational purposes. Furthermore, unlike NLTK, it also does not allow users to select multiple algorithms that give similar functionality [31].

Stanford CoreNLP

Stanford CoreNLP is a toolkit released by the Natural language software research group of Stanford University. It is written in Java programming language and offers modules for basic and advanced natural language processing tasks such as lemmatization, parts of speech, parse trees, named entity recognition, sentiment analysis, and so on. Furthermore, it supports the variety of languages such as Arabic, Chinese, English, French, German and Spanish.

For our research and development project, we use an existing approach that is already implemented using Stanford CoreNLP library. While we use, NLTK and Spacy to compare results obtained with Stanford CoreNLP library [32].

PyEnchant

PyEnchant is spell-checking library based on Enchant library. It offers all functionalities of Enchant library in Python and has built-in English dictionary, where, one can check incorrect and suggested words based on the queried word [33].

Tools

Nbgrader

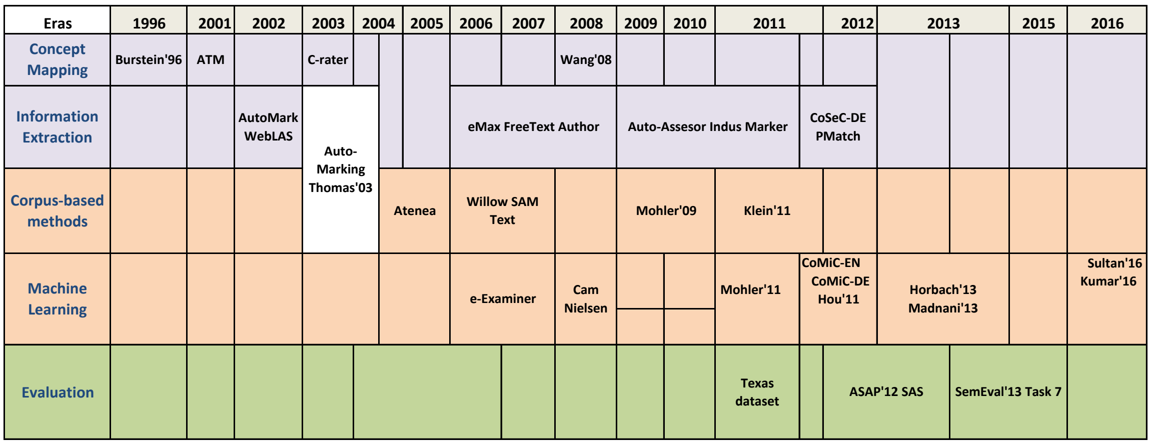
Nbgrader is a tool that allows users to create and grade assignments in the Ipython Jupyter notebook. This tool helps teachers to create assignments based on coding exercises and text free-responses. Furthermore, it includes databases, where students can submit their assignments and teacher can give them feedback. Apart from this, it also allows instructors to write the test cases of solutions and later on, test cases are automatically compared with the student's answers to check whether test cases are fulfilled or not [34].

For this research and development project, we will also use this tool, as it makes the task of auto-grading easier.

Chapter 3

Related Work

This chapter provides the broad review of existing automatic short answer grading systems(ASAGs). Research shows that there are more than 30 automatic short answer grading systems(ASAGs) categorized into 5 different eras of methodology and evaluation [2]. Furthermore, figure 3.1 show time periods and names of most of the systems introduced in each era. First four eras are method-based and the last era is evaluation-based.



Eras	1996	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013	2015	2016
Concept Mapping	Burstein'96	ATM		C-rater					Wang'08							
Information Extraction			AutoMark WebLAS	Auto-Marking Thomas'03			eMax FreeText Author				Auto-Assesor Indus Marker		CoSeC-DE PMatch			
Corpus-based methods						Atenea	Willow SAM Text			Mohler'09		Klein'11				
Machine Learning							e-Examiner		Cam Nielsen			Mohler'11	CoMiC-EN CoMiC-DE Hou'11	Horbach'13 Madnani'13		Sultan'16 Kumar'16
Evaluation												Texas dataset	ASAP'12 SAS	SemEval'13 Task 7		

Figure 3.1: [2] The eras of automatic short answer grading systems(ASAGs)

The following section gives the description of different eras in ASAGs:

3.1 Era of Concept Mapping

In this era, systems work with the mapping of concepts between student's and instructor's answer. Answers provided by student and instructor consist of several concepts, and grades are assigned based on presence and absence of each concept [2]. Furthermore, questions designed for these systems can have one or more correct answers related to a particular topic, rather than open-ended answers that include personal experiences, suggestions, etcetera [35]. Based on the number of common concepts between student's and instructor's answer, the final score is assigned. This era includes the systems such as Burstein'96, Automatic Text Marker(ATM), Concept-Rater(c-rater), and Wang'08. The following below gives the introduction of the most popular system of this era:

Automated Text Marker(ATM)

The figure 3.2 shows proposed architecture of ATM prototype. For a given model and student answer, first grammar of both the answers is checked by syntax analyzer provided by [36]. After that, semantic analysis of both answers is determined based on dependencies. Later on, comparison module is used to find what concepts are similar and final score is computed accordingly. Since this prototype provides two separate scores, one is based on the grammar and other on text content, therefore, an instructor has to assign weights explicitly for each score, while calculating the final score for student answer [2][3].

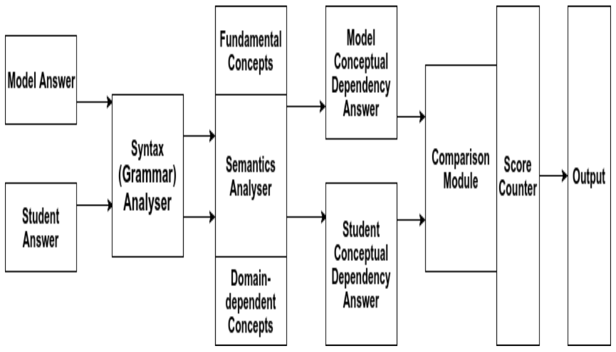

Model Answer
Student Answer
Syntax (Grammar) Analyser
Fundamental Concepts
Semantics Analyser
Domain-dependent Concepts
Model Conceptual Dependency Answer
Student Conceptual Dependency Answer
Comparison Module
Score Counter
Output

Figure 3.2: [3] ATM basic architecture

C-rater

It is short answer grading tool, introduced by Educational Testing Service(ETS). To grade student's response, it determines the presence and absence of each concept by comparing it with the reference answer provided by the teacher. This is done by extracting features from answers such as syntactic variation; the difference between active and passive voice, pronoun reference; if the noun is replaced by the pronoun, morphological variation; change in internal structure of word due to syntax of sentence e.g. (laptop, laptops, laptop's), and use of synonyms. Furthermore, this tool also determines the misspelled words by checking its existence in the dictionary, and misspelled words are corrected by using minimum edit distance algorithm [35][2].

However, according to Mohler and Mihalcea [37], proposed approach ignores bag-of-words technique, due to which difference between "dog bites man" and "man bites dog" is not considered.

3.2 Era of Information Extraction

In this era, systems work by extracting facts from the student's response to patterns such as regular expressions or parse tree and compare with the reference answer provided by the teacher. Since short answers are mostly expected to include specific ideas, these ideas are modeled by templates in instructor's answer and later on compared with student answer for final score [38][2]. Following section briefly describe some popular systems related to this era:

AutoMark

This system employs information extraction technique to assign scores to short free-text responses. The proposed approach introduces specific template to identity acceptable answers for each question known as a syntactic-semantic template. Model answer is marked with possible expected answers, if student's response fulfills predefined marked template, then a response is considered as correct. Figure 3.3 shows the template of example "The Earth rotates around the Sun". Now, patterns of student's response are expected to match one of the mentioned verbs e.g: "rotate", "revolve", "orbit", "travel", "move", with one of nouns "Earth" or "world", and with one of the prepositions.

However, according to Mohler and Mihalcea [37], this approach requires manually crafted patterns as templates or large training data beforehand to grade the student answer.

Indus Marker

This is another approach for short answer grading. It shares some common properties with existing systems such as AutoMark, but also has unique design features. For a given reference and student answer, first spell checking is performed by JOrtho; an open-source spell checker, after that using Stanford CoreNLP Dependency parser [39], the relation between words and part of speech tags are determined. Based on the similarity of relations between words and part of speech tags in reference and student answer, the final grade is computed. For evaluation purpose, authors use the dataset with approximately 279 student answers for each question. In total, there are 5 questions. In starting, model answers and around 50 student answers were used as target/correct syntactical structures and grammatical relations. While, on remaining answers, the algorithm is tested. As

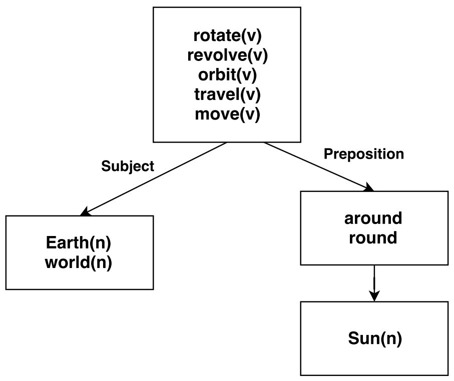


Figure 3.3: [4] Simple mark scheme template of above example “ The Earth rotates around the Sun ”

a result, authors conclude that human-system agreement rate is around 96%, which outperforms existing systems [2][40].

3.3 Era of Corpus-based Method

These methods measure the degree of similarity between words using information acquired from large corpora such as Wikipedia, Google, AltaVista, etcetera. Corpus-based methods are useful, when student answer contains paraphrase of words present in teacher answer, as using vocabulary only from reference answer may limit the correct answers to be determined. Following section briefly describe some popular systems related to this era:

Mohler’09

In this approach, Mohler and Mihalcea [37] explore and evaluate unsupervised techniques for the short-answer grading system. The paper establishes three valuable contributions, firstly, authors have shown the comparison of performance between corpus-based and knowledge-based measures. Secondly, authors examined the effect of the domain and corpus size on the performance of corpus-based measures. As a result, authors concluded that notable improvement can be achieved with the use of medium size domain-specific corpus built from Wikipedia. Thirdly, using pseudo-relevance feedback technique, the system is able to improve the quality by considering the students’ correct response with the highest score. Authors created a dataset with 21 questions and 630 student answers, to evaluate existing approaches and proposed approach based on Pearson correlation coefficient, where average score of human graders is compared with the score provided by the system. Overall, proposed system outperforms existing unsupervised methods for short answer grading [2].

Klein'11

In this paper, Klein et al. [10] implement automatic short answer grading system using Latent Semantic Analysis(LSA). LSA is a technique to create the vector representation of text. Using vectors, the distance between two documents is computed with the help of cosine similarity [41]. Initially, all the student answers are pre-processed to standardize the texts by removing punctuations and stop words. After that, term frequency matrix of all the terms present in each answer is constructed, where content words are assigned higher weights using Term Frequency-Inverse Document Frequency(TF-IDF) weighting scheme. In order to reduce the dimension of TF-IDF vectors, singular value decomposition(SVD) is applied. To test the algorithm, a small set of answers(M) is selected randomly and graded manually. All answers outside of set M are compared with answers graded using pre-defined similarity function.

However, the drawback of the current approach is that, to achieve desired effectiveness level of a system, the majority of the student answers should be manually graded. Furthermore, the quantity of answers to be graded manually needs to be determined correctly.

3.4 Era of Machine Learning

In this era, systems use a number of features obtained from natural language processing techniques such as named entity recognition, part of speech taggers, parse trees, morphological analysis; lemmatization and stemming, that contribute for final grade using regression and classification [2]. For this research and development project, we also use machine learning method; more specifically, approach by Sultan et al. [7]. Following section briefly describe some popular systems related to this era:

Mohler'11

Mohler et al. [42] evaluate the existing bag-of-word approaches to assess the short text responses. In order to improve the performance of existing approaches, Mohler et al. [42] use machine learning techniques such as support vector machine to combine lexical semantic similarity with graph alignment features. To comprehend the syntactic difference between sentences, dependency graphs of student's response and instructor's model answer are generated from the Stanford CoreNLP Dependency parser [39]. Common features such as root matching, exact match, stemmed match, the common relation between children and parents, etcetera, are extracted and fed to train machine learning algorithm to perform alignment of both answers(student and instructor answers). In order to compute optimal alignment between graphs of both answers, Hungarian algorithm is applied.

However, finding an optimal alignment is not enough to determine aligned answers are entailing as discussed in [43]. For suppose, if two sentences are exactly identical, and we replace adjective from one sentence to its antonym, are still considered as optimal alignment. Therefore, to make the system more robust, Mohler et al. [42] also employed word embeddings, where vectors are created for each answer, by adding vector for each word present in an answer and ignoring stop words. After that, cosine similarity is computed between vectors of both answers. To produce final grade, alignment scores from graph alignment features; scores from Hungarian algorithm, and lexical similarity; scores obtained from cosine similarity, are considered for training of SVM model for regression.

To evaluate the approach, Mohler et al. [42] created the dataset named as Texas, consists of 2273 student answers from the computer science assignments; more details given in section 3.5. As a result, it is concluded that machine learning techniques improve the performance of existing bag-of-word approaches. While, rudimentary alignment features are not enough to be considered as the standalone grading system. For the future work, quality of answer alignments can be enhanced by training a model using a graph to graph alignments.

This approach became popular during that time. However, according to Sultan et al. [7] it uses the large set of similarity measures for a supervised learning; which means a large number of features leads to higher runtime and difficulty during implementation.

Sultan'16

This is another system proposed by Sultan et al. [7] for automatic short answer grading, which is fast, easy to implement and highly accurate as compared to previous approaches. For a given sentence, the system extracts text similarity features such as word alignment, embeddings, question demoting, term weighting and length ratio. Using these features, the supervised model is trained in order to compute the grade for student answer based on similarity with reference answer. Furthermore, Sultan et al. [7] evaluate approach on dataset proposed by Mohler et al. [42] called Texas dataset, and on SemEval-2013 dataset [44]. From machine learning point of view, evaluation on Texas dataset is regression task; more specifically ridge regression is used to compute grade based on similarity features, and evaluation on SemEval-2013 is considered as classification task; particularly task is to find whether an answer is relevant or not. Lastly, results show that current approach outperforms previous approaches on both datasets.

As it performs better than previously existing approaches, therefore, we use this approach for our research and development project, later on, we explain this approach in more details.

3.5 Era of Evaluation

During this era, existing approaches make use of shared corpora, in order to compare the performance of proposed approach with the existing approaches. It also includes competitions and evaluation forums, where for a common task, existing approaches compete against each other for the sake of reputation or prize money [2]. Following section briefly describe some competitions and evaluation forums.

Texas Dataset

Existing approaches also use a dataset from Mohler et al. [42] to do the performance comparison. It is created from Data Structures course at the University of North Texas. It consists of ten assignments and two examinations, overall around 80 questions. The total number of answers including all the student responses are around 2273. Furthermore, answers were independently graded by two human graders on integer scale between 0 and 5, where 0 represent completely incorrect and 5 represents the perfect answer. Moreover, an average grade of two evaluators is taken and used as the gold standard to examine the automatic grading task. So, in other words, for each student answer; we have 3 scores; two from human graders and one average between them [42]. For our project, we also use this dataset for evaluation purposes.

Automated Student Assessment Prize(ASAP)'12

ASAP is competition for automatic short answer grading organized by Kaggle [45]; the largest community of data scientists and competition hosting company. For this task, data was comprised of 10 questions from arts course. The participants were provided 1,800 student answers for training, each answer contains score to be predicted and confidence score. For testing purpose, approximately 6,000 student answers were used. For evaluation purpose, Quadratic weighted kappa was considered and evaluated based on an agreement between predicted scores and confidence scores between two human graders [45].

SemEval'13 Task 7

Semantic Evaluation is an ongoing series of evaluations to evaluate semantic analysis systems [46], and it was first large-scale and non-commercial competition for automatic short answer grading. Furthermore, task introduced two datasets; BEETLE and SCIENTSBANK. The SCIENTSBANK training corpus consists of approximately 10,000 responses to 197 questions from 15 different science domains. All student responses were graded by multiple human graders on a scale of "Correct", "Partially correct incomplete", "Contradictory", "Irrelevant" and "non-domain" [47]. On the other hand, BEETLE training corpus contains 56 questions from the basic electricity and electronics domain, and approximately 3000 student answers to 56 questions. Each answer is approximately of 1 or 2 sentences [44].

Chapter 4

Monolingual Alignment

Monolingual alignment is one of the important component for short answer grading. It is a way of aligning words that are semantically similar present in a pair of sentences; student and teacher answer. This alignment is necessary as it determines how and to what extent student and teacher answers are related to each other. Furthermore, the task of aligning words is important for numerous applications such as textual inference, text comparison, short answer grading, text summarization, paraphrase detection and so on [5].

According to state of the art, monolingual alignment was proposed for various tasks such as textual entailment recognition [48][49], paraphrase identification [50], textual similarity assessment [51], however, they were not evaluated on alignment benchmarks as there was not any annotated corpus available for evaluation.

During 2007, Brockett [52] introduced manually aligned corpus with training and test sets from Microsoft Research, due to which evaluation and comparison between different monolingual alignment approaches became possible. With the passage of time, different monolingual alignment approaches such as MANLI [53], Thadani & McKeown [54], Yao et al. [55], Xuchen Yao and Clark [56], and Sultan et al. [5] were introduced and evaluated on benchmark by Brockett [52]. Among all existing approaches, Sultan et al. [5] outperform on publicly available benchmarks. Therefore, for our project, we use monolingual alignment by Sultan et al. [5]. Later on, we also explain his approach and implement using Spacy and NLTK libraries.

Furthermore, word semantic similarity is an important component for most of the word aligner systems, and different ways for computing word similarity are being used such as string similarity, resource-based similarity(which is derived from WordNet), and distributional similarity; determined using word co-occurrence statistics in large corpora. Due to balance between accuracy, speed, and precision, Sultan et al. [5] uses Paraphrase Database(PPDB) [57] that is constructed on large bilingual parallel corpora consisting of lexical and phrasal paraphrases [58].

4.1 System Description

Figure 4.1 shows pipeline of word aligner that consists of four modules, each module align a unique pair of words present in student and teacher answer. Each module considers contextual evidence to align the words. By contextual evidence we mean, determining word alignment based on surrounding words or phrases. However, last two modules also include word semantic similarity. Later on, we explain each module in more details. First, we will have look at components that determine word similarity and how we can extract contextual evidence(or contextual similarity) from sentences to align the words.

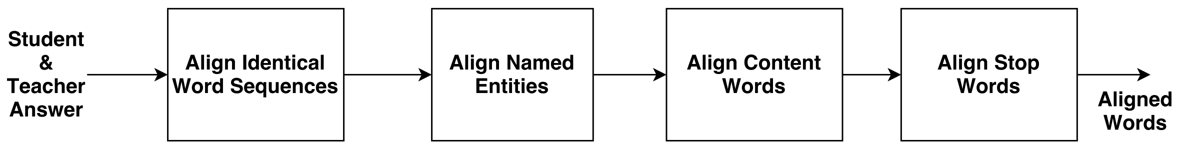


Figure 4.1: [5] Pipeline of monolingual word aligner

Word Similarity

This similarity determines whether words are identically or semantically similar. To determine, we check similarity for three cases:

- If two words or their lemmas are exactly identical, we consider similarity score as 1.
- If two words or their lemmas are not identical, then we use Paraphrase Database(PPDB) [57] to determine whether they are semantically similar or not. We use the term *ppdbSim* to represent score of similarity(between value of (0,1)).
- If pair of words are not present in Paraphrase Database(PPDB), we consider similarity score as 0 [5][58].

Contextual Similarity

Contextual similarity is determined from two complementary sources:

- Syntactic dependencies.
- Textual neighborhood of two words to be aligned.

Assume we have two sentences(by two, we mean student and teacher answer) S and T, consider a pair of words; s ∈ S and t ∈ T, to find alignment pair if $\exists r_s \in$ S and $\exists r_t \in$ T, such that:

- (s, t) $\in R_{sim}$ and $(r_s, r_t) \in R_{sim}$, where r_s and r_t are children of words to be aligned and R_{sim} is a binary relation that indicates semantic relatedness between each pair of words; in our case (s, t) and (r_s, r_t), which is determined by using Paraphrase Database(PPDB). This similarity score should be greater than or equal to *ppdbSim*.

- (s, r_s) $\in R_{C_1}$ and (t, r_t) $\in R_{C_2}$ such that $R_{C_1} \approx R_{C_2}$, where R_{C_1} and R_{C_2} represent specific kind of contextual relations such as nsubj(nominal subject), dobj(direct object), iobj(indirect object), etcetera, between word to be aligned(parent) and their children in each sentence(for example, nsubj dependency relation between a verb and noun). Lastly, symbol $\approx$ denotes equivalence between two relationships, it can be identical also [5].

In the following section, we will explain more about two sources; syntactic dependencies and textual neighborhood, and how they provide evidence for alignment of words.

Syntactic Dependencies

Syntactic dependencies are one of the important sources for extracting contextual evidence. They represent the grammatical structure of a sentence in terms of binary relations(relation between root/governor /head and a dependent/child) between tokens. In short, it tells us children and parent relations for each word.

Given two sentences S and T, first we compute part of speech tags for each word present in S and T, and then we determine dependency parse tree of sentences that gives parent and children relation of each word. For instance, if we want to align verbs in sentences, then two nsubj(nominal subject) children (r_s, r_t) of two verbs (s, t) not only indicate verbs are similar, but also two children are similar if (s,t) $\in R_{sim}$ and $(r_s, r_t) \in R_{sim}$.

Furthermore, dependency types(such as nsubj, dobj, more listed in 2.2) can compute equivalence between two words. Figure 4.2 shows two sentences with their dependency relations and POS tags. To align the verbs in both sentences, we determine verbs in S and T. Since S and T have two verbs; "wrote" and "wrote" respectively, that are similar. We then check for their dependency relations. For this example, we will consider the relationship between content words. If we look at the relation between "book" and "wrote", dobj dependency in sentence S is equivalent to the rcmod dependency in sentence T, since they represent the similar semantic relation between identical word pairs in S and T. Although parent-child orientations are opposite in both sentences. On the other hand, "He" in both sentences are also aligned since they share same relation with their parent word "wrote".

In the following section, we will describe the different type of parent-child orientations that provide evidence for aligning words based on dependency types and also how "wrote" and "book" are aligned based on their dependency relations. From notations point of view, s and t represents "wrote", r_s and r_t denotes "book" in both sentences(S and T).

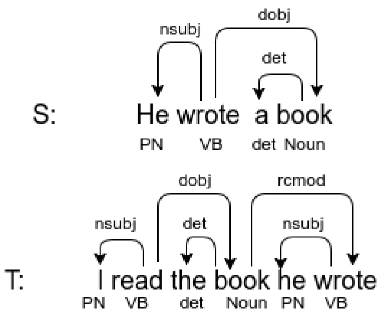


Figure 4.2: [5] Example of equivalent dependency types

Figure 4.3 shows possible combination of parent-child orientations. If (s,t) $\in R_{sim}$ and $(r_s, r_t) \in R_{sim}$ shown by bidirectional arrows, then each parent-child orientation(a,b,c,d in figure) denotes set of possible ways that gives indication of similarity in contexts of token s in sentence S and token t in sentence T. In order to determine, to which orientation our example corresponds to, we see that s(wrote) points to its children r_s(book), and r_t(book) points to t(wrote) in sentence T. If we look at parent-child orientation in figure 4.3, we can conclude that our example resembles to orientation (c).

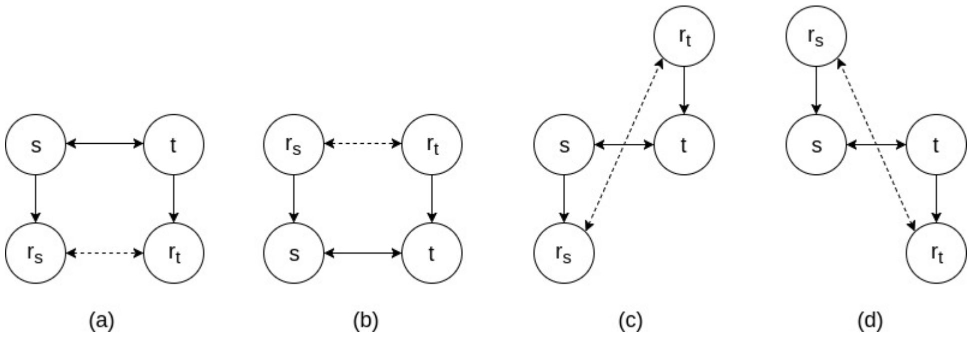
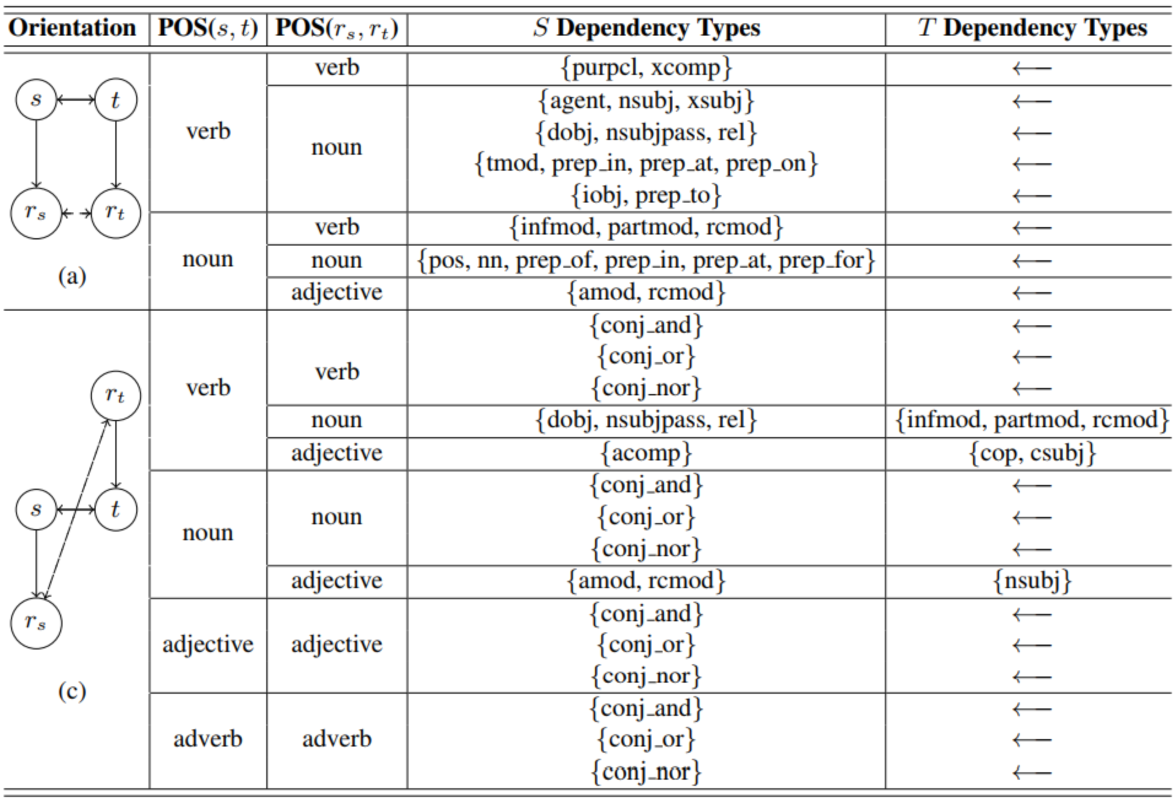


Figure 4.3: [5] Parent child orientations in dependencies

Furthermore, we apply dependency type equivalence only to content words that lie in four main lexical categories; nouns, verbs, adjectives, and adverbs. We use NLTK and Spacy part of speech tagger to identify the category of each word. Figure 4.4 shows equivalent dependency structure. '$\leftarrow$' in a column of "T Dependency Types" represents the duplication of same rows in a column of "S Dependency Types". Second and third columns represent part of speech of tokens s and t and their children r_s and r_t.

Since our example corresponds to parent-child orientation(c), so we consider second row in figure 4.4. Furthermore, since s and t are verbs, r_s and r_t are nouns. We can see that possible relations between verb and noun in figure 4.4 are {dobj, nsubjpass, rel} in S dependency types and {infmod, partmod, rcmod} in T dependency types. If relation in sentence S between verb and noun(which is dobj in our example), lies in S dependency types and relation in sentence T between verb and noun(which is rcmod in our example) lies in T dependency types. Then, we consider both the children in sentence S and T are aligned.

Figure 4.4 contains only orientations (a) and (c), because other orientations; (b) and (d) will be almost similar, to avoid redundancy sultan et.al(2014) does not present these orientations. Also, note that, figure contains word pairs(s_i, t_j) so that POS(s_i) = POS$(t_j$, This is because 90% of all content words in manually aligned corpus by Brockett[52] are within same lexical category. Therefore, Sultan et al.(2014) consider this as reasonable point to start.



Orientation	POS(s,t)	POS(r_s,r_t)	S Dependency Types	T Dependency Types
(a)	verb	verb	{purpcl, xcomp}	$\longleftarrow$
		noun	{agent, nsubj, xsubj}	$\longleftarrow$
			{dobj, nsubjpass, rel}	$\longleftarrow$
			{tmod, prep_in, prep_at, prep_on}	$\longleftarrow$
			{iobj, prep_to}	$\longleftarrow$
	noun	verb	{infmod, partmod, rcmod}	$\longleftarrow$
		noun	{pos, nn, prep_of, prep_in, prep_at, prep_for}	$\longleftarrow$
		adjective	{amod, rcmod}	$\longleftarrow$
(c)	verb	verb	{conj_and}	$\longleftarrow$
			{conj_or}	$\longleftarrow$
			{conj_nor}	$\longleftarrow$
		noun	{dobj, nsubjpass, rel}	{infmod, partmod, rcmod}
		adjective	{acomp}	{cop, csubj}
	noun	noun	{conj_and}	$\longleftarrow$
			{conj_or}	$\longleftarrow$
			{conj_nor}	$\longleftarrow$
		adjective	{amod, rcmod}	{nsubj}
	adjective	adjective	{conj_and}	$\longleftarrow$
			{conj_or}	$\longleftarrow$
			{conj_nor}	$\longleftarrow$
	adverb	adverb	{conj_and}	$\longleftarrow$
			{conj_or}	$\longleftarrow$
			{conj_nor}	$\longleftarrow$

Figure 4.4: [5] Equivalent dependency structure of orientations (a) and (c)

Algorithm for extracting contextual evidence based on dependencies is shown in algorithm 1, it takes input word pair(s_i, t_j) from sentences S and T, returns contextual evidence as indexes of children of tokens s_i and t_j along with their positive similarity score for each matching row in figure 4.4.

```
input  :
        1. S, T: Sentences to be aligned
        2. i: Index of a word in S
        3. j: Index of a word in T
        4. EQ: Dependency type equivalences(in Figure 4.4 )
output: context = {(k,l)} : pairs of word indexes
1 if wordSim($s_k$, $t_l$) > 0 and (i, k, $\tau_s$) $\in$ dependencies(S) and (j, l, $\tau_t$) $\in$
  dependencies(T) and POS($s_i$) = POS($t_j$) and POS($s_k$) = POS($t_l$) and ($\tau_s = \tau_t$) or
  ($POS(s_i), POS(s_k), \tau_s, \tau_t \in EQ$) then
2     context ← {(k,l) } // $s_k$ and $t_l$ are children of token s and t.  $\tau_s$ and $\tau_t$
      are dependency relation between word in s/t and their children, k and
      l are indexes of children of s and t
3 end
```

Algorithm 1: [5] depContext(S, T, i, j, EQ)

Textual Neighborhood

We consider evidence based on textual neighborhood due to following reasons:

- The accuracy of contextual evidence using syntactic dependencies is limited due to the accuracy of parsers such as Stanford CoreNLP, NLTK, and Spacy dependency parsers.
- Here, we determine contextual evidence based on consecutive or neighborhood words.

For each word pair (s_i, t_j), we consider content words(excluding stop words and punctuations) within a window of (-3, +3) neighborhood words lies at the left and right side of word pairs. Algorithm 2 shows how contextual evidence using textual neighborhood is considered. It takes input indexes i and j, sentences S and T and stop words, returns neighborhood words of i and j within the window of (-3, +3).

```
input  :
        1. S, T: Sentences to be aligned
        2. i: Index of a word in S
        3. j: Index of a word in T
        4. STOP: A set of stop words
output: context = {(k,l)} : pairs of word indexes
1 $C_i$ ← {k : k $\in$ [i - 3, i + 3] and k $\neq$ i and $s_k \notin STOP$} // k denote neighborhood
  word indexes in S and $s_k$ denote neighborhood words
2 $C_j$ ← {l : l $\in$ [j - 3, j + 3] and l $\neq$ j and $t_l \notin STOP$} // l denote neighborhood
  word indexes in T and $t_l$ denote neighborhood words
3 context ← $C_i \times C_j$
```

Algorithm 2: [5] textContext(S, T, i, j, STOP)

4.1.1 Alignment Pipeline

So far, we looked into two sources for extracting contextual evidence which is used by each module present in the pipeline. Now we explain working of each alignment module, and the reason why Sultan et al. [5] has placed modules in this order.

Identical Word Sequences

It considers contextual evidence to align the identical words in sentence S and T. As previously we discussed, there are two sources for extracting contextual evidence; syntactic dependencies(consider parent-child dependencies for alignment) and textual neighborhood. This module aligns words based on textual neighborhood only, instead of (3,3) window of words, we align word sequence that contains at least one content word(noun, verb, adjective, adverb) present in both sentences. This simple approach shows a high precision(around 97%) on manually aligned corpus by Brockett [52]. From now on, we name output of this module as *wsAlign.* Let's consider simple example where:
S = " I eat five pizzas " and T = " I eat fruits and pizzas "
Alignment by this module(*wsAlign*) = [['pizzas', 'pizzas'], ['I', 'I'], ['eat', 'eat']]

Since 'I' and 'eat' are consecutive words containing at least one content word; 'eat', therefore, module aligns them, while 'pizzas' is aligned since it is also a content word.

Named Entities(NE)

It is the second module of our word aligner pipeline. For a given sentences, we determine named entities categories/tags for each word using NLTK and Spacy libraries. After that, "Learn Named Entities" sub-module, assigns possible NE tags to words that do not have any tag, by comparing it with another sentence. For suppose we have sentence S = "UAE is one of the biggest country in the world" and T = "Many people are living in the United Arab Emirates". Now "UAE" in S is not assigned any tag since it is just acronym, but, "United Arab Emirates" in T is already assigned NE tag/category as it is a noun. Therefore, first, we determine whether "UAE" is an acronym of "United Arab Emirates" by just comparing each letter of "UAE" with each word of "United Arab Emirates", and then we assign same named entity as "United Arab Emirates" have, to "UAE".

Second sub-module i.e. "Align Full Matches" aligns named entities that are exactly similar in both sentences such as "United Nation Organizations" or any other Person/Organization/Location names.

Furthermore, "Align Acronyms" aligns words with their acronyms such as "UAE" and "United Arab Emirates" and "Align Subset Matches" align subsets such as S contain "United States" and T contains "United", as a result, we get "[[United, United]]".

The main purpose to place NE module before content words module is to make sure that alignment of acronyms is done before content words.

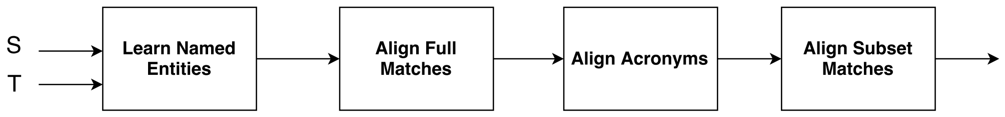


Figure 4.5: Named Entities Module

Limitations of Named Entities Module

- Since named entities are instances of nouns. Therefore, named entity tag is assigned to a word, if its first letter is capital such as "John", "United States", and "Bonn", rather than, "john", "united states", and "bonn". This problem is solved during alignment of nouns.
- Alignment of an acronym is not optimal, the algorithm only align acronyms with their full-forms, by comparing each letter of the acronym with the first letter of each word in full-form, as we have seen in case of "United Arab Emirates". This does not perform well, when we are aligning "USA" with "United States of America".

Content Words

This module uses contextual evidence and word similarity to align the words. Contextual evidence is extracted using syntactic dependencies(algorithm 1) and textual neighborhood(algortihm 2).

Algorithm 3 explains the alignment process for content words based on syntactic dependencies. For a given sentence S and T, dependency based context for each word pair(s_i, t_j); excluding stop words and not already aligned words, is extracted using algorithm 1. Furthermore, contextual similarity(*contextSim*) is determined as a sum of word similarities of the content word pairs(s_k, t_l) (as shown in algorithm; lines 2-5), where word similarity (*wordSim*) returns a score within $\{0, ppdbSim, 1\}$. In addition to this, alignment score($\phi(i,j)$) is computed as the weighted sum of contextual and word similarity(lines 6-10). Later on, we explain criteria of setting weights and *ppdbSim*. After that, module aligns word pairs(s_i, t_j) having non-zero evidence(Ψ) in descending order of alignment score($\phi(i,j)$)(lines 15-19 in algorithm). Lastly, it also aligns all the pairs(such as children/parents of word pair (s_i, t_j)) that helps for contextual evidence of alignment for (s_i, t_j)(lines 20-24).

```
input  :
          1. S, T: Sentences to be aligned
          2. EQ: Dependency type equivalences( as shown in figure 4.4)
          3. $A_E$: Already aligned word pair indexes
          4. STOP: A set of stop words
          5. w: Weight of word similarity relative to contextual similarity
output: A = {(i, j)} : word index pairs of aligned words $\{(s_i, t_j)\}$ where $s_i \in$ S and $t_j$ $\in$ T
1  $\Psi \leftarrow \emptyset$; $\Delta_\Psi \leftarrow \emptyset$ ; $\Phi \leftarrow \emptyset$
2  for $s_i \in S$, $t_j \in T$ do
3      if $s_i \notin STOP$ and $\neg\exists t_l$ : (i, l) $\in A_E$ and $t_j \notin STOP$ and $\neg\exists s_k$ : (k, j) $\in A_E$ and
       wordSim($s_i$, $t_j$) > 0 then
4          context $\leftarrow$ depContext(S, T, i, j, EQ) // l and k are children indexes,
           context is computed using algorithm 1, returns pair of word
           indexes(k,l)
5          contextSim $\leftarrow \sum_{(k,l)\in context} wordSim(s_k, t_l)$ // for each index pairs in
           context, compute contextSim similarity
6          if contextSim > 0 then
7              $\Psi \leftarrow \Psi \cup$ {(i, j)} // collect evidence of similarity
8              $\Delta_\Psi(i, j) \leftarrow$ context // store word indexes(k,l)
9              $\Phi_{i,j} \leftarrow$ w * wordSim($s_i, t_j$) + (1 - w) * contextSim
10         end
11     end
12 end
13 Sort $\Psi$ in descending order of $\Phi(i, j)$
14 A $\leftarrow \emptyset$
15 for (i, j) $\in \Psi$ do
16     if $\neg\exists l$ : (i, l) $\in$ A and
17     $\neg\exists k$ : (k, j) $\in$ A then
18         A $\leftarrow$ A $\cup$ {(i,j)}
19     end
20     for (k,l) $\in \Delta_\Psi(i, j)$ do
21         if $\neg\exists q$ : (k,q) $\in A \cup A_E$ and $\neg\exists p$ : (p, l) $\in A \cup A_E$ then
22             A $\leftarrow$ A $\cup$ {(k, l)}
23         end
24     end
25 end
```

Algorithm 3: [5]cwDepAlign(S, T, EQ, A_E, STOP, w)

Algorithm 4 describes the alignment process for content words based on textual neighborhood. For a given sentence S and T, textual neighborhood for each word pair is extracted using algorithm 2, that consists of neighboring content word pairs(s_k, t_l)(lines 2-4). Furthermore, contextual similarity(*contextSum*) is computed as sum of word similarities of the content word pairs(s_k, t_l)(line 5), and alignment score($\phi(i, j)$) is determined as the weighted sum of contextual and word similarity(line 9). Lastly, we do not consider textual neighbors of similar words for alignment as they are weaker sources of evidence. We use alignment score($\phi(i, j)$) to align one-to-one word alignment(lines 10-17).

```
input :
        1. S, T: Sentences to be aligned
        2. $A_E$: Already aligned word pair indexes
        3. w: Weight of word similarity relative to contextual similarity
        4. STOP: A set of stop words
output: A = {(i, j)} : word index pairs of aligned words {($s_i$, $t_j$)} where $s_i$ ∈ S and $t_j$ ∈ T
1  $\Psi \leftarrow \emptyset$; $\Phi \leftarrow \emptyset$
2  for $s_i \in S$, $t_j \in T$ do
3      if $s_i \notin STOP$ and $\neg\exists t_l$ : (i, l) $\in A_E$ and $t_j \notin STOP$ and $\neg\exists s_k$ : (k, j) $\in A_E$ and
       wordSim($s_i$, $t_j$) > 0 then
4          $\Psi \leftarrow \Psi \cup$ {(i, j)}
5          context ← textContext(S, T, i, j, STOP) // l and k are children
           indexes, context is computed using algorithm 2, returns pair of
           word indexes(k,l)
6          contextSim ← $\sum_{(k,l) \in context} wordSim(s_k, t_l)$ // for each index pairs in
           context, compute contextSim similarity
7          $\Phi_{i,j}$ ← w * wordSim($s_i, t_j$) + (1 - w) * contextSim
8      end
9  end
10 Sort $\Psi$ in descending order of $\Phi(i, j)$
11 A ← $\emptyset$
12 for (i, j) $\in \Psi$ do
13     if $\neg\exists l$ : (i, l) ∈ A and
14     $\neg\exists k$ : (k, j) ∈ A then
15         A ← A ∪ {(i,j)}
16     end
17 end
```

Algorithm 4: [5]cwTextAlign(S, T, A_E, w, STOP)

Stop Words

This module also considers contextual evidence and word similarity to align the words. Some of the stop words are aligned in previous modules; identical word sequences(as explained in 4.1.1) and in syntactic dependencies (as explained in section 4.1). For the not aligned stop words, we use syntactic dependencies and textual neighborhoods as we did in previous modules, but with three adjustments.

Firstly, we consider only aligned pairs for contextual similarity(*contextSim*) rather than all semantically similar word pairs. Since stop words is the last module of our pipeline, therefore, we use aligned pairs from previous modules. Secondly, as stop words; such as determiners and models, shows little difference in dependencies, therefore, we do not consider type equivalences(like if two stop words lies in the same group(refer 4.4), we do not consider them as aligned, unlike previous modules), rather than only exact matching of dependencies. Lastly, to account contextual evidence using textual neighborhoods, alignment of stop words is determined by considering the alignment of left and right neighbors. This is done to make sure all stop words are aligned because dependency parsers are not always optimal.

Since stop words utilize syntactic dependencies and textual neighborhood-based alignments, therefore we consider two modules; swDepAlign and swTextAlign in algorithm 5.

```
  input :
          1. S, T: Sentences to be aligned
          2. EQ: Dependency type equivalences(4.4)
          3. w: Weight of word similarity relative to contextual similarity
          4. STOP: A set of stop words
  output: A = {(i, j)} : word index pairs of aligned words {($s_i$, $t_j$)} where $s_i$ ∈ S and $t_j$ ∈ T
1 A ← wsAlign(S, T)
2 A ← A ∪ neAlign(S, T, EQ, A, w)
3 A ← A ∪ cwDepAlign(S, T, EQ, A, w, STOP)
4 A ← A ∪ cwTextAlign(S, T, A, w, STOP)
5 A ← A ∪ swDepAlign(S, T, A, w, STOP)
6 A ← A ∪ swTextAlign(S, T, A, w, STOP)
```

Algorithm 5: [5] align(S, T, EQ, w, STOP)

Complete Pipeline Overview

Algorithm 5 shows complete pipeline of word aligner. Each module does not align the words which are aligned by previous modules. Sultan et al. [5] arrange the modules in this order due to following reasons:

- *wsAlign* is a module that has highest precision.
- *neAlign* is placed before content words because it is convenient to align the partial and full mentions; such as acronyms before content words.
- In content words, syntactic dependencies are considered before textual neighborhood due to higher precision, evaluated on manually aligned corpus by Brockett [52].

- Stop words are placed in last as they are dependent on existing alignments of content words.

Furthermore, word aligner also contains two free parameters; *ppdbSim* and w(as shown in algorithm 3 and 4). Sultan et al.(2014) consider values of both parameters as (0.9, 0.9), because these values produce best F1 score on corpus by Brockett [52].

4.1.2 Word Aligner Implementation

Figure 4.6 shows detailed implementation of word aligner. During pre-processing, for a given student and teacher answer, first, the dependency tree, and parameters; such as part of speech tags, lemmas and named entity recognition are determined. An output of pre-processing is fed to the pipeline of word aligner. First, we align the punctuations, and then common neighboring words that contain at least one content words are aligned. After that, hyphen words, named entities, and content words based on syntactic dependencies and textual neighborhood are aligned. However, for the application of short answer grading, Sultan et al. [7] does not use the alignment of stop words, may be due to performance. Therefore, this module is not presented in figure 4.6

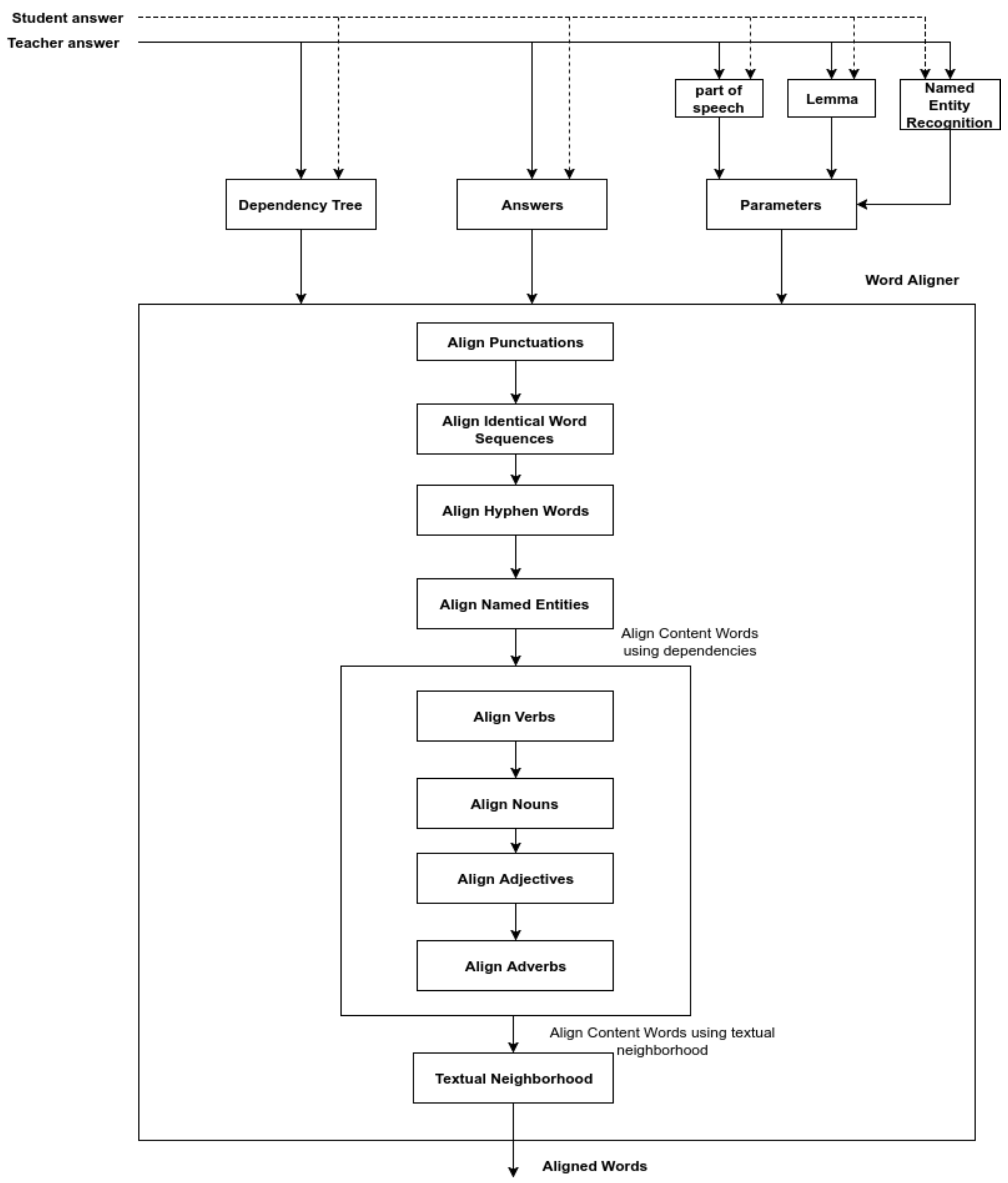
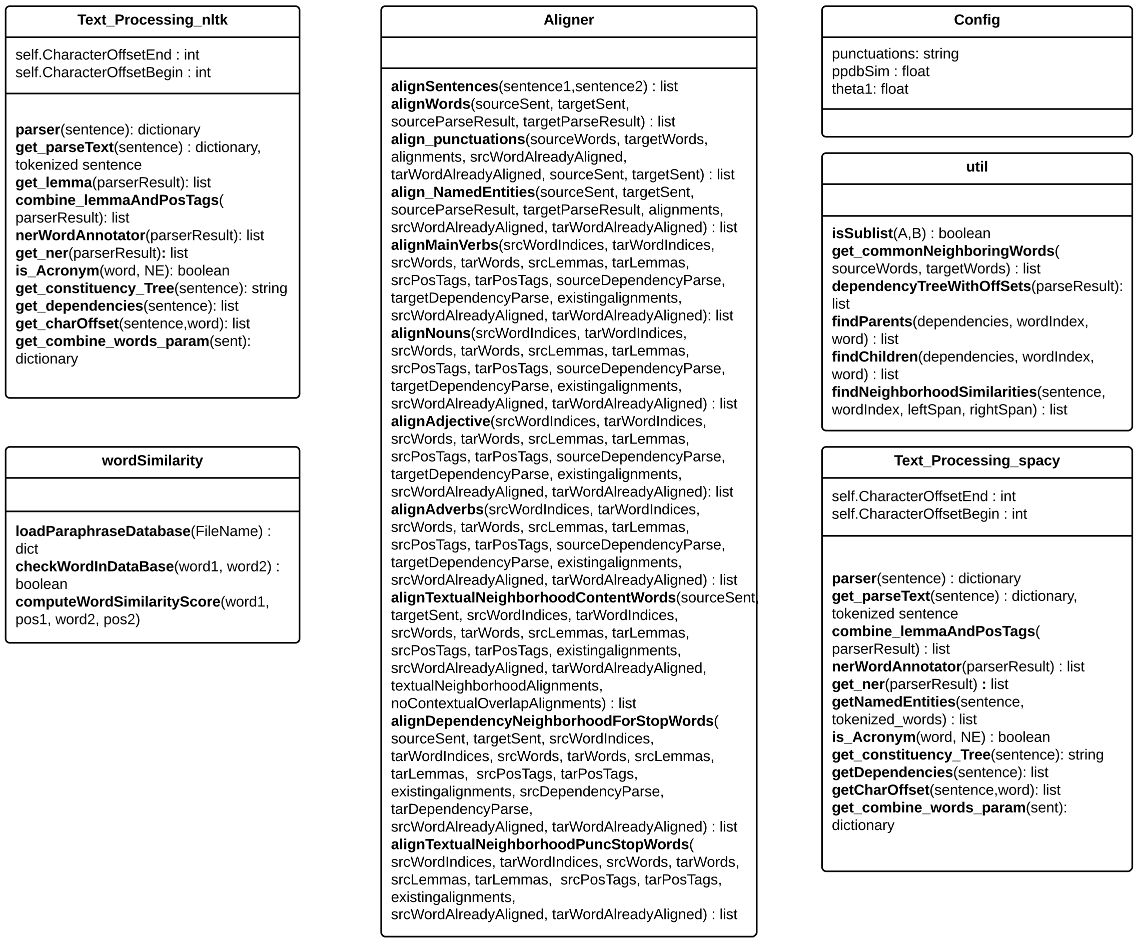


Figure 4.6: Detailed diagram of word aligner

Class Diagram

Figure 4.7 shows class diagram of word aligner implementation. We divide word aligner into six classes. Aligner class includesall the main functions that are available in the block of word aligner in figure 4.6, where, Config, wordSimilarity, util contains auxiliary functions for Aligner class. Furthermore, since we implement word aligner using NLTK and Spacy, therefore, we have separate pre-processing classes; Text_Processing_nltk and Text_Processing_spacy for each library. Pre-processing classes contain mainly functions that are shown outside of word Aligner block in figure 4.6.

Note: we do not display all functions in Aligner class to reduce complexity while displaying. It contains only main functions.



Text_Processing_nltk

self.CharacterOffsetEnd : int
self.CharacterOffsetBegin : int

parser(sentence): dictionary
get_parseText(sentence) : dictionary, tokenized sentence
get_lemma(parserResult): list
combine_lemmaAndPosTags(parserResult): list
nerWordAnnotator(parserResult): list
get_ner(parserResult)**:** list
is_Acronym(word, NE): boolean
get_constituency_Tree(sentence): string
get_dependencies(sentence): list
get_charOffset(sentence,word): list
get_combine_words_param(sent): dictionary

wordSimilarity

loadParaphraseDatabase(FileName) : dict
checkWordInDataBase(word1, word2) : boolean
computeWordSimilarityScore(word1, pos1, word2, pos2)

Aligner

alignSentences(sentence1,sentence2) : list
alignWords(sourceSent, targetSent, sourceParseResult, targetParseResult) : list
align_punctuations(sourceWords, targetWords, alignments, srcWordAlreadyAligned, tarWordAlreadyAligned, sourceSent, targetSent) : list
align_NamedEntities(sourceSent, targetSent, sourceParseResult, targetParseResult, alignments, srcWordAlreadyAligned, tarWordAlreadyAligned) : list
alignMainVerbs(srcWordIndices, tarWordIndices, srcWords, tarWords, srcLemmas, tarLemmas, srcPosTags, tarPosTags, sourceDependencyParse, targetDependencyParse, existingalignments, srcWordAlreadyAligned, tarWordAlreadyAligned): list
alignNouns(srcWordIndices, tarWordIndices, srcWords, tarWords, srcLemmas, tarLemmas, srcPosTags, tarPosTags, sourceDependencyParse, targetDependencyParse, existingalignments, srcWordAlreadyAligned, tarWordAlreadyAligned) : list
alignAdjective(srcWordIndices, tarWordIndices, srcWords, tarWords, srcLemmas, tarLemmas, srcPosTags, tarPosTags, sourceDependencyParse, targetDependencyParse, existingalignments, srcWordAlreadyAligned, tarWordAlreadyAligned): list
alignAdverbs(srcWordIndices, tarWordIndices, srcWords, tarWords, srcLemmas, tarLemmas, srcPosTags, tarPosTags, sourceDependencyParse, targetDependencyParse, existingalignments, srcWordAlreadyAligned, tarWordAlreadyAligned) : list
alignTextualNeighborhoodContentWords(sourceSent, targetSent, srcWordIndices, tarWordIndices, srcWords, tarWords, srcLemmas, tarLemmas, srcPosTags, tarPosTags, existingalignments, srcWordAlreadyAligned, tarWordAlreadyAligned, textualNeighborhoodAlignments, noContextualOverlapAlignments) : list
alignDependencyNeighborhoodForStopWords(sourceSent, targetSent, srcWordIndices, tarWordIndices, srcWords, tarWords, srcLemmas, tarLemmas, srcPosTags, tarPosTags, existingalignments, srcDependencyParse, tarDependencyParse, srcWordAlreadyAligned, tarWordAlreadyAligned) : list
alignTextualNeighborhoodPuncStopWords(srcWordIndices, tarWordIndices, srcWords, tarWords, srcLemmas, tarLemmas, srcPosTags, tarPosTags, existingalignments, srcWordAlreadyAligned, tarWordAlreadyAligned) : list

Config

punctuations: string
ppdbSim : float
theta1: float

util

isSublist(A,B) : boolean
get_commonNeighboringWords(sourceWords, targetWords) : list
dependencyTreeWithOffSets(parseResult): list
findParents(dependencies, wordIndex, word) : list
findChildren(dependencies, wordIndex, word) : list
findNeighborhoodSimilarities(sentence, wordIndex, leftSpan, rightSpan) : list

Text_Processing_spacy

self.CharacterOffsetEnd : int
self.CharacterOffsetBegin : int

parser(sentence) : dictionary
get_parseText(sentence) : dictionary, tokenized sentence
combine_lemmaAndPosTags(parserResult) : list
nerWordAnnotator(parserResult) : list
get_ner(parserResult) **:** list
getNamedEntities(sentence, tokenized_words) : list
is_Acronym(word, NE) : boolean
get_constituency_Tree(sentence): string
getDependencies(sentence): list
getCharOffset(sentence,word): list
get_combine_words_param(sent): dictionary

Figure 4.7: Class diagram of word aligner implementation

Design Considerations

We use Sultan et al. [7] approach and re-implement in an object-oriented way for easy debugging, maintenance and avoid code repetition. Whereas, Sultan et al. [7] do not use object-oriented approach. Furthermore, the code is designed in such a way that on runtime, the user can define which library; Spacy or NLTK, to be used for word alignment or for complete automatic short answer grader(ASAG) pipeline. For both libraries, the only pre-processing portion is changed, while word aligner block as shown in figure 4.6 remain same. Since design considerations for complete automatic short answer grader(ASAG) remains same, therefore, we do not describe design considerations in next chapter.

4.1.3 Results

Tables 4.1 and 4.2 shows results of word aligner for teacher and student answers taken from MRC dataset using Stanford CoreNLP(Sultan et al. [5]), NLTK and Spacy. All the libraries almost return same results. We show two tables; 4.1 contains examples where word aligner works better and 4.2 contains examples where word aligner fails to align properly.

Furthermore, word aligner can also be used as an assistant for human graders. Like, for a given teacher and student answer, it can highlight the similar words. This can save time for the human grader.

Teacher Answer	Student Answer	Stanford CoreNLP Aligner	NLTK Aligner	Spacy Aligner
A homogeneous transformation matrix is a 4 x 4 matrix that combines a rotation and a translation into a single compact form, thereby representing the transformation between two coordinate frames.	the,translation and rotation that apply to any point	[‘rotation’, ‘rotation’], [‘translation’, ‘translation’]	[‘rotation’, ‘rotation’], [‘translation’, ‘translation’]	[‘rotation’, ‘rotation’], [‘translation’, ‘translation’]
Rotation matrices are orthogonal, i. e. their inverse is equal to their transpose, and their determinant is equal to 1.	Determinate is 1	[‘determinant’, ‘Determinate’], [‘1’, ‘1’]	[‘determinant’, ‘Determinate’], [‘1’, ‘1’]]	[‘1’, ‘1’], [‘determinant’, ‘Determinate’]
An eigenvector,(of a matrix A),is a vector that doesnt change its direction when multiplied by A; in other words, an eigenvector obeys the relation Ax =,lambda,x, where,lambda,is a constant called an eigenvalue.	Eigenvectors are vectors. Eigen values are the magnitude of those vectors	[‘vector’, ‘vectors’], [‘eigenvector’, ‘Eigenvectors’], [‘is’, ‘are’], [‘eigenvalue’, ‘values’]	[‘eigenvector’, ‘Eigenvectors’], [‘vector’, ‘vectors’], [‘eigenvalue’, ‘values’]	[‘eigenvector’, ‘Eigenvectors’], [‘vector’, ‘vectors’], [‘eigenvalue’, ‘values’]
Separation of variables, educated guess,(Ansatz) , variation of parameters, numerical,(e. g. Runge - Kutta methods)	1. Solve by separating variables. 2. Euler method. 3. Runge Kutta method.	[‘Runge’, ‘Runge’], [‘Kutta’, ‘Kutta’], [‘methods’, ‘method’], [‘Separation’, ‘separating’], [‘variables’, ‘variables’]	[‘Runge’, ‘Runge’], [‘Kutta’, ‘Kutta’], [‘methods’, ‘method’], [‘Separation’, ‘separating’], [‘variables’, ‘variables’]	[‘Runge’, ‘Runge’], [‘Kutta’, ‘Kutta’], [‘methods’, ‘method’], [‘Separation’, ‘separating’], [‘variables’, ‘variables’]

Separation of variables, educated guess,(Ansatz) , variation of parameters, numerical,(e. g. Runge - Kutta methods)	* By Separation of Variables. * By Variation of parameters. * By Undetermined Coefficients	[‘Separation’, ‘Separation’], [‘of’, ‘of’], [‘variables’, ‘Variables’], [‘variation’, ‘Variation’], [‘of’, ‘of’], [‘parameters’, ‘parameters’]	[‘Separation’, ‘Separation’], [‘of’, ‘of’], [‘variables’, ‘Variables’], [‘variation’, ‘Variation’], [‘of’, ‘of’], [‘parameters’, ‘parameters’]	[‘Separation’, ‘Separation’], [‘of’, ‘of’], [‘variables’, ‘Variables’], [‘variation’, ‘Variation’], [‘of’, ‘of’], [‘parameters’, ‘parameters’]
An orthonormal basis of a vector space is a basis whose vectors are all unit vectors that are orthogonal to each other.	An orthonormal basis consists of unitvectors and are perpendicular to each other,$AA^T = I$	[’An’, ’An’], [’orthonormal’, ’orthonormal’], [’basis’, ’basis’], [’to’, ’to’], [’each’, ’each’], [’other’, ’other’]	[’An’, ’An’], [’orthonormal’, ’orthonormal’], [’basis’, ’basis’], [’to’, ’to’], [’each’, ’each’], [’other’, ’other’]	[[’An’, ’An’], [’orthonormal’, ’orthonormal’], [’basis’, ’basis’], [’to’, ’to’], [’each’, ’each’], [’other’, ’other’]]
We can end up in gimbal lock if we use Euler angles, i. e. since rotations are done sequentially, certain axes can get aligned, which reduces the degrees of freedom. Representing rotations with quaternions can be used to overcome this problem.	If Euler angles are used for expressing rotations, when two of the axes are in parallel configuration, there may occur a loss of one degree of freedom, causing a problem of gimber lock. To overcome this problem, quartanions, which are extension of complex numbers, are used to express angles. This representation solves the problem of Gimber lock.	[’to’, ’To’], [’overcome’, ’overcome’], [’this’, ’this’], [’problem’, ’problem’], [’Euler’, ’Euler’], [’angles’, ’angles’], [’which’, ’which’], [’of’, ’of’], [’freedom’, ’freedom’], [’used’, ’used’], [’use’, ’used’], [’if’, ’If’], [’degrees’, ’degree’], [’lock’, ’lock’], [’rotations’, ’rotations’], [’axes’, ’axes’], [’certain’, ’numbers’], [’Representing’, ’representation’]	[’to’, ’To’], [’overcome’, ’overcome’], [’this’, ’this’], [’problem’, ’problem’], [’Euler’, ’Euler’], [’angles’, ’angles’], [’which’, ’which’], [’of’, ’of’], [’freedom’, ’freedom’], [’used’, ’used’], [’use’, ’used’], [’degrees’, ’degree’], [’lock’, ’lock’], [’rotations’, ’rotations’], [’axes’, ’axes’], [’certain’, ’numbers’], [’Representing’, ’representation’]	[’to’, ’To’], [’overcome’, ’overcome’], [’this’, ’this’], [’problem’, ’problem’], [’Euler’, ’Euler’], [’angles’, ’angles’], [’which’, ’which’], [’of’, ’of’], [’freedom’, ’freedom’], [’used’, ’used’], [’use’, ’used’], [’degrees’, ’degree’], [’lock’, ’lock’], [’rotations’, ’rotations’], [’axes’, ’axes’], [’certain’, ’numbers’], [’Representing’, ’representation’]]

Table 4.1: Table shows teacher and student answers from MRC dataset using Stanford CoreNLP(Sultan et al. [5]), NLTK and Spacy, where word aligner works better

Teacher Answer	Student Answer	Stanford CoreNLP Aligner	NLTK Aligner	Spacy Aligner
A homogeneous transformation matrix is a **4 x 4** matrix that combines a rotation and a translation into a single compact form, thereby representing the transformation between two coordinate frames.	The homogeneous transform matrix is a **4x4** matrix that casts translation and rotation matrices into a single transformation matrix.	[‘matrix’, ‘matrix’], [‘is’, ‘is’], [‘a’, ‘a’], [‘into’, ‘into’], [‘a’, ‘a’], [‘single’, ‘single’], [‘matrix’, ‘matrix’], [‘that’, ‘that’], [‘transformation’, ‘transformation’], [‘rotation’, ‘rotation’], [‘translation’, ‘translation’], [‘homogeneous’, ‘homogeneous’], [‘transformation’, ‘transform’]	[‘matrix’, ‘matrix’], [‘is’, ‘is’], [‘a’, ‘a’], [‘into’, ‘into’], [‘a’, ‘a’] , [‘single’, ‘single’], [‘matrix’, ‘matrix’], [‘that’, ‘that’] , [‘transformation’, ‘transformation’], [‘homogeneous’, ‘homogeneous’], [‘rotation’, ‘rotation’], [‘translation’, ‘translation’], [‘transformation’, ‘transform’]	[‘matrix’, ‘matrix’], [‘is’, ‘is’], [‘a’, ‘a’], [‘into’, ‘into’], [‘a’, ‘a’], [‘single’, ‘single’], [‘matrix’, ‘matrix’], [‘that’, ‘that’], [‘transformation’, ‘transformation’] [‘homogeneous’, ‘homogeneous’], [‘rotation’, ’rotation’], [‘translation’, ‘translation’], [‘transformation’, ‘transform’]
Rotation matrices are orthogonal, i. e. their inverse is equal to their transpose, and their determinant is equal to 1.	$A^T = A^{-1}$	[‘1’, ‘1’]	[‘1’, ‘1’]	[‘1’, ‘1’]
Rotation matrices are orthogonal, i. e. their inverse is equal to their transpose, and their determinant is equal to 1.	1. Preserves length. 2. preserves the angles. 3. orthogonal matrix. 4. $A^{-1} = A^T$. 5. $AA^T = A^TA$. 6. $AA^T = I$	[‘1’, ‘1’], [‘matrices’, ‘matrix’], [‘orthogonal’, ‘orthogonal’]	[‘1’, ‘1’], [‘matrices’, ‘matrix’], [‘orthogonal’, ‘orthogonal’]	[‘1’, ‘1’], [‘matrices’, ‘matrix’], [‘orthogonal’, ‘orthogonal’], [‘their’, ‘I’]

The characteristic polynomial of a matrix X is given as $\|X - lambdaI\| = 0$ and is used for calculating the eigenvalues of X.	det X - lambda I = 0 is the characteristic polynomial of matrix X. The polynomial is used to determine the eigenvalues and eigenvectors of the matrix.	[’X’, ’X’], [’-’, ’-’], [’lambda’, ’lambda’], [’I’, ’I’], [’0’, ’0’], [’The’, ’the’], [’characteristic’, ’characteristic’], [’polynomial’, ’polynomial’], [’of’, ’of’], [’matrix’, ’matrix’], [’X’, ’X’], [’is’, ’is’], [’used’, ’used’], [’the’, ’the’], [’eigenvalues’, ’eigenvalues’], [’calculating’, ’determine’]	[’X’, ’X’], [’-’, ’-’], [’lambda’, ’lambda’], [’I’, ’I’], [’=’, ’=’], [’0’, ’0’], [’The’, ’the’], [’characteristic’, ’characteristic’], [’polynomial’, ’polynomial’], [’of’, ’of’], [’matrix’, ’matrix’], [’X’, ’X’], [’is’, ’is’], [’used’, ’used’], [’the’, ’the’], [’eigenvalues’, ’eigenvalues’], [’calculating’, ’determine’]	[’lambda’, ’lambda’], [’I’, ’I’], [’=’, ’=’], [’0’, ’0’], [’The’, ’the’], [’characteristic’, ’characteristic’], [’polynomial’, ’polynomial’], [’of’, ’of’], [’X’, ’X’], [’-’, ’-’], [’is’, ’is’], [’used’, ’used’], [’the’, ’the’], [’eigenvalues’, ’eigenvalues’], [’matrix’, ’matrix’], [’calculating’, ’determine’], [’X’, ’X’]
The characteristic polynomial of a matrix X is given as $\|X - lambdaI\| = 0$ and is used for calculating the eigenvalues of X.	$\|A - lambdaI\| = 0$ It is used to calculate the eigenvalues of a matrix	[’-’, ’-’], [’lambda’, ’lambda’], [’I’, ’I’], [’0’, ’0’], [’the’, ’the’], [’eigenvalues’, ’eigenvalues’], [’of’, ’of’], [’a’, ’a’], [’matrix’, ’matrix’], [’is’, ’is’], [’used’, ’used’], [’calculating’, ’calculate’]	[’lambda’, ’lambda’], [’I’, ’I’], [u’=’, ’=’], [’0’, ’0’], [’the’, ’the’], [’eigenvalues’, ’eigenvalues’], [’of’, ’of’], [’a’, ’a’], [’matrix’, ’matrix’], [’is’, ’is’], [’used’, ’used’], [’calculating’, ’calculate’]	[’lambda’, ’lambda’], [’I’, ’I’], [’the’, ’the’], [’eigenvalues’, ’eigenvalues’], [’of’, ’of’], [’a’, ’a’], [’matrix’, ’matrix’], [’=’, ’=’], [’0’, ’0’], [’is’, ’is’], [’used’, ’used’], [’calculating’, ’calculate’]

Table 4.2: Table shows teacher and student answers from MRC dataset using Stanford CoreNLP(Sultan et al. [5]), NLTK and Spacy, where word aligner does not works properly

4.1.4 Limitations of Word Aligner

Although word aligner by Sultan et al. [5] outperforms existing approaches. But, still, there are limitations due to which performance may be affected.

- Word aligner is unable to represent and use distant term dependencies among words.
- It is also incapable of aligning word phrases, as Sultan et al. [5] consider only dependency relations between words rather than phrases.
- Sultan et al. [5] uses paraphrase database(PPDB) to check word similarity. Since, no word similarity database/resource is perfect, therefore, this also does not align words in some cases.
- Furthermore, results of pre-processing such as word tokenization also affects the alignment process. If we look in example 1 of table 4.2, system does not aligns "4 x 4" in teacher's answer with "4x4" in student's response. Because word tokenizer of all 3 libraries considers "4x4" in student's response as one word. While, in teacher's answer, "4 x 4" are tokenized as 3 words.
- Word aligner only considers current knowledge from student and teacher answer. For suppose, if we look at example 2 in table 4.2, Teacher answer is "Rotation matrices are orthogonal, i. e. their inverse is equal to their transpose, and their determinant is equal to 1 " and student answer is "$A^T = A^{-1}$". Now, a student has also written correctly that matrices are orthogonal. But, word aligner only aligns "1".
- Generally, word aligner does not aligns properly when there are equations or mathematical terms involved in answers. If we look at example 4, teacher answer includes "$|X - lambdaI|$" and student has written "det X - lambda I", but aligner does not aligns the "determinant" with "||". Similarly in example 5, teacher has written "$|X - lambdaI|$" and student has written "$|A - lambdaI|$". But word aligner does not aligns "A" with "X"

Chapter 5

Features for Automatic Short Answer Grader

Figure 5.1 describes block diagram of automatic short answer grader(ASAG) system. For a given student and teacher answer, we extract four features; word alignment, length ratio, word embeddings, and question demoting. These features are fed to regression model where final score of student answer is determined based on similarity with teacher answer. The following section gives an explanation of these four features.

5.1 Word Alignment

The purpose of this feature is to align the content words in student and teacher answer, that are identical and semantically similar. The similarity score between teacher and student answer is computed as, the ratio of the sum of aligned content words in student and teacher answer to the sum of total contents words in student and teacher answer. We already have discussed the process of aligning words in the previous chapter(4), so we will not discuss here.

Furthermore, in order to avoid penalizing lengthy student answers that still include valid answers, Sultan et al. [7] also considers second feature, that determines coverage of aligned content words only in teacher response. This feature is named as *coverage* of the teacher answer by the student response.

5.2 Semantic Vector Similarity

Since word alignment has some limitations as we listed in the previous chapter, therefore, semantic vector similarity is employed. This feature uses word embeddings model trained by Baroni et al.

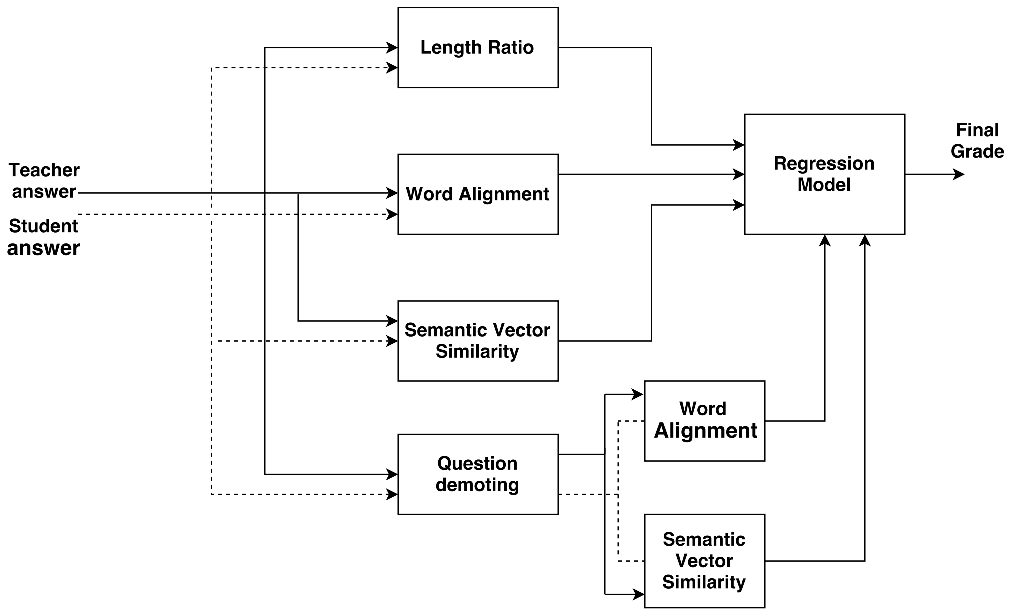


Figure 5.1: Block Diagram shows features fed to regression model in automatic short answer grading system

[23] using word2vec toolkit. This model is trained on a corpus of around 2.8 billion words from ukWaC[59], English Wikipedia[60], and British National Corpus[61]. For a given student and teacher answer, first, we compute the semantic vector of the answer, which is the sum of all content words embeddings. For each content word, we get the vector of 400 dimensions, we sum all the vectors of each content word related to each answer. Now, we have two 400 dimensions vector for each answer, we use cosine similarity on student and teacher answer, which is used as a feature for the regression model.

Figure 5.2 show embedding vectors of two sentences; sentence1 = "Five men are dead from an accident", sentence2 = "Five people died from a collision". We plot 400-dimensional vector and reduce it to 2-dimensions using t-distributed stochastic neighbor embedding(TSNE) for easier visualization. Since both the sentences contain synonym word, therefore, cosine similarity between two vectors is 0.99. This shows both the sentences are semantically similar. Furthermore, this gives better results only, when both the sentences have an equal number of content words and they are similar. While, when one sentence has more content words than the other sentence. Even if some content words are similar in both sentences, overall, similarity will be very low.

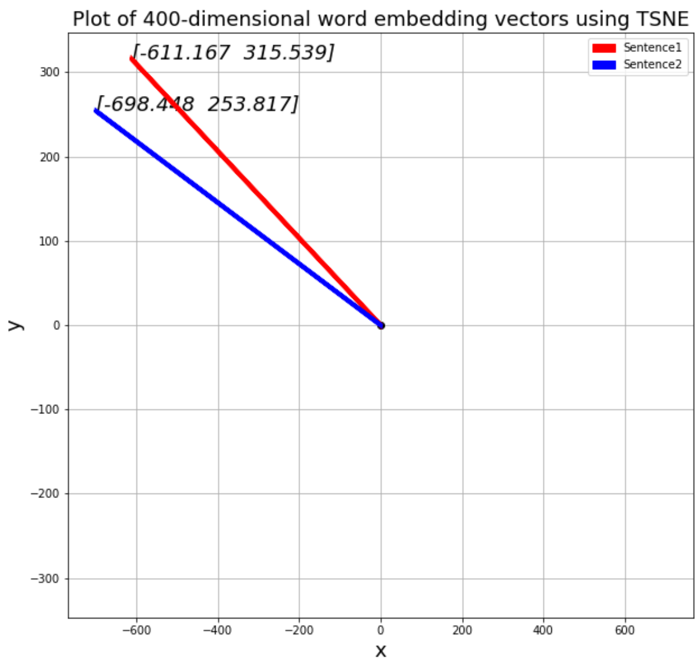


Figure 5.2: Figure shows embedding vectors of two sentences, reduced from 400-dimensional vector to 2-dimensions using TSNE

5.3 Question Demoting

The goal of this feature is to avoid giving marks to student answer for repeating words from the question. This is determined by recomputing above features again after ignoring words that appear in question from student and teacher answer. The reason Sultan et al. [7] does, because it improves the overall performance of the system.

5.4 Length Ratio

The idea of this feature is to determine whether student answer contains enough detail or not. This is done by computing ratio of the number of content words in student answer to the number of content words in teacher answer, this is also used as a feature. According to Sultan et al. [7] this features increases the overall performance. However, in some cases, this feature does not perform well. For suppose, student answer contains irrelevant content words, this feature does not check whether those words are similar to teacher answer. But, just computes the ratio of content words in student and the teacher answer, and assigns the final score.

5.5 Regression Model

In order to compute the final score of student answer, we fed all four features; word alignment, semantic vector similarity, question demoting, and length ratio, along with their assigned scores to train the ridge regression model(Scikit-learn [62]). After training and saving optimized parameters, we use the same model to predict on test data. We evaluate trained model based on Pearson's r coefficient and mean square error(MSE).

5.6 Class diagram of ASAG system

Figure 5.3 shows class diagram of ASAG system. We divide our implementation into four classes. ReadDataSet class is used to read dataset and pre-process it before extracting features. Features class is used to extract all four features as shown in Figure 5.1. These features are later on fed to the regression model. RidgeModel class contains regression model. Lastly, TrainAndApplyGrader class passes required features to regression model and predict on the trained model.

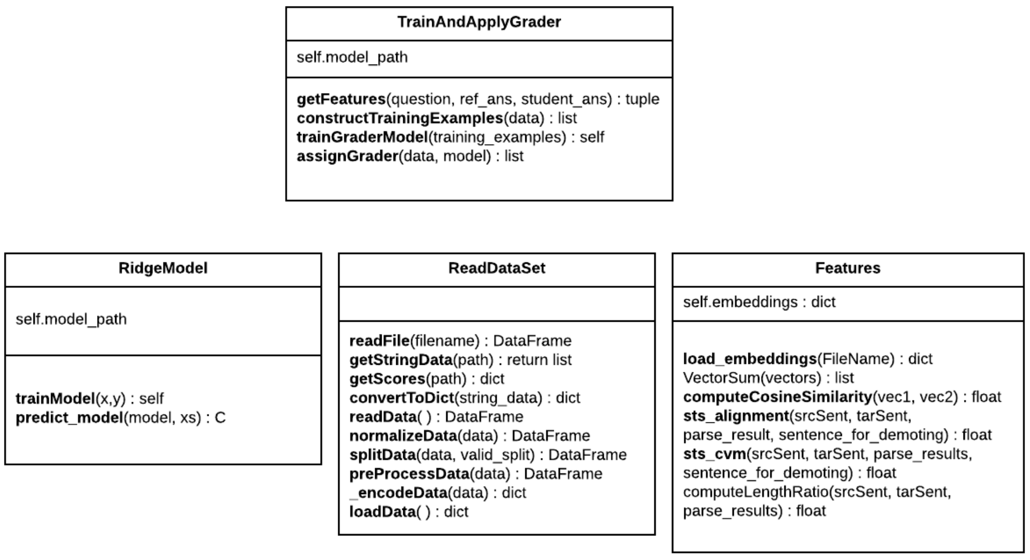

TrainAndApplyGrader
self.model_path
getFeatures(question, ref_ans, student_ans) : tuple
constructTrainingExamples(data) : list
trainGraderModel(training_examples) : self
assignGrader(data, model) : list
RidgeModel
self.model_path
trainModel(x,y) : self
predict_model(model, xs) : C
ReadDataSet
readFile(filename) : DataFrame
getStringData(path) : return list
getScores(path) : dict
convertToDict(string_data) : dict
readData( ) : DataFrame
normalizeData(data) : DataFrame
splitData(data, valid_split) : DataFrame
preProcessData(data) : DataFrame
_encodeData(data) : dict
loadData( ) : dict
Features
self.embeddings : dict
load_embeddings(FileName) : dict
VectorSum(vectors) : list
computeCosineSimilarity(vec1, vec2) : float
sts_alignment(srcSent, tarSent, parse_result, sentence_for_demoting) : float
sts_cvm(srcSent, tarSent, parse_results, sentence_for_demoting) : float
computeLengthRatio(srcSent, tarSent, parse_results) : float

Figure 5.3: Class diagram of ASAG system

5.7 Results

Tables 5.1 and 5.2 shows comparison of grades assigned to some students answers by human graders and ASAG systems; Stanford(Sultan et al. [7]), NLTK and Spacy. All the grades are normalized to the scale of 0 to 1.

Table 5.1 shows cases where grades assigned by system are closer to the grades assigned by human graders.

Table 5.2 contains examples where system grades incorrectly. This is may be due to the difference in grades of both the human graders, and the system does not learn correctly. Furthermore, if we look at example 3, where student does not write any answer; "No answer", but the system still gives very less grade. This is because of length ratio(as discussed in 5.4) feature, since it only checks the ratio of content words present in student and teacher answer, without making sure of similarity. Although, in this case, the difference is very less, if the student writes any content words that does not resemble the teacher response. This feature still assigns some scores to the student answer.

In addition to this, If we take a look at example 4, where grades by the both human grades are same, but, still system does not reward student response correctly. This is may be due to coverage feature and similarity score(as discussed in 5.1). Although, student answer is correct, but word aligner does not aligns all the content words present in student and teacher answer. Also, semantic vector similarity(as discussed in 5.2) of student and teacher answer will be low, as number of content words in teacher's answer are greater than student response.

Question	Teacher Answer	Student Answer	Human grader 1	Human grader 2	Average grade	Sultan	NLTK	Spacy
What properties do rotation matrices have?	Rotation matrices are orthogonal, i.e. their inverse is equal to their transpose, and their determinant is equal to 1.	Rotation matrix has determinant of 1. Rotation matrix preserves the norm of the vector. Rotation matrix are orthogonal matrix. Hence the inverse is equal to its transpose. Rotation matrix are linear	1	1	1	1.06	0.99	0.94
What properties do rotation matrices have?	Rotation matrices are orthogonal, i.e. their inverse is equal to their transpose, and their determinant is equal to 1.	* When rotation matrix and its transpose is multiplied we get an Identity matrix. i.e Rotation matrix is equal to its inverse. * The determinant of the rotation matrix is 1.	1	1	1	0.95	1.07	1.02
Write down the characteristic polynomial of a matrix X. What is this polynomial used for?	The characteristic polynomial of a matrix X is given as $\|X - \lambda I\| = 0$ and is used for calculating the eigenvalues of X.	$\\|A - \lambda I\\| = 0$ It is used to calculate the eigenvalues of a matrix	1	1	1	1.05	0.88	0.88

Apart from the SVD, what are two other matrix decompositions and what are the forms of the decomposed matrices?	PLU (permutation matrix + lower triangular matrix + upper triangular matrix), QR (orthogonal matrix + upper triangular matrix).	Eigen decomposition: A= P D Pinv where Pinv is inverse of P and D is diagnol matrix . P contains tthe eigen vector and D contains eigen value. LU decomposition: A= L U, whrer L is a lower triangular matrix and U is a upper triangular matrix	1	1	1	0.88	0.86	0.8
Given a matrix A and its SVD decomposition $U\Sigma V^T$, what do the columns of V represent? What about the columns of U?	The columns of V are the eigenvectors of A^TA; the columns of U are the eigenvectors of AA^T.	V is the normalized eigen vector of A*A matrix, where A* is the transpose of A U is the normalized eigen vector of AA*. The eigen vectors are arranged in the descending order of associated singular values	0.5	0.5	0.5	0.62	0.58	0.57

Table 5.1: Comparison of grades assigned to students answers by human graders and ASAG systems; Stanford(Sultan et al. [7]), NLTK and Spacy. It shows examples where human grades and system grades are closer or equal

Question	Teacher Answer	Student Answer	Human grader 1	Human grader 2	Average grade	Sultan	NLTK	Spacy
What is a homogeneous transform?	A homogeneous transformation matrix is a 4×4 matrix that combines a rotation and a translation into a single compact form, thereby representing the transformation between two coordinate frames.	homogeneous transform is [0,0,0,1] which is used in transformation matrix which contains the rotation matrix and translation vector	0	1	0.5	0.79	0.75	0.79
What is an orthonormal basis of a vector space?	An orthonormal basis of a vector space is a basis whose vectors are all unit vectors that are orthogonal to each other.	They represent the direction towards the vectors are pointing.	0	0	0	0.5	0.47	0.54
What properties do rotation matrices have?	Rotation matrices are orthogonal, i.e. their inverse is equal to their transpose, and their determinant is equal to 1.	No answer	0	0	0	0.05	0.16	0.03

What problem can arise if we use Euler angles for expressing rotations? What other representation can be used to overcome this problem?	We can end up in gimbal lock if we use Euler angles, i.e. since rotations are done sequentially, certain axes can get aligned, which reduces the degrees of freedom. Representing rotations with quaternions can be used to overcome this problem.	Gimmberlock arises in euler rotations. Quaternions are used to overcome this.	1	1	1	0.64	0.51	0.61
What are eigenvectors and eigenvalues?	An eigenvector (of a matrix A) is a vector that doesnt change its direction when multiplied by A; in other words, an eigenvector obeys the relation $Ax = lambdax$, where $lambda$ is a constant called an eigenvalue.	Eigenvalues are the values obtained after solving the characteristic equation of a matrix i.e they are the roota of the characteristic equation. Eigenvectors will contain the maximum variant and important part of the data projected. $\|\|Ax - b\|\| = 0$ Determinant of the characteristic equation is zero	0.25	0.75	0.5	0.7	0.65	0.76

Given a matrix A and its SVD decomposition $U\Sigma V^T$, what do the columns of V represent? What about the columns of U?	The columns of V are the eigenvectors of A^TA; the columns of U are the eigenvectors of AA^T.	The columns of V consist of the eigen vectors of A^TA. The columns of U consist of the eigen vectors of AA^T	1	1	1	0.76	0.71	0.73

Table 5.2: Comparison of grades assigned to students answers by human graders and ASAG systems; Stanford(Sultan et al. [7]), NLTK and Spacy. It shows cases where system grades almost incorrectly

Chapter 6

Datasets

We use datasets to evaluate performance of Sultan et al. [7] approach implemented using NLTK and Spacy. Two datasets; Texas and MRC are used for evaluation of ASAG system. While, common misspelling dataset is used to evaluate the performance of spell checking techniques. However, Sultan et al. [7] does not use any spell checking techniques, but words that are exactly identical or semantically similar in student and teacher answer are considered for final grading.

6.1 Common Misspellings Dataset

We downloaded a dataset of common misspellings with their correct answers from Wikipedia[63], around 670 common misspelled words are selected randomly to evaluate the performance of spell correction techniques.

6.2 Datasets for ASAG

Texas Dataset

To compare performance between existing approach implemented using Stanford CoreNLP library and current approach implemented using NLTK and Spacy libraries, we use publicly available dataset. We already have discussed about this dataset in section 3.5. So, we will not discuss here.

Mathematics for Robotics and Control(MRC) Course Dataset

In this project, we also use a dataset from MRC course to do the performance comparison. It contains 10 questions answered by 17 students; in total, we have 170 answers for 10 questions. It is graded by two human graders; Aleksandar Mitrevski and Santosh George Thoduka. We also compute average score of two human graders. In short, for each student answer, we have three grades. All the scores are normalized to the scale of 0 to 1. Therefore, we expand this dataset to 510 answers for 10 questions, so that we can use it for evaluation purpose easily.

Chapter 7

Evaluations and Results

7.1 Evaluation of Spell-correction Techniques

We evaluate the performance of spell-correction using Minimum edit distance and N-grams on dataset downloaded from Wikipedia[63], that contains misspelled words along with correct words.

After evaluating the performance of spell corrector using N-grams, and Minimum edit distance on a dataset, both methods have achieved an accuracy of 69.19% and 59.75% respectively.

However, the accuracy of both techniques is not sufficient, may be due to following reasons:

- It is not optimal to find misspelled words because PyEnchant library contains a lot of words that seems incorrect/out of the dictionary.
  - For example: If we write “Always” as “Alway” it considers as incorrect and suggested words are : [’Alway’, ’Amway’, ’Al way’, ’Al-way’, ’Away’, ’Always’, ’Allay’, ’Alwin’]
  - Since word “Alway” has maximum correlation coefficient therefore it replaces as “Alway”
  - We can also see nouns such as “Allay”, “Alwin” in list of suggested words.
- However, in some cases, it does spell correction efficiently.
  - For example: when our sentence is “Alway rememer oovember and decemer”
  - It returns: [‘Always’, ‘remember’, ‘November’, ‘and’, ‘December’]

It can be concluded that above results are not much convincing to replace the misspelled word with correct word, but since in automatic short answer grading we need to detect misspelled words in student answers, this can be done with the help of PyEnchant library.

7.2 Evaluation of Automatic Short answer grading system

7.2.1 Evaluation Metrics

Evaluation metrics are used to distinguish algorithms based on their performances. In order to evaluate Sultan et al. [7] approach, we compare performances based on Pearson correlation coefficient, runtime test and ease of implementation. The following section describes these metrics:

Pearson Correlation Coefficient

It measures the strength of a linear association between two variables. It is determined by drawing a line of best fit through the data of two variables, and it indicates how close the data points are to the new line/model. It is denoted by r. The value of r ranges from +1 to -1, where 0 indicates there is no association between two variables, the value greater than 0 shows a positive association; means the value of both the variables increases and value less than 0 indicates a negative association; which means the value of both the variables decreases[64].

In our case, we compute r between predicted scores by trained ridge regression model and correct scores assigned by the teacher. This r we use to compare performance on datasets among different libraries.

Root mean square error(RMSE)

RMSE is a measurement of differences between predicted values by model and real value. In other words, it represents how close the predicted and actual data are to each other. It is commonly used for regression analysis to determine the error of the model. It is computed as:

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(P_i - R_i)}{n}} \tag{7.1}$$

Where P_i is set of prediction values, R_i is set of real values and n is the total number of values. Its value ranges from 0 to 1, where 0 represents that real and predicted values are exactly equal. While 1 represents, both values are dissimilar[65][66].

Runtime Test

The most computationally expensive step of this system is an extraction of alignment features by removing stop words. Runtime complexity depends upon the number of content word pairs in two input sentences(student and teacher answer), given the feature extraction process, runtime complexity is O($n_c.m_c$), where n_c and m_c are the number of content words in two sentences.

In results section 7.2.2, we show the number of questions graded per minute in MRC dataset. However, since we do not have exact system configuration as Sultan et al. [7], therefore, we do not compare runtime on Texas dataset among different libraries. Furthermore, Runtime is conducted on system with configuration; Intel$^{®}$ Core i5-4210M, 2.4GHz.

7.2.2 Results

Table 7.1: Performance on Texas dataset using Sultan et al. [7] approach implemented using Stanford CoreNLP, NLTK and Spacy

System	Pearson's r	RMSE
Sultan et al. [7]	0.63	0.85
NLTK	0.55	0.20
Spacy	0.55	0.20

Table 7.2: Performance on MRC dataset using Sultan et al. [7] approach implemented using Stanford CoreNLP, NLTK and Spacy

System	Pearson's r	RMSE	Runtime(Average questions graded per min)
Sultan et al. [7]	0.66	0.26	9
NLTK	0.62	0.28	2
Spacy	0.62	0.27	7

Observations about Results

Results shown in tables 7.1 and 7.2 are different among Stanford CoreNLP, NLTK and Spacy due to number of reasons:

- Pre-processing of all three libraries is different from each other, that includes computation of dependency parse tree, named entity recognition(NER), lemmatization, and part of speech(POS) tagger(as shown in figure 4.6)
- NLTK uses dependency parse tree, NER and POS tagger from Stanford CoreNLP library. Results are still dissimilar from Sultan et al. [7] because of different versions of these parameters and different lemmatizer and tokenizer. Since NLTK uses some computation of parameters from Stanford CoreNLP, therefore "Pearson's r" of NLTK in case of MRC dataset is closer to Sultan et al. [7] result, unlike Spacy.
- NER module using Stanford CoreNLP and NLTK has only three pre-defined categories; "ORGANIZATION", "LOCATION", and "PERSON". While, SPACY has many of them such as "PERSON", "NORP(Nationalities or religious or political groups)", "FACILITY(Buildings, highways, airports, etcetera)", "ORGANIZATION", "GPE(Countries, states, cities)", "LANGUAGE" and many more[67]. Therefore, alignment of named entities by word aligner using NLTK and Stanford CoreNLP vs Spacy is not always similar. However, in MRC and Mohler

dataset, we do not have named entities, so this module does not align any words. But, this matters when we want to align some general sentences.

- Furthermore, dependency parse tree(syntactic dependencies) module of Spacy is a bit different as compare to NLTK and Stanford CoreNLP. At least names of dependency relations are not similar in Spacy, as we see in figure 4.3. Sultan et al. [7] does not discuss in his implementation, how exactly these relations are constructed. Therefore, it is difficult to modify dependency relations in Sultan et al(2016) algorithm, so that it is made completely suitable for Spacy.

- Tokenization is also different for all three libraries, which also affect the results of part of speech, lemmatization and named entities of the sentence. For suppose:
  - Sentence = "(infix expressions are converted to postfix, (i.e, 3+2 is changed to 32+)"
  - This sentence is taken from the Texas dataset(3.5)
  - NLTK tokenizer result is: ['(' , 'infix', 'expressions', 'are', 'converted', 'to', 'postfix', ',', '(', 'i.e', ',', '3+2', 'is', 'changed', 'to', '32+', ')'] . We can see numbers and signs between them are not properly tokenized.
  - Stanford CoreNLP tokenizer result is: ['(' , 'infix', 'expressions', 'are', 'converted', 'to', 'postfix', ',', '(', 'i.e', ',', '3', '+2', 'is', 'changed', 'to', '32','+', ')']. The result of tokenization using Stanford CoreNLP is little bit different than NLTK tokenizer. As we can see, '+2' is considered as one word.
  - Spacy tokenizer gives result as: ['(', 'infix', 'expressions', 'are', 'converted', 'to', 'postfix', ',', '(', 'i.e', ',', '3', '+', '2', 'is', 'changed', 'to', '32', '+', ')']. This tokenizer gives appropriate word tokenization.

Detailed analysis for each Question from MRC Dataset

Table 7.3 shows Pearson's r for each question using all three libraries and among two human graders. For each question, we determine Pearson's r between system grade and an average grade of both human graders. The table gives an overview of questions that system has graded correctly or incorrectly. Like, Question 2, 4, 7, 8, 9, 10 are graded correctly as their Pearson's r is closer to 1. While Pearson's r of remaining questions varies a lot as compared to average human grade. Furthermore, in case of questions 1 and 7, human grades for student answers are not same. But, Pearson's r of question 7 is still closer to 1. Figures (7.2, 7.3, 7.1, 7.5, 7.4, 7.6, 7.8, 7.7, 7.9) show comparison among system's grade using all three libraries with average grade for questions 1, 5 and 10.

Table 7.3: Pearson Correlation Coefficient for each Question using Stanford CoreNLP, NLTK and Spacy

Questions	Human Graders Pearson's r	Stanford Pearson's r	NLTK Pearson's r	Spacy Pearson's r
Question 1	-1	0.05	0.27	0.03
Question 2	1	0.96	0.95	0.95
Question 3	1	-0.34	-0.14	-0.28
Question 4	1	0.97	1	0.93
Question 5	0.86	-0.14	-0.44	-0.29
Question 6	1	0.36	0.44	0.22
Question 7	-1	0.93	0.89	0.9
Question 8	1	0.97	0.89	0.98
Question 9	0.95	0.87	0.96	0.88
Question 10	0.97	1	1	1

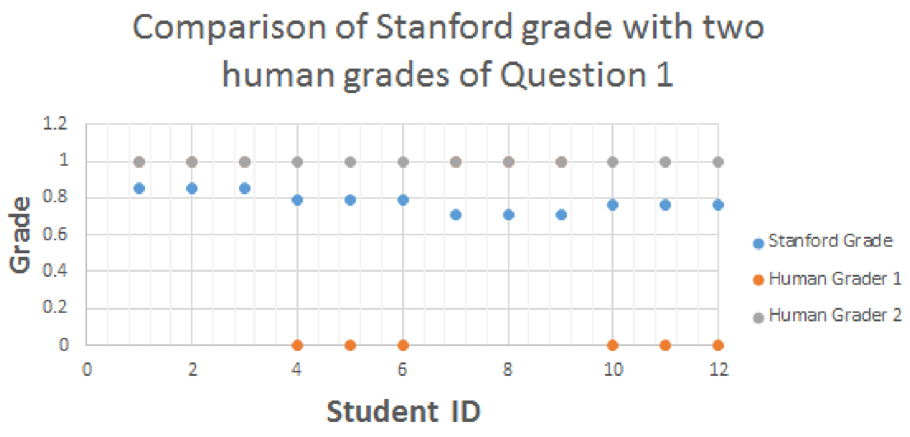


Figure 7.1: Figure shows comparison of grades of question 1 assigned to all students by Stanford CoreNLP system and Human graders. We can clearly see the difference of grades between both human graders and system's output, as system's grade is not much closer to the average grade

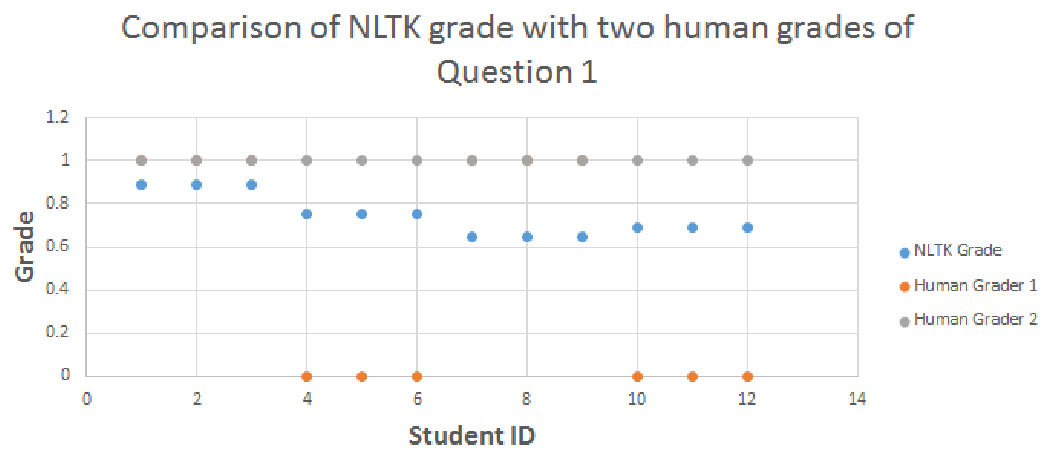


Figure 7.2: Figure shows comparison of grades of question 1 assigned to all students by NLTK system and Human graders. NLTK grade is bit closer to the average grade. Therefore, Pearson'r of NLTK is more than Stanford CoreNLP.

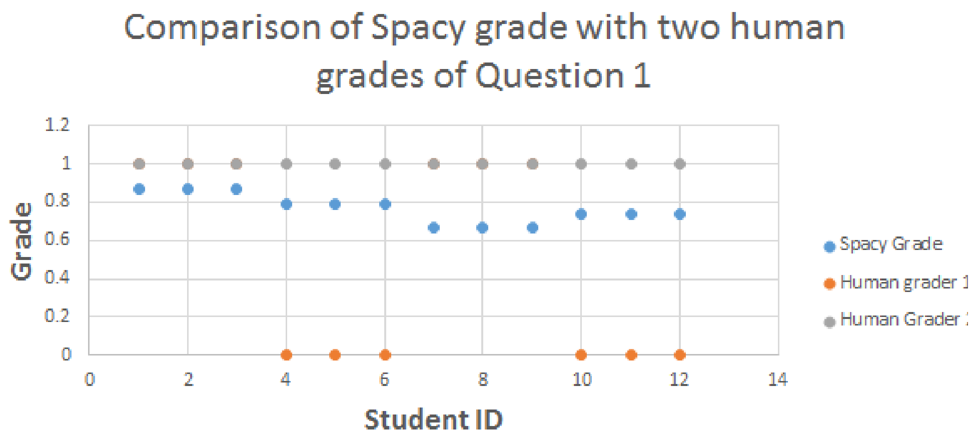


Figure 7.3: Figure shows comparison of grades of question 1 assigned to all students by Spacy system and Human graders. Here, we can clearly see the difference of grades between both human graders and system's output, as system's grade is not much closer to the average grade

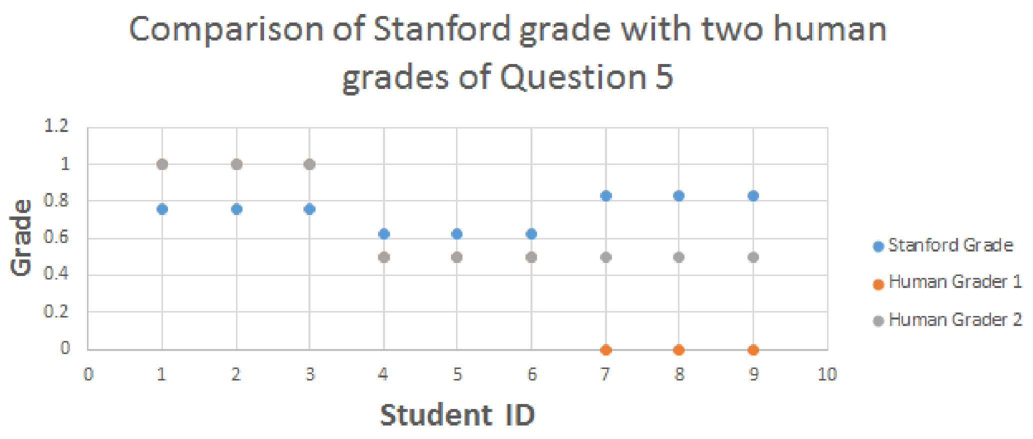


Figure 7.4: Figure shows comparison of grades of question 5 assigned to all students by Stanford CoreNLP system and Human graders. System's grade and average grade are in opposite direction, due to which Pearson'r is negative.

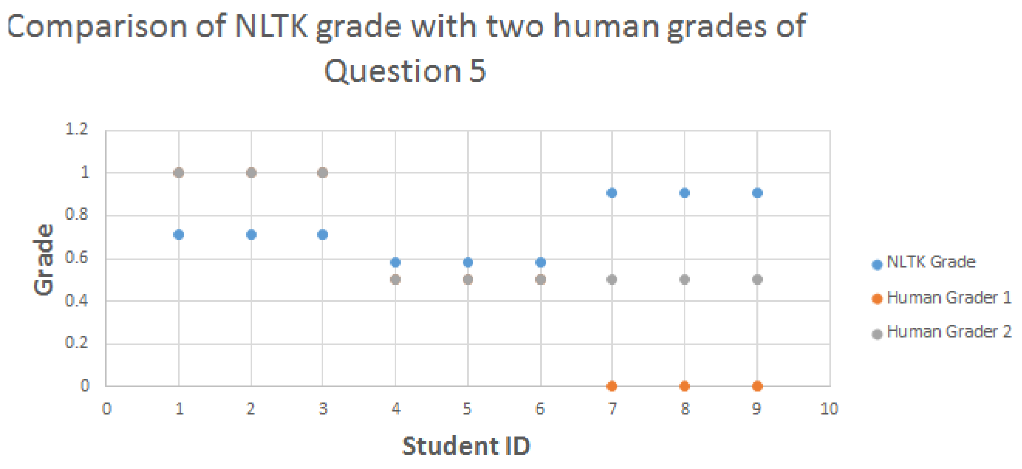


Figure 7.5: Figure shows comparison of grades of question 5 assigned to all students by NLTK system and Human graders. System's grade and average grade are in opposite direction, due to which Pearson'r is negative.

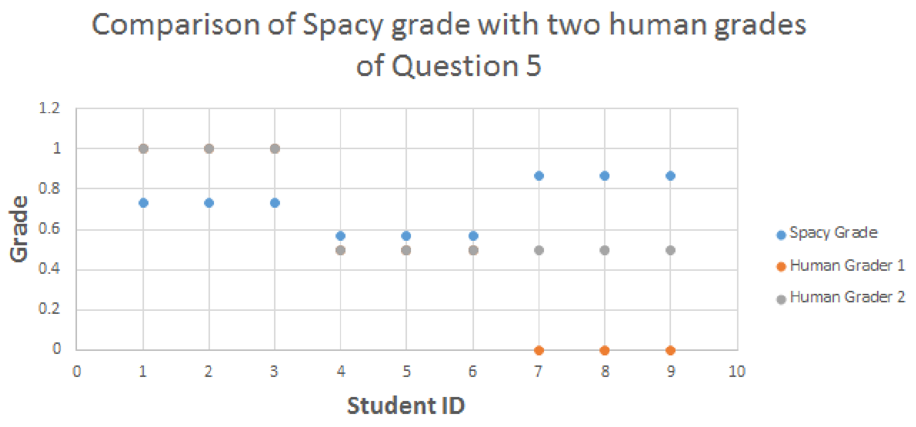


Figure 7.6: Figure shows comparison of grades of question 5 assigned to all students by Spacy system and Human graders. System's grade and average grade are in opposite direction, due to which Pearson'r is negative.

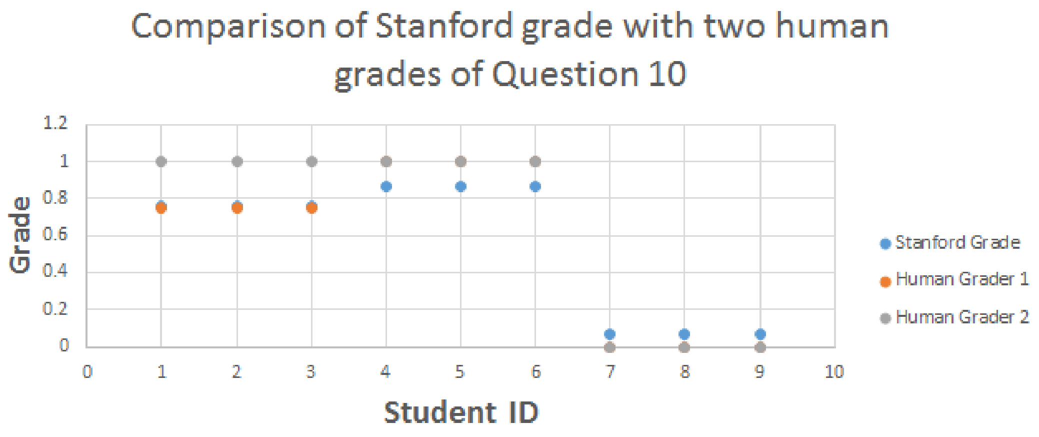


Figure 7.7: Figure shows comparison of grades of question 10 assigned to all students by Stanford CoreNLP system and Human graders. System's grade and average grade are almost closer to each other.

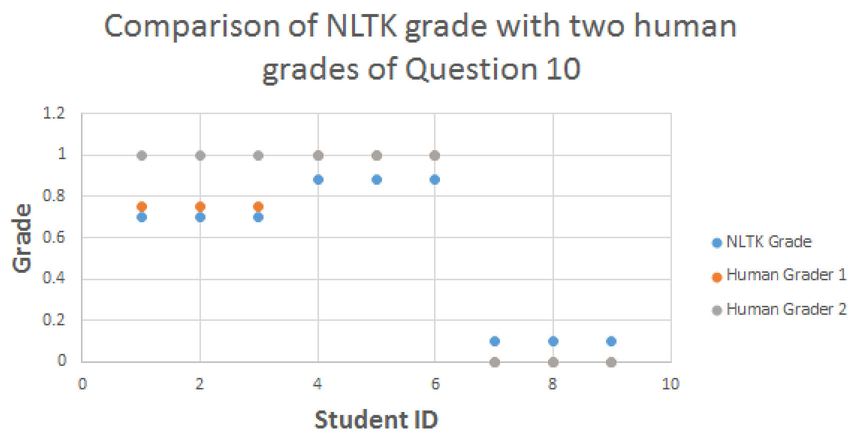


Figure 7.8: Figure shows comparison of grades of question 10 assigned to all students by NLTK system and Human graders. System's grade and average grade are almost closer to each other.

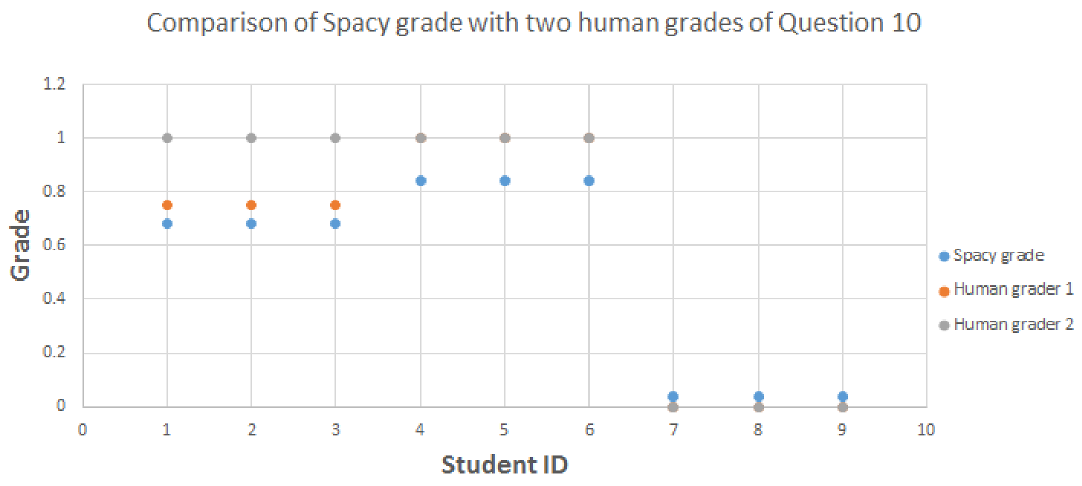


Figure 7.9: Figure shows comparison of grades of question 10 assigned to all students by Spacy system and Human graders. System's grade and average grade are almost closer to each other.

Chapter 8

Conclusions and Future Work

Contributions

- We re-implemented one of the state-of-the-art approach using two libraries; NLTK and Spacy.
- The publicly available dataset; Texas dataset, was selected and performance was compared among Stanford, NLTK and Spacy libraries.
- Word aligner was implemented in two different libraries, and this will be released as an open source contribution.
- MRC dataset was collected and results were compared among Stanford, NLTK and Spacy libraries.
- Nbgrader tool was used to convert Ipython Jupyter notebooks into text files, to make an evaluation on MRC dataset easier.
- The code of complete project was designed in an object-oriented way for easy debugging, maintenance and avoid code repetition.
- The complete code of the project is available on GitHub (`https://github.com/rameshjesswani/Semantic-Textual-Similarity`) as well as submitted in CDs.
- Runtime tests, word alignment results, grades assigned to students responses using MRC dataset, are submitted using excel sheets in CDs.

Conclusion

In this work, we reviewed the state-of-the-art approaches for automatic short answer grading, and evaluated single approach already implemented using Stanford CoreNLP library, we use the same approach and re-implement with the help of two libraries; NLTK and Spacy, on publicly available Texas and an in-house created dataset of MRC. On Texas dataset, Stanford CoreNLP library has better Pearson correlation coefficient and lowest RMSE than NLTK and Spacy libraries. While on MRC dataset, all three libraries shows comparative results on Pearson correlation coefficient, RMSE and runtime.

Furthermore, results of word aligner on MRC dataset using all 3 libraries shows that, it can also be implemented individually to assist the human grader by finding semantically similar words.

Future Work

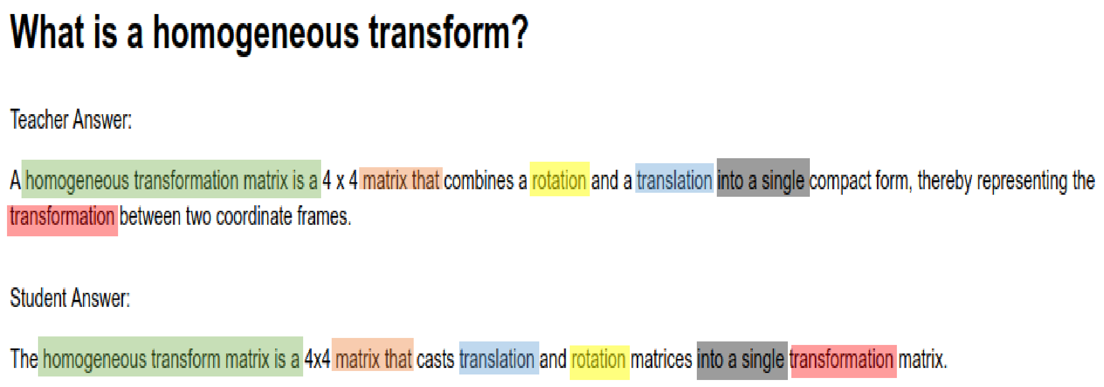
For future work, improvements in Pearson correlation coefficient may be possible by using different word embeddings model. For suppose, if we train model for word embeddings based on the corpus of only particular course, the results may appear much better as compare to current ones. In addition to this, Google[68] and Facebook [69] have released their word embeddings model, these can also be used and may provide better results.

Another task for the future work will be to determine the optimum pre-processing techniques such as tokenization, part of speech tagger, dependency parsers, named entity recognition, lemmatization, and the paraphrase database, as these can also improve the overall results of ASAG system.

In addition to this, other features proposed by the state-of-the-art can also be integrated to enhance the performance of the system. Mohler et al. [42] uses pseudo-relevance feedback technique to improve the performance of the system, by integrating the correct answers provided by the students.

Integration with Nbgrader

Results of word aligner can be integrated with the nbgrader in Jupyter Ipython notebook, as this will help human grader to grade the students' answers faster. Figure 8.1 shows demo of integration of the word aligner with the nbgrader. To give an idea, how similar words can be highlighted, we just highlight manually.



What is a homogeneous transform?

Teacher Answer:

A homogeneous transformation matrix is a 4 x 4 matrix that combines a rotation and a translation into a single compact form, thereby representing the transformation between two coordinate frames.

Student Answer:

The homogeneous transform matrix is a 4x4 matrix that casts translation and rotation matrices into a single transformation matrix.

Figure 8.1: Demo of integration of the word aligner with the nbgrader in Jupyter Ipython notebook

Bibliography

[1] Daniel Jurafsky and James H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition.* Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition, 2000. ISBN 0130950696.

[2] Steven Burrows, Iryna Gurevych, and Benno Stein. The eras and trends of automatic short answer grading. *International Journal of Artificial Intelligence in Education*, 25(1):60–117, Mar 2015. ISSN 1560-4306. doi: 10.1007/s40593-014-0026-8. URL `https://doi.org/10.1007/s40593-014-0026-8`.

[3] David Callear, Jenny Jerrams-Smith, Victor Soh, Dr. Jenny Jerrams-smith, and Hants Po Ae. Caa of short non-mcq answers. In *Proceedings of the 5th International CAA conference*, 2001. URL `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.58.2210&rep=rep1&type=pdf`.

[4] Tom Mitchell, Terry Russell, Peter Broomhead, and Nicola Aldridge. Towards robust computerised marking of free-text responses. 2002. URL `http://hdl.handle.net/2134/1884`.

[5] Md. Arafat Sultan, Steven Bethard, and Tamara Sumner. Back to basics for monolingual alignment: Exploiting word similarity and contextual evidence. *TACL*, 2:219–230, 2014. URL `https://tacl2013.cs.columbia.edu/ojs/index.php/tacl/article/view/292`.

[6] Marie catherine De Marneffe and Christopher D. Manning. Stanford typed dependencies manual, 2008.

[7] Md. Arafat Sultan, Cristobal Salazar, and Tamara Sumner. Fast and easy short answer grading with high accuracy. In *NAACL HLT 2016, The 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, San Diego California, USA, June 12-17, 2016*, pages 1070–1075, 2016. URL `http://aclweb.org/anthology/N/N16/N16-1123.pdf`.

[8] Mason and Grove-Stephensen. Automated free text marking with paperless school. *IN: Proceedings of the 6th CAA Conference*, 2002.

[9] Raheel Siddiqi, Christopher J. Harrison, and Rosheena Siddiqi. Improving teaching and learning through automated short-answer marking. *TLT*, 3(3):237–249, 2010. doi: 10.1109/TLT.2010.4. URL https://doi.org/10.1109/TLT.2010.4.

[10] Richard Klein, Angelo Kyrilov, and Mayya Tokman. Automated assessment of short free-text responses in computer science using latent semantic analysis. In *Proceedings of the 16th Annual Joint Conference on Innovation and Technology in Computer Science Education*, ITiCSE '11, pages 158–162, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0697-3. doi: 10.1145/1999747.1999793. URL http://doi.acm.org/10.1145/1999747.1999793.

[11] Word tokenization- stanford nlp - professor dan jurafsky & chris manning. https://https://www.youtube.com/watch?v=nfoudtpBV68&list=PL6397E4B26D00A269, . accessed: 2017-11-04.

[12] Stop words. https://en.wikipedia.org/wiki/Stop_words. accessed: 2017-11-25.

[13] Hinrich Schutze Christopher D. Manning, Prabhakar Raghavan. *Introduction to Information Retrieval.* Cambridge University, UK, 1st edition, 2008.

[14] Wael H. Gomaa and Aly A. Fahmy. A survey of text similarity approaches. *International Journal of Computer Applications*, 68(13), Apr 2013.

[15] What are n-grams. http://text-analytics101.rxnlp.com/2014/11/what-are-n-grams.html?google_comment_id=z12ft1pifnrdjhwsi225jzbqgmv3hlovs.

[16] An intro to parts of speech and pos tagging nlp dan jurafsky and chris manning. https://www.youtube.com/watch?v=LivXkL2DO_w&t=1s, . accessed: 2017-11-04.

[17] Named-entity recognition. https://en.wikipedia.org/w/index.php?title=Named-entity_recognition&oldid=811428985, . accessed: 2017-11-25.

[18] Introduction to named entity recognition. http://www.datacommunitydc.org/blog/2013/04/a-survey-of-stochastic-and-gazetteer-based-approaches-for-named-entity-recognition. accessed: 2017-11-25.

[19] Parse tree. https://en.wikipedia.org/wiki/Parse_tree, . accessed: 2017-11-26.

[20] Word embeddings. https://medium.com/deeper-learning/glossary-of-deep-learning-word-embedding-f90c3cec34ca, . accessed: 2017-11-30.

[21] Introduction to word embeddings. https://https://www.youtube.com/watch?v=Eku_pbZ3-Mw, . accessed: 2017-11-04.

[22] Words as vectors. https://iksinc.wordpress.com/tag/continuous-bag-of-words-cbow/, . accessed: 2017-12-01.

[23] Marco Baroni, Georgiana Dinu, and Germán Kruszewski. Don't count, predict! a systematic comparison of context-counting vs. context-predicting semantic vectors. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 238–247, Baltimore, Maryland, June 2014. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/P14-1023.

[24] What is library. https://blog.alexdevero.com/programming-languages-libraries-and-frameworks/. accessed: 2017-11-28.

[25] Natural language toolkit. http://www.nltk.org/. accessed: 2017-11-28.

[26] 5 heroic python nlp libraries. https://elitedatascience.com/python-nlp-libraries. accessed: 2017-11-28.

[27] Snowball stemmers. http://text-processing.com/demo/stem/. accessed: 2017-11-28.

[28] Spacy documentation. https://spacy.io/usage/facts-figures, . accessed: 2017-11-28.

[29] Spacy vs nltk. https://www.quora.com/What-are-the-advantages-of-Spacy-vs-NLTK, . accessed: 2017-11-28.

[30] Spacy models. https://spacy.io/models/, . accessed: 2017-11-28.

[31] Spacy facts and figures. https://spacy.io/usage/facts-figures, . accessed: 2017-11-28.

[32] Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Prismatic Inc, Steven J. Bethard, and David Mcclosky. The stanford corenlp natural language processing toolkit. In *In ACL, System Demonstrations*, 2014.

[33] Pyenchant. https://http://pythonhosted.org/pyenchant/. accessed: 2017-11-28.

[34] Nbgrader. https://nbgrader.readthedocs.io/en/stable/. accessed: 2018-28-01.

[35] Claudia Leacock and Martin Chodorow. C-rater: Automated scoring of short-answer questions. *Computers and the Humanities*, 37(4):389–405, Nov 2003. ISSN 1572-8412. doi: 10.1023/A:1025779619903. URL https://doi.org/10.1023/A:1025779619903.

[36] E. Klein G. Pullum Gazdar, G. and I. Sag. *Generalised Phrase Structure Grammar*. Oxford: Blackwell, 1985.

[37] Michael Mohler, Rada and Mihalcea. Text-to-text semantic similarity for automatic short answer grading. In *Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics*, EACL '09, pages 567–575, Stroudsburg, PA, USA, 2009. Association for Computational Linguistics. URL `http://dl.acm.org/citation.cfm?id=1609067.1609130`.

[38] S. S. Kusumawardani U. Hasanah, A. E. Permanasari and F. S. Pribadi. A review of an information extraction technique approach for automatic short answer grading. *1st International Conference on Information Technology, Information Systems and Electrical Engineering (ICITISEE), Yogyakarta*, pages 192–196, 2016. doi: 10.1109/ICITISEE.2016.7803072.

[39] Daniel M. Cer, Marie-Catherine de Marneffe, Daniel Jurafsky, and Christopher D. Manning. Parsing to stanford dependencies: Trade-offs between speed and accuracy. In Nicoletta Calzolari, Khalid Choukri, Bente Maegaard, Joseph Mariani, Jan Odijk, Stelios Piperidis, Mike Rosner, and Daniel Tapias, editors, *LREC*. European Language Resources Association, 2010. ISBN 2-9517408-6-7. URL `http://dblp.uni-trier.de/db/conf/lrec/lrec2010.html#CerMJM10`.

[40] Christopher Harrison Raheel Siddiqi. A systematic approach to the automated marking of short-answer questions. In *Proceedings of the Twelfth International Multitopic Conference, pages 329 332, Karachi, Pakistan. IEEE, 2008.*

[41] Latent semantic analysis. `http://mccormickml.com/2016/03/25/lsa-for-text-classification-tutorial/`. accessed: 2017-12-6.

[42] Michael Mohler, Razvan Bunescu, and Rada Mihalcea. Learning to grade short answer questions using semantic similarity measures and dependency graph alignments. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies - Volume 1*, HLT '11, pages 752–762, Stroudsburg, PA, USA, 2011. Association for Computational Linguistics. ISBN 978-1-932432-87-9. URL `http://dl.acm.org/citation.cfm?id=2002472.2002568`.

[43] Aria D. Haghighi, Andrew Y. Ng, and Christopher D. Manning. Robust textual inference via graph matching. In *Proceedings of the Conference on Human Language Technology and Empirical Methods in Natural Language Processing*, HLT '05, pages 387–394, Stroudsburg, PA, USA, 2005. Association for Computational Linguistics. doi: 10.3115/1220575.1220624. URL `https://doi.org/10.3115/1220575.1220624`.

[44] Myroslava Dzikovska, Rodney Nielsen, Chris Brew, Claudia Leacock, Danilo Giampiccolo, Luisa Bentivogli, Peter Clark, Ido Dagan, and Hoa Trang Dang. Semeval-2013 task 7: The joint student response analysis and 8th recognizing textual entailment challenge. In *Second Joint Conference on Lexical and Computational Semantics (*SEM), Volume 2: Proceedings*

of the Seventh International Workshop on Semantic Evaluation (SemEval 2013), pages 263–274, Atlanta, Georgia, USA, June 2013. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/S13-2045.

[45] kaggle. https://www.kaggle.com/. accessed: 2017-12-8.

[46] Semeval. https://en.wikipedia.org/wiki/SemEval. accessed: 2017-12-8.

[47] Shourya Roy, Himanshu S. Bhatt, and Y. Narahari. An iterative transfer learning based ensemble technique for automatic short answer grading. *CoRR*, abs/1609.04909, 2016. URL http://arxiv.org/abs/1609.04909.

[48] Andrew Hickl and Jeremy Bensley. A discourse commitment-based framework for recognizing textual entailment. In *Proceedings of the ACL-PASCAL Workshop on Textual Entailment and Paraphrasing*, RTE '07, pages 171–176, Stroudsburg, PA, USA, 2007. Association for Computational Linguistics. URL http://dl.acm.org/citation.cfm?id=1654536.1654571.

[49] Andrew Hickl, Jeremy Bensley, John Williams, Kirk Roberts, Bryan Rink, and Ying Shi. Recognizing textual entailment with lccs groundhog system. In *In Proc. of the Second PASCAL Challenges Workshop*, 2005.

[50] Dipanjan Das and Noah A. Smith. Paraphrase identification as probabilistic quasi-synchronous recognition. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 1 - Volume 1*, ACL '09, pages 468–476, Stroudsburg, PA, USA, 2009. Association for Computational Linguistics. ISBN 978-1-932432-45-9. URL http://dl.acm.org/citation.cfm?id=1687878.1687944.

[51] Lushan Han, Abhay Kashyap, Tim Finin, James Mayfield, and Jonathan Weese. Umbc ebiquity-core: Semantic textual similarity systems, 2013.

[52] Chris Brockett. Aligning the rte 2006 corpus. Technical report, June 2007. URL https://www.microsoft.com/en-us/research/publication/aligning-the-rte-2006-corpus/.

[53] Nathanael Chambers, Daniel Cer, Trond Grenager, David Hall, Chloe Kiddon, Bill MacCartney, Marie-Catherine de Marneffe, Daniel Ramage, Eric Yeh, and Christopher D. Manning. Learning alignments and leveraging natural logic. In *Proceedings of the ACL-PASCAL Workshop on Textual Entailment and Paraphrasing*, RTE '07, pages 165–170, Stroudsburg, PA, USA, 2007. Association for Computational Linguistics. URL http://dl.acm.org/citation.cfm?id=1654536.1654570.

[54] Kapil Thadani and Kathleen McKeown. Optimal and syntactically-informed decoding for monolingual phrase-based alignment. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 254–259, Portland, Oregon, USA, June 2011. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/P11-2044`.

[55] Xuchen Yao, Benjamin Van Durme, Chris Callison-Burch, and Peter Clark. A lightweight and high performance monolingual word aligner. In *ACL (2)*, pages 702–707. The Association for Computer Linguistics, 2013. ISBN 978-1-937284-51-0. URL `http://dblp.uni-trier.de/db/conf/acl/acl2013-2.html#YaoDCC13`.

[56] Chris Callison-Burch Xuchen Yao, Benjamin Van Durme and Peter Clark. Semi-markov phrase-based monolingual alignment. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 590–600. Association for Computational Linguistics, 2013b.

[57] Juri Ganitkevitch, Benjamin Van Durme, and Chris Callison-burch. Ppdb: The paraphrase database. In *In HLT-NAACL 2013*, 2013.

[58] Md Arafat Sultan. *Short-Text Semantic Similarity: Algorithms and Applications*. PhD thesis, University of Colorado Boulder, 2016.

[59] Wacky - the web-as-corpus kool yinitiative. `http://wacky.sslmit.unibo.it/doku.php`. accessed: 2018-06-01.

[60] free encyclopedia. `https://en.wikipedia.org/wiki/Main_Page`, . accessed: 2018-06-01.

[61] British national corpus. `http://www.natcorp.ox.ac.uk`. accessed: 2018-06-01.

[62] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[63] Lists of common misspellings. `https://en.wikipedia.org/wiki/Wikipedia\protect\kern+.2222em\relaxLists_of_common_misspellings/A`. accessed: 2017-09-03.

[64] Pearson product-moment correlation. `https://statistics.laerd.com/statistical-guides/pearson-correlation-coefficient-statistical-guide.php`. accessed: 2018-10-01.

[65] Assessing the fit of regression models. `https://www.theanalysisfactor.com/assessing-the-fit-of-regression-models/`, . accessed: 2018-23-01.

[66] Rms error. `http://statweb.stanford.edu/~susan/courses/s60/split/node60.html`, . accessed: 2018-23-01.

[67] Named entity recogntion. `https://spacy.io/api/annotation#named-entities`, . accessed: 2018-08-01.

[68] word2vec. `https://code.google.com/archive/p/word2vec/`. accessed: 2018-28-01.

[69] facebook word embeddings. `https://github.com/facebookresearch/fastText`. accessed: 2018-28-01.