THESIS

A QUESTIONNAIRE INTEGRATION SYSTEM BASED ON QUESTION CLASSIFICATION AND SHORT TEXT SEMANTIC TEXTUAL SIMILARITY

Submitted by

Yu Qiu

Department of Computer Science

In partial fulfillment of the requirements

For the Degree of Master of Science

Colorado State University

Fort Collins, Colorado

Fall 2018

Master's Committee:

Advisor: Sangmi Lee Pallickara

Shrideep Pallickara
Kaigang Li

ABSTRACT

A QUESTIONNAIRE INTEGRATION SYSTEM BASED ON QUESTION CLASSIFICATION AND SHORT TEXT SEMANTIC TEXTUAL SIMILARITY

Semantic integration from heterogeneous sources involves a series of NLP tasks. Existing research has focused mainly on measuring two paired sentences. However, to find possible identical texts between two datasets, the sentences are not paired. To avoid pair-wise comparison, this thesis proposed a semantic similarity measuring system equipped with a precategorization module. It applies a hybrid question classification module, which subdivides all texts to coarse categories. The sentences are then paired from these subcategories. The core task is to detect identical texts between two sentences, which relates to the semantic textual similarity task in the NLP field. We built a short text semantic textual similarity measuring module. It combined conventional NLP techniques, including both semantic and syntactic features, with a Recurrent Convolutional Neural Network to accomplish an ensemble model. We also conducted a set of empirical evaluations. The results show that our system possesses a degree of generalization ability, and it performs well on heterogeneous sources.

DEDICATION

I would like to dedicate this thesis to my wife Cong Yin, my advisor Sangmi Lee Pallickara and everyone who care about me.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

1.1 Background

Big Data and Machine Learning have provided advances in information technology that offer substantial promise to public health and clinical research, and these techniques potentially play a critical role in enhancing disease prevention. The goal of this paper is to use Big Data and Machine Learning techniques to support clinical and public health research areas. More specifically, this study examines obesity, which is one of the most common, serious, and costly public health issues and one of the major risk factors associated with many serious health conditions (e.g., heart disease, type-2 diabetes). We collected public-access data and requested restricted data from the two structured, widely used longitudinal data sources:

- **NLSY97 dataset**

  NLSY97 consists of a nationally representative sample of approximately 9,000 US youths who were 12 to 16 years old as of December 31, 1996. Round 1 of the survey took place in 1997. Measures in employment, schooling, training, health, environment, and similar indices of wellness are included.

- **Add Health dataset**

  The Add Health cohort (started in 1994-95, grades 7-12) has followed US students into young adulthood to complete four in-home interviews. The data include measures of social, economic, psychological, and physical well-being with contextual data on the family, neighborhood, community, school, friendships, and peer groups, providing unique opportunities to study how social environments and behaviors in adolescence are linked to health and achievement outcomes in young adulthood.

The differences between the NLSY97 and Add Health datasets indicate that they would be complementary information resources for the purposes of this study. The premise of this study is that if we integrate the two datasets, the prediction results will improve. Therefore, the purpose of this paper is to semantically integrate the two questionnaire datasets, and unifying two tables requires identifying the common columns. Therefore, to semantically integrate two questionnaires, the key is to find identical questions. This paper proposed a system for finding possible equivalent questions across two questionnaire databases.

1.2 Scientific Challenges

The heterogeneity and large size of the questionnaire datasets introduce a set of scientific challenges:

- **Efficiency:** : Each questionnaire usually contains thousands of questions. If the system has to do the element-wise comparison, the time complexity will be $O(N^2)$, which is time consuming.

- **Heterogeneity:** Heterogeneous sources will introduce potential bias and noise when calculating the similarity between questions. Many NLP tasks have domain limitation because of the domain specificity of the input data. We need to train and evaluate our model so that it can have a degree of generality.

1.3 Research Questions

To integrate two questionnaires, there are two main problems that need to be addressed. Research questions that are explored in this thesis include:

RQ-1 How can we integrate semantically similar questions from multiple questionnaires? The system should be able to proceed with semantic comparisons at the coarse level to reduce the processing time. This question is addressed and resolved in Section 3.1 and Section 4.1.

RQ-2 How can we contrast similarity for semantically and syntactically alike questions? The system should also be able to calculate the semantic similarity between question pairs at the fine level. This question is addressed and resolved in Section 3.2 and Section 4.2.

1.4 Approach Summary

Our Natural Language Process (NLP) system consists of two modules. One module is responsible for Question Classification (QC), which is used to reduce running times. This module also contains some minor techniques for data cleaning. The second module mainly finishes the Short Text Semantic Textual Similarity job. The combination of these two modules makes it possible to find potentially equivalent questions within a reasonable time.

The purpose of QC is to represent the semantic classes of answers that correspond to targeted questions. Li & Roth [1] thought QC is a task that, given a question, maps it to one of the predefined k classes, which provides a semantic constraint on the sought-after answer. Sundblad [2] proposed that QC can loosely be defined as follows: given a question (represented by a set of features), assign the question to a single category or a set of categories (answer types). Loni [3], Laokulrat [4], and many other researchers have also offered definitions. Question classification is a vital part of a Question Answering (QA) system, and it can also be useful in our system. The scenario we faced involved thousands of questions, and highly similar or identical questions that potentially can be integrated should be classified into similar categories, though we do not need to search for their actual answers. Successfully extracting target questions will reduce the amount of processing time so that we do not need to waste time calculating similarity scores between sentences that are not similar to each other. Our first module applies question classification techniques, classifying questions into predefined coarse and grained classes. And then for the polar questions, we implement a triplet extractor to extract subject-predicate-object pairs to further determine the key information.

The ST-STS module is the core module of our system. Measuring Semantic Textual Similarity (STS) is the task of determining the similarity between two different text passages. Techniques for

detecting similarity between documents (long texts) have been researched in depth, and approaches include but are not limited to analyzing shared words and extracting document topics. However, such methods are effective only when dealing with long documents. In short texts, word co-occurrence may be rare or even nonexistent. The system we implemented is based on computing the similarity between short texts (mainly questions), and it employed both conventional NLP techniques and Deep Learning techniques. The purpose of the conventional NLP module is to extract NLP features using some traditional NLP techniques. We applied feature generation tools including Bag-of-Word, Bag-of-Dependency, N-gram overlap, Syntactic Structure overlap, and WordNet-Augmented overlap. All of the extracted features were input to several regression models to train a classification model. The Deep Learning Module uses a training dataset to train a Neural Network model. We built the sentence representations from the Word2Vec pretrained model and input them to a Recurrent Convolutional Neural Network. The final similarity score is equal to the average of the above two sub-modules scores.

Chapter 2

Related Works

There have been efforts to perform analytics over scientific data collections [5]. These efforts typically incorporate support for an underlying storage framework [6–12] and job scheduling [13]. These have included efforts that drive analytics based on queries [14–17], end-to-end frameworks [18], sketching algorithms [19], and ensemble methods [20, 21]. The models thus constructed may be deployed in settings as diverse as stream scheduling [22, 23]to virtualized environments [24].

2.1 Question Classification

There are many existing approaches to Question Target Classification, or Question Classification. Traditionally, questions are categorized based on their intents. Li and Roth [1] proposed a hierarchical classifier, which consists of a Coarse Classifier and a Fine Classifier. The Coarse Classifier maps input to 6 coarse classes. Afterwards, each coarse class label is expanded to a fixed set of fine classes. The Fine Classifier then classifies the input questions to different fine classes. To train and test the module, Li and Roth represented each question as a list of primitive features, including words, pos tags, named-entities, and so on. Subsequent work about QC processes introduced question categories from the perspective of user-intent analysis, including Navigational, Informational, Transactional [25], and Social Questions [26], or combined intents with the contents, such as Solution, Reason, or Fact, which was introduced by Bu, Zhu, et al. [27] Some researchers classify the questions into more vertical domains, such as Weather, Restaurants, and Maps, the purpose of which is to achieve a better organized knowledge base and more accurate answers [28].

This vertical taxonomy directs the QC problem toward a topic classification problem, which is a basic task in text classification. The content of a given sentence (question) is fully exploited, such as its lexical features (e.g., n-grams), syntactic features (e.g., parse trees), and semantic (e.g., WordNet-based) features. Therefore, based on these textual features, many models have been

developed and applied in QC. For example, specific lexical features are more important for determining the topic, and these methods are independent with languages [29, 30]. Syntactic and semantic features combined with machine learning models (e.g., support vector machines) are also capable of classification [1, 31, 32].

Some scholars have focused on the customization of a classification taxonomy in restricted domains, intending to improve the accuracy of QC through the analysis of domain characteristics. However, Hao [1] found that taxonomies for restricted domains have not demonstrated obvious accuracy advantages. Furthermore, this customization may lead to poorer universality and narrower adaptability. Laokularat [4] summarized the significance of QTC as follows: (1) QTC reduces the volume of candidate answers (2) helps review different question types and design corresponding solutions, and (3) filters out irrelevant answers. In our system, the intention is to reduce the number of comparisons between sentence pairs instead of reducing the volume of the answer pool. Generally speaking, the more categories in need of mapping, the lower the classification accuracy obtained. Based on our unique requirements, we would like to balance this tradeoff when choosing a suitable QC algorithm.

2.2 Semantic Textual Similarity

In general, there is extensive literature on measuring the similarity between documents or long texts; some ideas on measuring similarity between short texts or sentences are also derived from those works. The problem lies in the fact that these approaches need adequate information to perform well, and most likely we cannot find adequate information in single sentences or short texts. For example, two long, similar texts are likely to have enough co-occurring words, but, at the sentence level, two similar sentences might easily fail to share common words. Three main kinds of approaches are popularly used to compute semantic similarity: word co-occurrence approaches, corpus-based approaches and hybrid approaches.

Word co-occurrence approaches are most frequently used in applications such as information retrieval(IR). The most widely used word co-occurrence methods are called "bag-of-words" mod-

els. Usually the IR systems have a pre-compiled word list with n words. This list generally consists of millions of items in order to include all meaningful words in the language. Each document is represented using these words as a vector in n-dimensional space. The relevant documents are then retrieved based on the similarity between two document vectors.

Some research has focused on improving word co-occurrence approaches. One extension of word co-occurrence approaches is the use of a lexical dictionary to compute the similarity of a pair of words taken from two sentences. Sentence similarity is calculated from lexical relations between the terms appearing in a sentence and those appearing in others [33]. Some pattern matching methods that are commonly used in text mining are also applied [34]. The difference between pattern matching methods and pure word co-occurrence methods is that pattern matching methods incorporate local structural information. A meaning is conveyed in a limited set of patterns where each is represented using a regular expression to provide generalization. The problem with these approaches is that they require a complete pattern set for each meaning of a word. It asks for manual pattern set compilation, and there seems no automated way to do it.

Corpus-based approaches use the statistical information of words in a corpus. One well-known corpus-based approach is Latent Semantic Analysis (LSA). LSA uses a word by passage matrix formed to reect the presence of words in each of the passages used. This matrix is decomposed by singular value decomposition (SVD), and its dimensionality is reduced by removing small singular values. Finally, the sentences to be compared are represented in this reduced space as two vectors containing the meaning of their words. The similarity score is calculated as the similarity of these two vectors [35, 36]. Another well-known approach among corpus-based approaches is Hyperspace Analogues to Language (HAL) [37]. This approach is closely related to LSA because they both capture the meaning of a word by using lexical co-occurrence information. Unlike LSA, which builds an information matrix of words by text units of paragraphs or documents, HAL builds a word-by-word matrix based on word co-occurrence within a moving window. Subsequently a sentence vector is formed by adding together the word vectors for all words in the sentence. Similarity between two sentences is calculated using a metric such as Euclidean distance. However, the

authors experimental results showed that HAL was not as promising as LSA in the computation of similarity for short texts. This limitation might due to the method of building the memory matrix. Possibly, the word-by-word matrix does not capture sentence meaning well, and the sentence vector becomes diluted as large numbers of words are added to it.

Although LSA and HAL do use word co-occurrence information, their key feature is the use of corpora, which enables them to find similarity in sentences with no co-occurring words. The main drawbacks of these approaches at the sentence level are the failure to use syntactic information and the sparseness of the vector representation. Besides, these methods might ignore very similar sentences if the sentences have no words in common, and vice-versa, they might regard unrelated sentences as being similar just because they share common words. For example:

"How old are you?" and *"What is your age?"*

"My neighbour has a dog with four legs." and *"My neighbour has four legs."*

Some researchers indicated that negations and antonyms are not processed by these approaches. For example, *"He is a teacher."* and *"He is not a teacher."* are considered very similar. For many previous researches about calculating short text similarity, this is considered a flaw while it is not too important in our scenario.

There are also hybrid approaches that use both corpus-based and knowledge-based techniques. Li tried to overcome the limitations of both techniques by forming the word vector entirely based on the words in the compared sentences, then computing the semantic similarity by combining information drawn from a structured lexical database and from corpus statistics [38]. Mihalcea proposed a combined unsupervised method that uses six WordNet-based measures and two corpus-based measures and combines the results to show how these measures can be used to derive a short-text similarity measure [39]. The major disadvantage of this method is that it computes the similarity of words using eight different methods, which is not computationally efficient.

Olivia proposed a syntax-based measure for short-text semantic similarity, SyMSS. SyMSS captures and combines syntactic and semantic information to compute the semantic similarity of two sentences. Semantic information is obtained from WordNet, and syntactic information is

obtained through a deep parsing process that finds the phrases that make up the sentence as well as the phrases syntactic functions [40]. Islam and Inkpen presented a method for measuring the semantic similarity between short texts using a corpus-based measure of semantic word similarity and normalized and modified versions of the Longest Common Subsequence(LCS) string matching algorithm [41].

Another area related to our task is Paraphrase Identification (PI). This process is especially useful for overcoming the challenge of high redundancy in Twitter and the sparsity inherent in Twitter users short texts. Many researchers have investigated ways of automatically detecting paraphrases on formal texts like newswire texts. Qiu proposed a supervised, two-phase framework that detects dissimilarity between sentences and makes its paraphrase judgement based on the significance of such dissimilarities [42]. Das and Smith introduced a probabilistic model, which makes use of three quasi-synchronous grammar models as components. They then combined the model with a complementary logistic regression model based on lexical overlap features [43]. Socher, Huang, et al. introduced a method for paraphrase detection based on recursive autoencoders (RAE). The RAE targets vector representations. These researchers built the unsupervised RAEs based on an unfolding objective and learned feature vectors for phrases in syntactic trees. By combining the RAEs and a dynamic pooling layer which computes a fixed-sized representation from the variable-sized matrices, the pooled representation is used as input to the classifier [44]. Ji and Eisenstein designed a new discriminative term-weighting metric TF-KLD which includes the term frequency and the KL-divergence. They then combined the latent representation from matrix factorization as features with fine-grained n-gram overlap features in a classification algorithm to achieve the task [45]. There are many ideas and goals that Paraphrase Identification shares with our task. First, PI usually focuses on short texts or sentence pairs. Second, PI normally cares more about whether two sentences are semantically identical, the degree of similarity is not that important. Our system uses some good ideas from these studies. However, we still applied similarity measurements in our system because some cases that can be integrated are not paraphrases. Consider a test case in the Microsoft Research Paraphrase Corpus (MSR): "A BMI of 25 or above is considered overweight;

30 or above is considered obese." and "A BMI between 18.5 and 24.9 is considered normal, over 25 is considered overweight, and 30 or greater is defined as obese." These two sentences are not considered paraphrased because there is some information missing. However, they will have a high similarity score, and they can be integrated into our scenario.

Chapter 3

Methodology

To integrate semantically similar questions from multiple questionnaires efficiently, we designed and developed a question classification module to subdivide all questions to coarse categories. This avoids element-wise comparison for the whole dataset, which will reduce the processing time significantly. Meanwhile, to answer the second research question, we built a short text semantic textual similarity module to measure semantic similarity between comparable question pairs. This module combined conventional NLP techniques including both semantics and syntactic features, and a Recurrent Convolutional Neural Network to finish the task.

3.1 Question Classification

Figure 3.1 shows the basic structure of the question classification module. This module contains three parts: a feature extractor integrated with the Stanford NLP parser, a maximum entropy classifier, and a triplet extractor. The maximum entropy classifier is used to classify question types, mainly SBARQ (clauses introduced by subordinating conjunction). The triplet extractor [46] is for further information extraction for SQ (Yes/No questions and subconstituents of SBARQ without wh-elements). The reason for both parts is that, unlike the most QC datasets, almost 80 percent of the questions in our questionnaire dataset are polar questions. Regardless of whether question categories are coarse or grained, our system does not further classify the questions. By taking another step and extracting triplets, subject-predicate-object, we can further narrow the question content, and reduce the number of comparisons.

3.1.1 Feature Construction

Each question is represented as a vector of features before being fed into the ME classifier. This section introduces four binary feature sets that are used in the model: the question word feature, the N-gram feature, the head word feature, and the WordNet semantic feature [29].

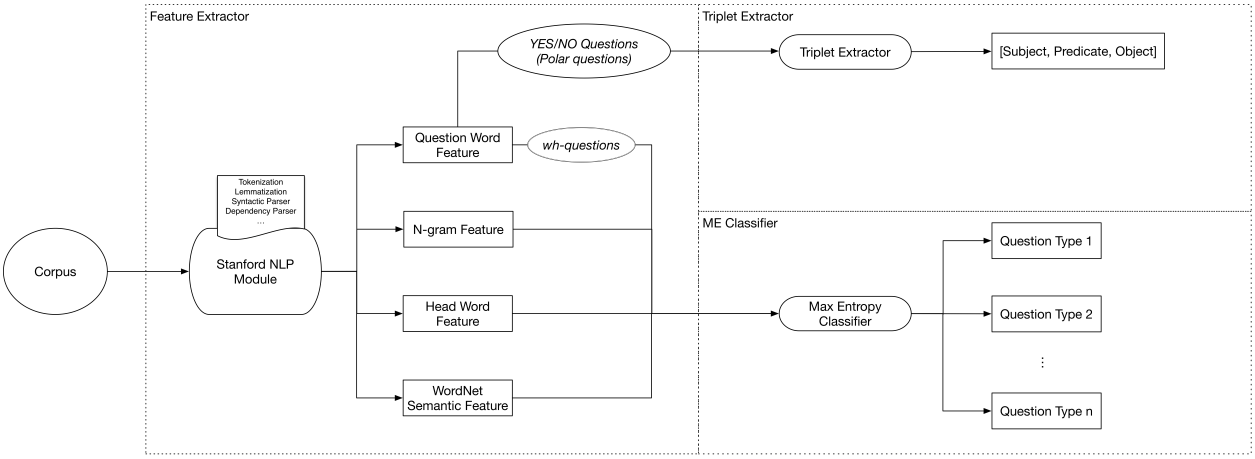


Figure 3.1: Structure of Question Classification Module

3.1.1.1 Question Word Feature

The question word feature is the question word that leads the question. For example, the question word of the question *What is the population of China?* is *what*. Therefore, we have what, which, when, who, how, why, be, and rest. *Rest* is the question type that does not belong to any previous type. For example, the question *Your weight?* is a rest-type question.

3.1.1.2 N-grams Feature

A N-gram is a sub-sequence of N words from a given sentence. For example, unigram is equivalent to the bag of words feature, and bigram forms the pairs of words feature, and so forth. We adopted unigram, bigram, and trigram features in our system. The reason to use these features is to provide word sense disambiguation for questions such as *How long do you sleep everyday?* and *How long is it from your home to work?* because *how long* could either refer to duration or distance. This feature can help us to clarify this ambiguity.

3.1.1.3 Head Word Feature

In linguistics, the head word of a phrase is the word that determines the syntactic category of that phrase. For example, the head of the noun phrase *boiling hot water* is the noun *water*. To obtain the head word feature, we need to apply a syntactic parser. A syntactic parser is a model

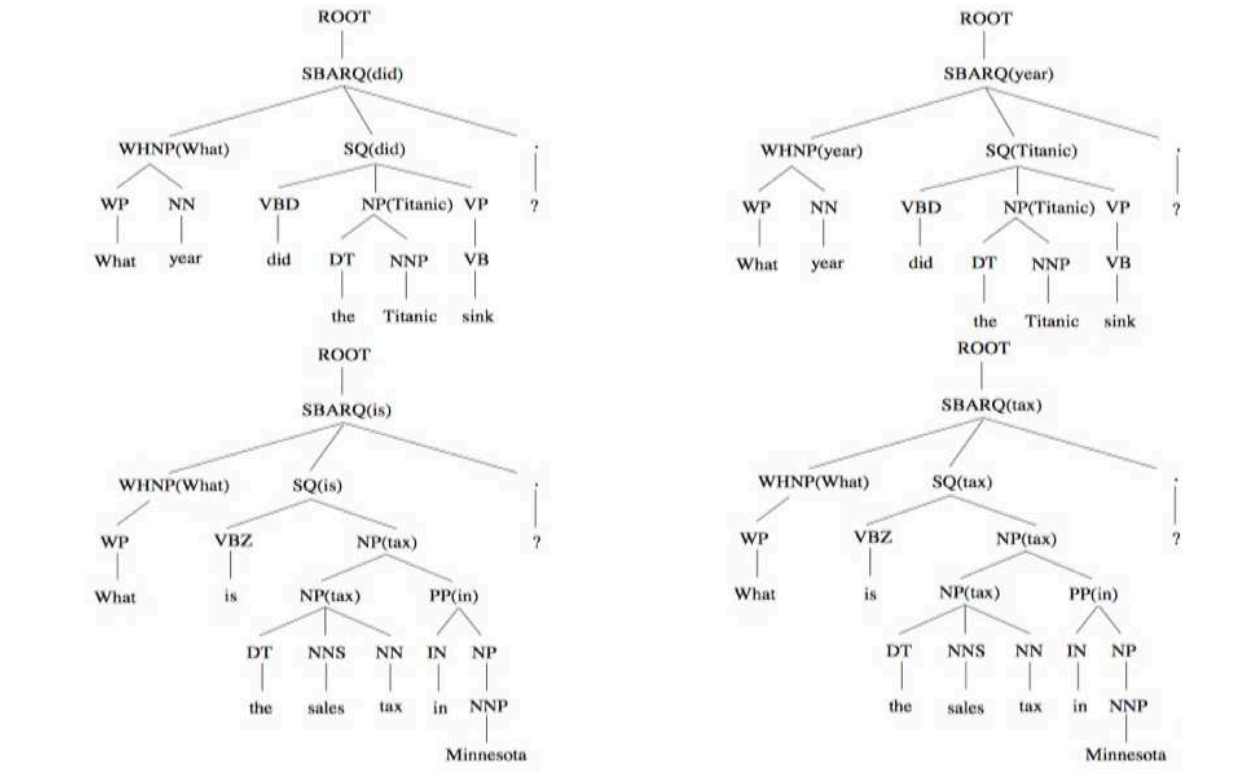


Figure 3.2: The difference between Klein and Manning's and Huang's head word definition

that outputs the grammatical structure of the given sentence. There are various state-of-art parsers available such as the OpenNLP parser [47], the Stanford Parser [48] and the Berkeley Parser [49]. We used the Stanford Parser in our system to identify the head word because the Stanford NLP library provides the most complete group of NLP tasks. For further analysis, we wanted to do tokenization, pos-tagging, syntactic parsing, dependency parsing, and other parsing tasks using only one query in order to reduce the cost of computation, so we chose the Stanford Parser.

There are various rules developed throughout the literature to guide semantic analysis [29, 50, 51]. In particular, the rules for finding the semantic head word of phrases including SBARQ, SQ, VP, and SINV, have been redefined such that there is a preference for using a noun or noun phrase rather than a verb or verb phrase for this task. The difference is shown inFigure 3.2. For example, in the question *"What year did the Titanic sink?"* The head word finder rules proposed by Klein and Manning [50] will extract the verb *did* as the head word. On the other hand, Huang's [29]

revised rules will extract the noun *year* as the head word. We found that the latter algorithm fit our situation better.

In the same paper, Huang also compiled a list of regular expressions to help question head word identification [29]. We adopted the question head word extraction algorithm proposed by Huang and revised it a bit to suit our requirements better.

Note that we only kept two regular patterns because the rest of Huangs papers appear frequently in QA system research, but they are rarely seen in the questionnaire area. There is no head word returned for when, where, or why-type questions, as these wh-words are informative enough. The reason for doing both is to reduce potential noisy information.

Revised question head word extraction Algorithm

Input: Question q
Output: Question head word

```
1.  if q.type == when|where|why then
2.      return null
3.  end if
4.  if q.type == how|which then
5.      return the word following word "how" | "which"
6.  end if
7.  if q.type == what && q.matches "DESC:reason pattern" then
8.      return "DESC:reason"
9.  end if
10. if q.type == who && q.matches "HUM:desc pattern" then
11.     return "HUM:desc"
12. String candidate = head word extracted from question parse tree
13. if candidate.tag starts with NN then
14.     return candidate
15. end if
16. return the first word whose tag starts with NN
```

DESC:reason pattern The question begins with what causes/cause
HUM:desc pattern The question begins with Who is/was and follows by a word starting with a capital letter

Figure 3.3: Revised question head word extraction algorithm

3.1.1.4 WordNet Semantic Feature

WordNet [52] is a large English lexicon where meaningfully related words are connected via cognitive synonyms. It is a useful tool for word semantic analysis and has been widely used in question classification. One of the most widely used type of information that is provided by WordNet is hypernyms: If A is a hypernym of B, then every A is a (kind of) B. In WordNet, words are organized into hierarchies with hypernym relationships; this process provides a natural way to augment hypernyms features from the original head word. For example, the question *What bread did you eat today?* requires knowing that baked goods are the hypernym of bread, and food is the hypernym of baked goods (bread→baked goods→food). We adopted Huang's first approach, which directly introduces hypernyms for the extracted head words.

The augment of hypernyms for given head word can be useful because it can introduce useful additional information, but on the other side, it can also bring some degree of noise if the hypernyms are not well identified. Three vital points should be taken into consideration during this process:

1) which part of speech senses should be augmented?

2) which sense of the given word should be augmented?

3) how long of the hierarchies is required to strike a balance between the generality and the specificity?

The first issue can be resolved by mapping the Penn Treebank pos tag of the given head word to its WordNet pos tag. The second problem is actually a word sense disambiguation (WSD) [53] problem. The Lesk algorithm [54] is a classical algorithm for resolving the WSD problem. It is based on the assumption that words in a given context are more likely to share a common topic. A basic implementation of this algorithm is described as follows:

a. Choosing pairs of ambiguous words within a context.

b. Checks their definitions in a dictionary, i.e. WordNet.

c. Chooses the senses so that to maximize the number of common terms in the definitions of the chosen words.

Here, the context words are words in the question other than the head word, and the dictionary is the gloss of a sense for a given word. The algorithm in Figure 3.4 shows the adapted Lesk algorithm:

Lesk algorithm for head word sense disambiguation

Input: Question q and its head word h
Output: Disambiguated sense for h

```
1.  int count = 0
2.  int maxCount = -1
3.  sense optimum = null
4.  for each sense s for h do
5.      count = 0
6.      for each context word w in q do
7.          int subMax = maximum number of common words in s def
    inition (gloss) and definition of any sense of w
8.          count = count + sumMax
9.      end for
10.     if count > maxCount then
11.         maxCount = count
12.         optimum = s
13.     end if
14. end for
15. return optimum
```

Figure 3.4: Lesk Algorithm for head word sense disambiguation

In detail, for each sense of given head word, this algorithm computes the maximum number of common words between gloss of this sense and the gloss of any senses of the context words. Among all head word senses, the sense that results in the maximum number of common words is chosen as the optimal sense for augmenting hypernyms.

Finally, we addressed the third problem in a heuristic way based on the experiments. In the experiments described in Chapter 4, we selected a subset of training data and ran the algorithm for $depth = 1, 2, 3, 4, 5, 6$. Considering the tradeoff between accuracy and time cost, 3 was chosen for the depth of the hierarchy length to detect common sense.

3.1.1.5 Triplet extraction for polar questions

For this study, the triplet was the $[subject, predicate, object]$ set extracted from a sentence. The reason we chose these three elements as the index of a sentence is that they deliver the core information of a sentence, and our target dataset consists of questionnaires. A questionnaire dataset contains more polar questions (Yes/No questions) than a QA dataset. The purpose of the Question Classification module is to reduce the number of comparisons between two datasets, so we need to extract information to further subdivide these questions if the majority of the dataset are polar questions.

Rusu [46] presented an approach to extracting subject-predicate-object triplets from a given sentence. He proposed an algorithm that is simply based on English grammar and syntactic structure. There are two reasons that this algorithm fits our needs:

1) The output subject-predicate-object triplet is informative enough for further deciding whether two polar questions have the same meaning.

2) The algorithm just analyzes the structure of syntactic parse tree of the given sentence. It does not need any training or learning process, which will substantially reduce time cost.

We applied a revised triplet-extraction algorithm based on the syntactic parse tree produced by the Stanford NLP parser and revised it according to our specific needs as shown in Figure 3.5.

The modification we made is that we changed the polar questions to declarative sentences first. Then, we considered that not every sentence has an object. For example, we cannot extract an object from the question "Did you exercise everyday?" Therefore, instead of returning a failure, we return an empty string for the object-extraction function.

3.1.2 Maximum Entropy Classifier

We decided to select Maximum entropy models, also known as log-linear and exponential learning models, which provide a general-purpose machine learning technique for classification and prediction. This technique has been applied successfully to natural language processing including part-of-speech tagging and named entity recognition.

Revised Triplet-Extraction Algorithm

```
1.  function TRIPLET-EXTRACTION(question):
2.      if wh-words of question == Did/Do/Does:
3.          remove Did/Do/Does from the question
4.      else if wh-words of question == Be:
5.          exchange the wh-word with the word after it
6.      result = EXTRACT-SUBJECT(NP_subtree) ∪ EXTRACT-
    PREDICATE(VP_subtree) ∪ EXTRACT-OBJECT(VP_siblings)
7.      if result ≠ failure then
8.          return result
9.      else
10.         return failure
11.
12. function EXTRACT-ATTRIBUTES(word):
13.     // search among the word's siblings
14.     if word is adjective
15.         result = all RB siblings
16.     else if word is noun
17.         result = all DT, PRP$, POS, JJ, CD, ADJP, QP, NP sibling
    s
18.     else if word is verb
19.         result = all ADVP siblings
20.     // search among the word's uncles
21.     if word is noun or word is adjective
22.         if uncle = PP
23.             result = uncle subtree
24.     else if word is verb and uncle is verb
25.         result = uncle subtree
26.     if result ≠ failure then
27.         return result
28.     else
29.         return failure
30.
31. function EXTRACT-SUBJECT(NP_subtree):
32.     subject = first noun found in NP_subtree
33.     subjectAttributes = EXTRACT-ATTRIBUTES(subject)
34.     result = subject ∪ subjectAttributes
35.     if result ≠ failure then
36.         return result
37.     else
38.         return failure
39.
40. function EXTRACT-PREDICATE(VP_subtree):
41.     predicate = deepest verb found in VP_subtree
42.     predicateAttributes = EXTRACT-ATTRIBUTES(predicate)
43.     result = predicate ∪ predicateAttributes
44.     if result ≠ failure then
45.         return result
46.     else
47.         return failure
48.
49. function EXTRACT-OBJECT(VP_subtree):
50.     siblings = find NP, PP and ADJP siblings of VP_subtree
51.     for each value in siblings do
52.         if value = NP or PP
53.             object = first noun in value
```

Figure 3.5: Revised triplet extraction algorithm

For this study, we adopted the NLTK implementation in our system, which can integrate features from many heterogeneous information sources for classification. Each feature corresponds to a constraint within a model. The following section introduces the principle of the ME classifier.

The principle of the maximum entropy classifier, which is the basis of the maximum entropy model, states that the probability distribution which best represents the current state of knowledge is the one with the largest entropy.

Suppose $P(x)$ is the probability density function of discrete random variable x, then the entropy of $P(x)$ is:

$$H(x) = -\sum_{i=1}^{n} p_i \log p_i. \tag{3.1}$$

Assuming that the classification model is a conditional probability distribution$P(y|x), x \in X \subseteq \mathbb{R}^n$, represents the input, $y \in Y$ represents the output, X, Y is the input set and output set, respectively. The purpose of this model is to, for the given input x, output y according to the conditional probability $P(y|x)$.

For a given training dataset,

$$T = (x_1, y_1), (x_2, y_2), ..., (x_N, y_N) \tag{3.2}$$

The purpose of training is to select the best classification model based on the maximum entropy principle. For the given dataset, we can obtain the empirical distribution of joint distribution and marginal distributions. We use the feature function $f_i(x, y)$ to describe a fact that occurs between x and y:.

$$f(x, y) = \begin{cases} 1, & \text{x and y satisfy some fact} \\ 0, & \text{otherwise} \end{cases} \tag{3.3}$$

The expected value of feature function $f(x, y)$ on the model $P(y|x)$ and the empirical distribution $\tilde{P}(x, y)$, denoted as $E_{\tilde{p}}(f)$.

Therefore, if $x \in \{x_1, x_2, ..., x_n\}, y \in \{y_1, y_2, ..., y_m\}$ is discrete random variable, given X, the conditional entropy of Y can be defined as:

$$H(y\,|x) = \sum_{i=1}^{n} p(x_i) H(y\,|x = x_i) = -\sum_{i=1}^{n} p(x_i) \sum_{j=1}^{m} p(y_j\,|x_i) \log p(y_j\,|x_i). \tag{3.4}$$

Based on knowing the above, the maximum entropy model is the model within model set C that satisfies all constraints:

$$C \equiv \{P \in \mathrm{P}\,|E_p(f_i) = E_{\bar{p}}(f_i), i = 1, 2, ..., n\}\,. \tag{3.5}$$

The model with the maximum conditional entropy $H(P)$ is then called the maximum entropy model.

3.2 Short-Text Semantic Textual Similarity

After we have done the question classification, we can pass the questions that share the same category to the short-text semantic textual similarity module. Figure 3.6 shows the structure of this module. It contains two sub-modules:

Conventional NLP Module extracts NLP features using some traditional NLP techniques. For the features that are independent of each other, like the Bag-of-Words feature, we first represented each sentence with these features and then adopted the kernel-based method to calculate the similarity of a pair of sentences. On the other hand, features that are calculated from two sentences, such as the N-gram overlap feature, can be simply calculated from directly. Both types of features together are poured into regression algorithms to make predictions.

Deep Learning Module encodes input sentence pairs into distributed vector representations. There are multiple widely used trained vectors like Word2Vec [55] and GloVe [56], which we used to train the end-to-end Recurrent Convolutional Neural Networks to obtain similarity scores.

The final similarity score generated by this process is the average of the above two sub-modules scores. In the next section, we will describe the system in detail.

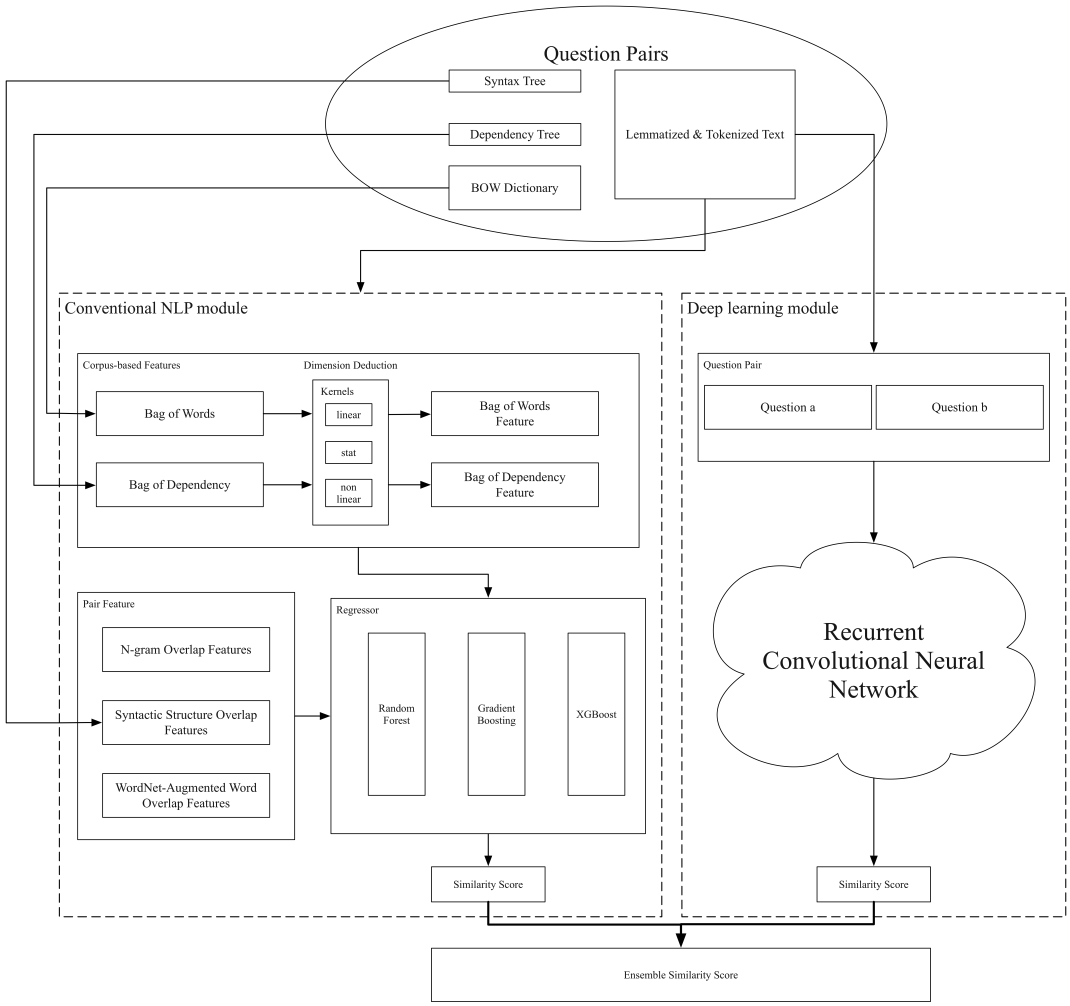


Figure 3.6: Structure of Semantic Textual Similarity

3.2.1 Conventional NLP Module

3.2.1.1 Feature weighting

In the process of feature construction, prior research showed that reweighting the counts of some distributional features will improve the paraphrase detection.

TF-IDF is a numerical statistic that is intended to reect how important a word is to a document in a corpus. The TF-IDF value is the product of two statistics: term frequency and inverse document frequency. This combination makes the TF-IDF value increase proportionally to the number of times a word appears in the document, and it is offset by the frequency the word in the

corpus. There are various schemes for calculating these two statistics, noting that the document in the dataset is a single question. Therefore, we used Boolean frequencies as the term frequency as follows: $tf(t,d) = 1$ if t occurs in d, and $idf(t,D) = log(1 + \frac{N}{n_t})$, where N stands for the total number of documents (questions) in the corpus, and n_t stands for the number of documents where the term t appears. At last, the TF-IDF value can be represented as:

$$TF\text{-}IDF(t,d,D) = tf(t,d) \cdot idf(t,D). \tag{3.6}$$

3.2.1.2 Corpus-based Features

The corpus-based features are the features that are related to the whole corpus. They are bag-of-word features and bag-of-dependency features. After calculating these features, we will apply several kernels to the sentence pair to reduce the dimensionality of the features.

Bag of Words

Each question is represented as the bag of its words, disregarding grammar and word order, but keeping multiplicity. Also, we weighted each word by its TF-IDF value.

Bag of Dependency

For each sentence, we interpreted its dependency tree as a set of triples: $[governor, dependency-label, subordinate]$. This feature is similar to the BOW feature in that we treat triples as words and represent each sentence as a bag of dependency-triples.

Dimensionality Reduction

The dimensionality of the features constructed above from BOW and BOD is huge (approximately more than 70K features), and the high dimensionality suppresses the discriminating power of other features. In the latter part of the paper, we will see that the dimensionality of features constructed from a sentence pair (less than 100) and the deep learning network (less than 1K) is much smaller in our system. In order to reduce the high dimensionality of corpus-based features,

Table 3.1: List of 10 kernel functions

Type	Measures
Linear kernel	Cosine distance, Manhattan distance, Euclidean distance, Chebyshev distance
Stat kernel	Pearson coefficient, Spearman coefficient
Non-linear kernel	polynomial, rbf, laplacian, sigmoid

we used 10 kernel functions to calculate sentence pair similarities. Table 3.1 lists all the kernel functions we used in this module. In total, we collected 20 corpus-based features after dimension deduction.

3.2.1.3 Pair Features

Three types of sentence pair matching features are designed to directly calculate the similarity between two questions based on the N-gram overlap, syntactic structure overlap, and WordNet-Augmented word overlap.

N-gram Overlap Features

Let S_1 and S_2 be the sets of consecutive N-grams in the first and the second questions respectively. The N-gram overlap feature is defined as follows [57]:

$$ngo(S_1, S_2) = 2 \times \left(\frac{|S_1|}{|S_1 \cap S_2|} + \frac{|S_2|}{|S_1 \cap S_2|} \right)^{-1} \tag{3.7}$$

We obtained N-grams at the lemmatized word level. We applied $n = [1, 2, 3]$ and collected 3 features.

WordNet-Augmented Word Overlap

The N-gram overlap feature will output a high similarity value only if exactly the same words (or lemmas) appear in both questions. To allow for some lexical variation, we used WordNet to assign partial scores to words that are not common to both questions. We used the definition provided by $\check{S}arić$ [57]:

$$P_W N(S_1, S_2) = \frac{1}{|S_2|} \sum_{w_1 \in S_1} score(w_1, S_2) \tag{3.8}$$

$$score(w, S) = \begin{cases} 1 & \text{if } w \in S \\ \max\limits_{w' \in S} sim\,(w, w') & \text{otherwise} \end{cases} \tag{3.9}$$

where $sim(\cdot, \cdot)$ represents the WordNet path length similarity. The overall feature is defined as the harmonic mean of $P_{WN}\,(S_1, S_2)$ and $P_{WN}\,(S_2, S_1)$.

Syntactic Structure Overlap Features

N-gram, BOW, and other features discussed above are purely lexicon-based approaches, which are often inadequate for performing more complex tasks involving the use of more varying syntactic structures. In order to use more structural or syntactical information and capture higher order dependencies between grammar rules, we adopted Wangs [58] syntactic tree matching algorithm, which originated from Collinss [59] approach.

According to Zhangs [32] definition, the tree fragments of a syntactic tree are all of its subtrees that include at least one terminal word or one production rule, with the restriction that no production rules can be broken into incomplete parts. Wang proposed the following weighting schemes for the tree fragments:

Preliminary 1: The weighting factor δ_i denotes the importance of node i in the parsing tree. Its value differs for different types of nodes:

- $\delta_i = 1.2$, where node i is either the POS tag VB or NN.
- $\delta_i = 1.1$, where node i is either the POS tag VP or NP.
- $\delta_i = 1$, for all other types of nodes.

This preliminary came from the intuition that nouns and verbs are considered to be more important than other types of terms.

Preliminary 2: The weighting coefficient θ_k for tree fragment k conveys the importance of the tree fragment, whose value is the production of all weighting factors of node i that belong to the tree fragment k, i.e. $\theta_k = \prod_{i \in fragment\ k} \delta_i$

This preliminary means that the more important nodes a tree fragment contains, the more important this tree fragment is.

The next two preliminaries define the size of sub-tree S_i and its weighting factor λ, together with the depth of the sub-tree D_i and its weighting factor μ as follows:

Preliminary 3: The size of the tree fragment S_i is defined by the number of nodes that it contains. The size of the weighting factor λ is a tuning parameter indicating the importance of the size factor.

Preliminary 4: The depth of the tree fragment D_i is defined as the level of the tree fragment root in the entire syntactic parsing tree, with D_{root}. The depth weighting factor μ is a tuning parameter indicating the importance of the depth factor.

Given the parameters listed above, Wang gave the following weighting scheme for the tree fragment:

Definition 1: The weight of a tree fragment w_i is defined as $\theta_i \lambda^{Si} \mu^{Di}$, where θ_i is its weighting coefficient, S_i is the size of the sub-tree, λ is the size weighting factor, D_i is the depth of the sub-tree, and μ is the depth weighting factor.

Based on the weighting scheme of tree fragments above, Wang proposed an algorithm to calculate the weight of matching tree fragments along with similarity metrics.

Preliminary 5: If two tree fragments TF_1 and TF_2 are identical, the weight of their resulting matching tree fragment TF is defined to be:

$$w(TF) = w(TF_1)w(TF_2) \tag{3.10}$$

From here, we can calculate the overall matching score between two nodes r_1 and r_2 to be the multiplication of the weights of all matched tree fragments under the roots of r_1 and r_2:

$$M(r_1, r_2) = \begin{cases} 0 & \text{if } r_1 \neq r_2 \\ \prod_{i=1}^{\eta} w\left(TF_i(r_1, r_2)\right) & \text{otherwise} \end{cases} \tag{3.11}$$

where $r_1 \neq r_2$ stands for the fact that either labels or production rules for r_1 and r_2 are different. $TF_i(r_1, r_2)$ is the i-th matching tree fragment under r_1 and r_2, and η is the total number of tree fragments.

After calculating the node matching score between two nodes, we are able to find the similarity score between the two syntactic parsing trees T_1 and T_2. By traversing the parsing trees in post-order and calculating the pair-wise node matching scores, we can get a $|T_1| \times |T_2|$ matrix of $M(r_1, r_2)$. The summation of all scores is used to represent the similarity score between two parsing trees as follows:

$$sim(T_1, T_2) = \sum_{r_1 \in T_1} \sum_{r_2 \in T_2} M(r_1, r_2) \tag{3.12}$$

and the normalized similarity score would be

$$sim2(T_1, T_2) = \frac{sim(T_1, T_2)}{\sqrt{sim(T_1, T_1) sim(T_2, T_2)}} \tag{3.13}$$

By applying a dynamic programming technique, we can calculate the final similarity score between two parsing trees in polynomial time.

3.2.1.4 Regression Models

The modules above generated 14 features altogether. Next, we explore four learning algorithms for regression: Random Forests (RF), Support Vector Machine (SVM), Gradient Boosting (GB), and XGBoost (XGB). The first three algorithms are available in the Scikit-Learn [60] library and XGB [61] is open source and accessible on Github.

3.2.2 Recurrent Convolutional Neural Network

The conventional approach to measuring similarity between texts as described above is limited because some of processes lose the order information and some of them ignore the context. Consider, for example, the following sentence:

A sunset stroll along the South Bank affords an array of stunning vantage points.

To analyze the word *Bank* in the sentence, we cannot identify its correct meaning if we isolate this word. A bank can be either a financial institution or sloping land. Therefore, we need to include more context to achieve disambiguation. If we see one word ahead and get the South Bank bi-gram, we can see that both words are capitalized. People who are unfamiliar with London may think this is the name of a bank. However, by analyzing enough contextstrolling along the South Bankwe can ensure that it means the name of a location, and it has nothing to do with the bank.

Recently, pretrained word vector and deep learning models have introduced new approaches to NLP tasks. Socher [44, 62, 63] proposed a process for building Recursive Neural Networks. This approach has proved to be effective for semantic construction at the sentence level. However, building a recursive neural network requires a tree structure to process semantic construction, and the quality of the network depends strongly on the accuracy of the tree. Moreover, to construct the textual tree requires at least$O(n^2)$ time complexity, where n stands for the length of the sentence. Finally, when representing documents, the relationship between two sentences does not always form a tree structure, which makes semantic construction difficult.

A Recurrent Neural Network (RNN) can finish semantic construction in $O(n)$ time. This model processes the whole document word by word, and it saves all context information to a fix-sized hidden layer. The advantage of an RNN is that it can capture the context information and process on the long-distance context better than conventional approaches. However, for a forwarding RNN, for instance, the posterior words will have more importance than the anterior ones. Therefore, the RNN will consider more of the information from the latter part when building semantics for the whole document. However, because not all documents will emphasize the latter part, the algorithm of the RNN may affect the accuracy of semantic representation.

To deal with the problem encountered by RNNs,, Collobert [64] proposed a Convolutional Neural Network (CNN) to build the semantic representation. By using a max-pooling technique, a CNN can find the most useful textual section, and the time complexity is also $O(n)$. Therefore, a CNN usually performs better on semantic representation. However, the current CNN model usually applies a relatively simple convolutional kernel, like a fixed input window [64, 65]. When using these models, the method of determining the window size is crucial. When the window is too narrow, the context information may be insufficient. While when the window size is too large, it will lead to the increase of the parameters, which increases the difficulty of model optimization.

To resolve the defects above, Lai [66] proposed a Recurrent Convolutional Neural Network (RCNN). This approach first applies a bi-directional recurrent structure, which may introduce considerably less noise compared to a traditional window-based neural network, to capture the contextual information to the greatest extent possible when learning word representations of texts. The next step employs a max-pooling layer that automatically decides which features play key roles in text classification to capture the key component in the texts. By combining these two features, an RCNN has the advantages of both an RNN and a CNN, so an RCNN depicts context information better and provides an unbiased representation of the whole document. Moreover, the RCNN model shows a time complexity of $O(n)$, which is linearly correlated with the length of the text length.

3.2.2.1 Background: Deep Learning Models

Recursive Neural Network

The structure of the Recursive Neural Network model is shown inFigure 3.7.The concept follows a tree structure, summarizing the word semantic representation to phrases and finally achieving the whole sentences semantic representation.

The Recursive Neural Network usually uses a binomial tree, in some cases (like the dependency parse tree [67]) a multinomial tree is used. Now we would like to introduce the Recursive Neural

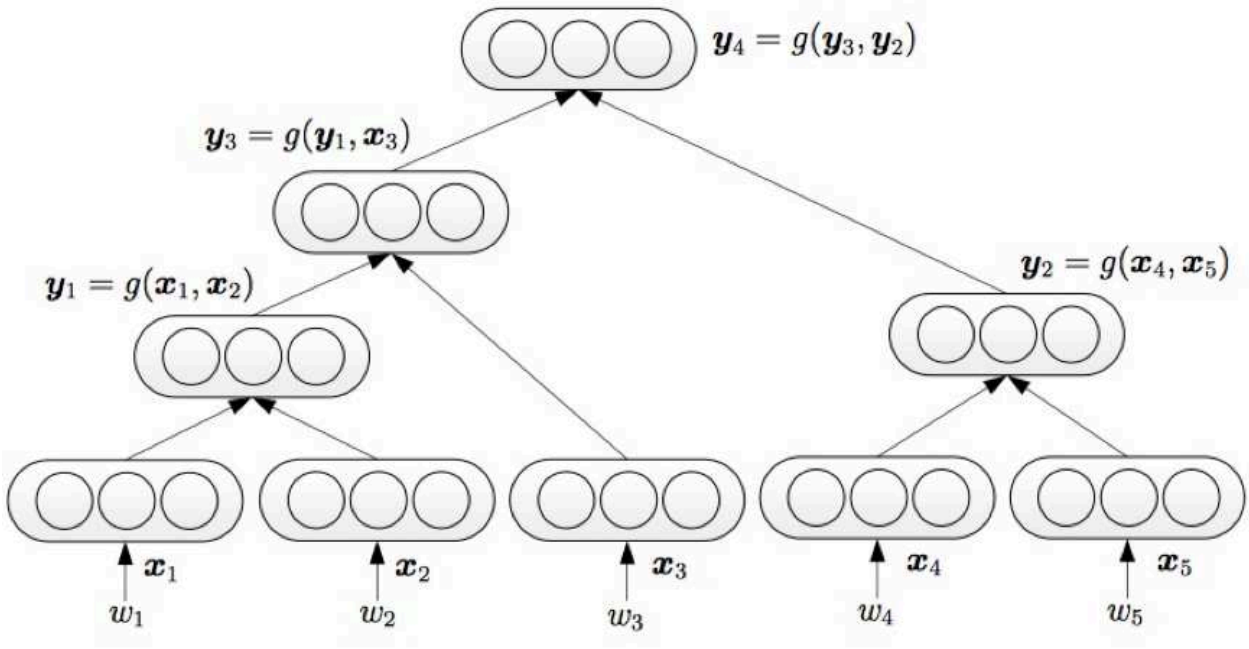


Figure 3.7: The Structure of Recursive Neural Network

Network by demonstrating the methods for constructing the tree structure and the composition function $y = f(a, b)$ from the child node to the parent node.

There are two common ways to build the tree: 1) Use a parser to build a syntax tree [44, 63]or 2) use a greedy algorithm to rebuild the neighbor child subtree that has the smallest error [62]. Using a semantic parser will ensure that the tree structure is a syntax tree. Each leaf in the tree will respond to a word in the sentence. The node after composition will also represent the phrases in the sentence. The second approach, which is also unsupervised, can automatically find the pattern in the data, but it cannot ensure that each node in the tree has an actual syntactic meaning.

There are generally three types of composition function $y = f(a, b)$ from child node to parent node:

a. Syntax-based

Child node is represented as vector a, b, and parent node can be calculated thusly:

$$y = \phi(H[a; b]) \tag{3.14}$$

where ϕ stands for non-linear activation function, and weighting matrix H can either be fixed [68], or varied based on the different syntax structure. This method is often used in syntax analysis.

b. Matrix-based

In vector-based method, each node is represented by two parts: a matrix and a vector. For $[A, a]$ and $[B, b]$ child nodes, the composition function is as follows:

$$y = \phi(H[Ba, Ab]) \tag{3.15}$$

$$Y = W_M \begin{bmatrix} A \\ B \end{bmatrix} \tag{3.16}$$

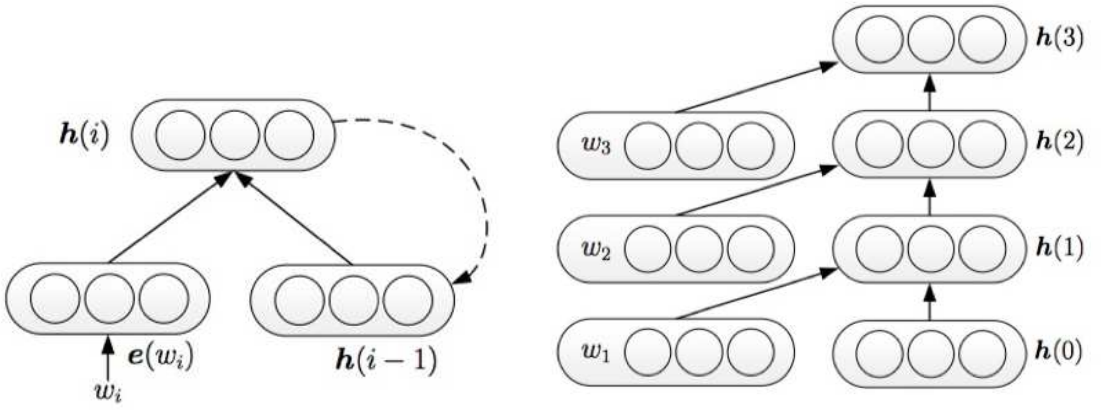
where $W_M \in \mathbb{R}^{|a| \times 2|a|}$, which ensures that the semantic transformation matrix corresponding to the parent node $Y \in \mathbb{R}^{|a| \times |a|}$ has the same dimensionality as the matrix A, B. Each word has a semantic transformation matrix in this method. For the words that affect other parts of the sentence, like the negation words, the normal syntax-based method cannot accurately depict the relations. The matrix-based method can resolve this problem.

Recurrent Neural Network

The Recurrent Neural Network (RNN) model was first proposed by Elman in 1990 [69]. The idea is to recurrently input each word in the document while building a hidden layer that keeps all of the context information.

The RNN model is a special case of a Recursive Neural Network. It can be seen as a tree where the left child of each non-leaf node is a leaf node. This special structure produces two important characteristics. First, since the network structure is fixed, the model only needs $O(n)$ time to build the semantics, which is much more efficient than the Recursive Neural Network. Second, the RNN structure is very deep. The depth of the network is equivalent to the number of words

in the sentence. Therefore, the traditional training method does not work on RNN because of the vanishing or exploding gradient problem, which needs to be resolved by special optimization techniques.



(a) Recurrent Structure **(b)** Expanded Recurrent Structure

Figure 3.8: Structure of Recurrent Neural Network

The semantic construction process of the RNN model is similar to that of the Recursive Neural Network model. Each word and all of the hidden layers representing its left-side context together form the new hidden layer (structure is shown in Figure 3.8, and equation is shown in Eq.3.17). The process moves from the first word of the sentence to the last, and the hidden layer of the last word represents the whole text semantics.

$$h(i) = \phi\left(H[e(w_i); h(i-1)]\right) \tag{3.17}$$

Regarding optimization techniques, there are differences between the RNN and other neural networks. For normal neural networks, a back-propagation algorithm can be implemented easily with the help of the chain rules of derivatives. However, in the RNN, the weighing matrix H is reused, so directly differentiating the matrix is difficult. One naïve method is Back-propagation Though Time (BPTT). In this method, we first expand the network to the format as shown in Figure 3.8(b). For each level, the model uses the normal BP technique to update each hidden layer and

repeatedly update the weighing matrix H. There are several ways to deal with the vanishing gradient problem. The most straightforward method is that when using BPTT to optimize the network, only propagate for a fixed-sized length (5 levels, for example). Hochreither and Schmidhuber[43] proposed the Long Short-Term Memory (LSTM) model in 1997. This model introduces a memory cell, which can save long distance information, and it is a widely-used optimization scheme.

However, regardless of how the model is optimized, the semantics in the RNN are more likely to lean to the latter part. Therefore, the RNN model is rarely applied to represent the whole documents semantics. Because it can effectively represent the context information, this model is more commonly seen in the sequence labeling task.

Convolutional Neural Network

The Convolutional Neural Network (CNN) approach was first proposed by Fukushima in 1982 [70]. Then, LeCun added an important improvement in 1998 [71]. The CNN model is widely used in natural language processing. Collobert first applied it to a semantic labeling task [64].]. Kalchbrenner and Kim presented work using a CNN for text classification [72, 73].

The structure of CNN is shown in Figure 3.9. The core concept is local connection and weight sharing. In a normal feedforward neural network, each node in the hidden layer has full connection with all of the nodes in the input layer. While in the CNN, each node in the hidden layer only has connections to a fixed-sized area in the hidden layer. The size is denoted as wind (stands for window). For instance, the structure in the figure has $wind = 3$. It can be formulated as follows:

$$x_i = \left[e(w_{i-\lfloor wind/2 \rfloor}; \cdots; e(w_i); \cdots; e(w_{i+\lfloor wind/2 \rfloor})\right] \tag{3.18}$$

$$h_i^{(1)} = tanh(Wx_i + b) \tag{3.19}$$

After building several hidden layers, the CNN usually applies a pooling technique to compress the hidden layers with various sizes to a fixed-size hidden layer. Commonly used techniques are average-pooling and max-pooling. The max-pooling formula is

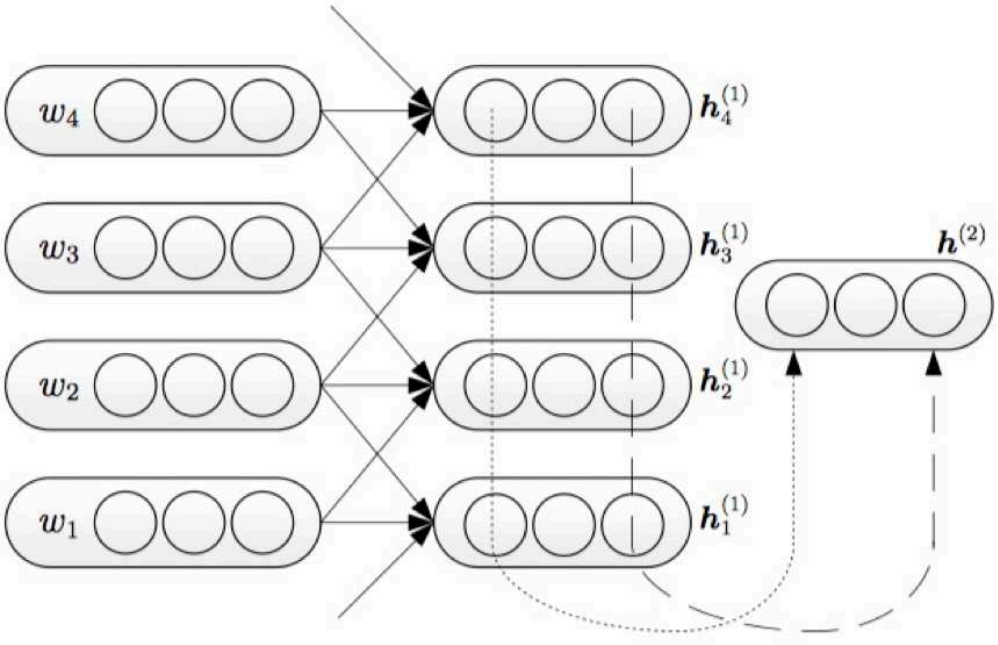


Figure 3.9: The Structure of Convolutional Neural Network

$$h^{(2)} = \max_{i=1}^{n} h_i^{(i)} \tag{3.20}$$

By using a convolutional kernel, a CNN can model different parts of a sentence and achieve the full semantics from all local nodes with the help of the pooling layer. Also, the overall time complexity is only $O(n)$.

3.2.2.2 Applying Recurrent Convolutional Neural Network

There are some attempts to combine the Recurrent Neural Network and the Convolutional Neural Network called the Recurrent Convolutional Neural Network. This model can have the dual advantages of the RNNs ability to consider long enough context and the CNN models unbiased nature and easy training. Siwei Lai [66] proposed an RCNN model to build document semantics. The figure below shows the network structure he proposed. The input of the network is document D, which consists of a word sequence $w_1, w_2, \cdots, w_n$. The number of the output node is 2, which

corresponds to whether two questions are identical or not. We use $P(I|Q_a, Q_b, \theta)$ to represent whether question a and question b are identical, and θ is the parameters of the network.

Lai combined the word and its context to represent the word itself. The context can help with disambiguation to achieve more accurate semantics. This process uses a bi-directional recurrent structure. We modified it for the purpose of calculating sentence pair similarity.

We define that $c_l(w_l)$ is the left-side context semantic representation for the word w_i, $c_r(w_i)$ is the right-side context representation for the word w_i. Both $c_l(w_i)$ and $c_r(w_i)$ are dense real vectors. The dimensionality is $|c|$. The formula for $c_l(w_i)$ is shown below:

$$c_l(w_i) = \phi\left(W^{(l)}c_l(w_{i-1}) + W^{(sl)}e(w_{i-1}))\right) \tag{3.21}$$

Here $e(w_{i-1})$ is the word vector for the word w_{i-1}.The word vector is also a real vector with low dimensionality $|e|$. All left-side context $c_l(w_1)$ for the first word w_1 is shared between sentences. $W^{(l)}$ is a matrix that transforms the hidden layer from the previous words left-side context into the current one. $W^{(sl)}$ is a matrix that is used to combine the previous word vector with the current word vector. ϕ is a non-linear activation function. The formula for right-side context representation is similar:

$$c_r(w_i) = \phi\left(W^{(r)}c_r(w_{i+1}) + W^{(sr)}e(w_{(i+1)})\right) \tag{3.22}$$

The left-side and right-side context vectors, respectively, can capture the semantic information. After getting the context information for the word w_i. We can define the word representation x_i to be the concatenation of the word s left-side context vector $c_l(w_i)$, the word w_is word vector $e(w_i)$, and the word w_is right-side context vector $c_r(w_i)$:

$$x_i = [c_l(w_i); e(w_i); c_r(w_i)] \tag{3.23}$$

The recurrent structure helps us to acquire all left-side context representations c_l with only one forward scanning. Similarly, with one backward scanning, we can acquire all right-side context

representations c_r. Therefore, the time complexity of the whole process is $O(n)$. Meanwhile, this recurrent structure will perform better than a CNN because it contains more context information than the fixed-size window approach used in CNN.

After acquiring x_i for the word w_i, Lai applied a linear transformation but instead of the tanh activation function, we applied ReLU (Rectified Linear Units) as our activation function:

$$y_i = ReLU\left(W^{(2)}x_i + b^{(2)}\right) \tag{3.24}$$

$$ReLU(x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{if } x < 0 \end{cases} \tag{3.25}$$

where y_i is a latent semantic vector, in which each semantic vector will be analyzed to determine the most important factor for representing the text by the convolutional neural network. When all of the representations of words are calculated, a max-pooling layer is applied:

$$y^{(2)} = \max_{i=1}^{n} y_i \tag{3.26}$$

The max function is an element-size function, which means the *k*-th element of $y^{(2)}$ is the maximum in the *k*-th elements of . The reason to adopt a max-pooling layer is that we thought for the textual information retrieval, the key information usually relies on a few words or phrases and their combination. The max-pooling layer attempts to find the most important latent semantic factors in the document. Moreover, the time complexity of the pooling layer is also $O(n)$, which makes the overall model have the time complexity $O(n)$.

Figure 3.10 shows the structure of the network. The input to the network is a pair consisting of sentences a and b. After being processed by the convolutional layer and max-pooling layer, two sentences with varying lengths are converted into fixed-length vectors. With the pooling layer, we can make a comparison between sentence pairs:

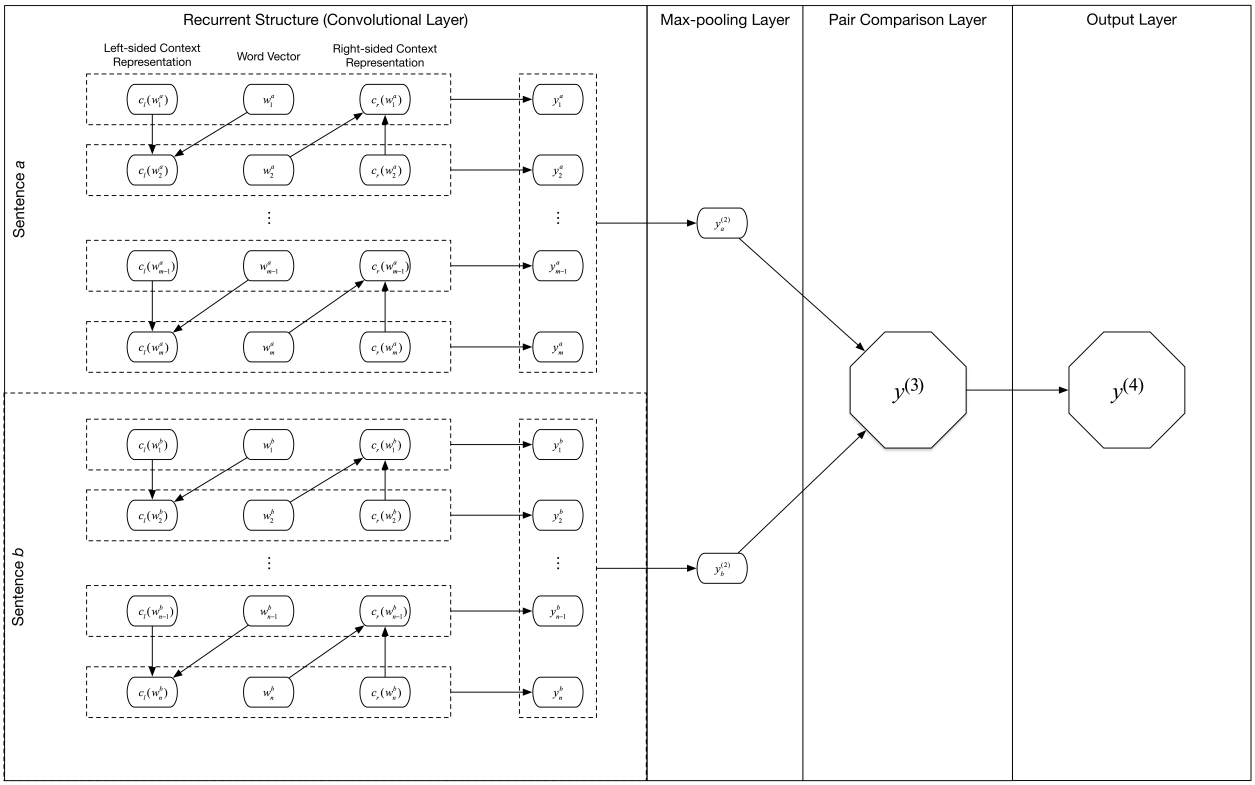


Figure 3.10: The Structure of Recurrent Convolutional Neural Network

$$y^{(3)} = |y_a^{(2)} - y_b^{(2)}| \tag{3.27}$$

The last part of our model is an output layer. Similar to traditional neural networks, it is defined as follows:

$$y^{(4)} = W^{(4)} y^{(3)} + b^{(4)} \tag{3.28}$$

At last, we applied the *softmax* function to $y^{(4)}$, which converts the output numbers into probabilities:

$$P\left(I|Q_a, Q_b, \theta\right) = \frac{exp\left(y_i^{(4)}\right)}{\sum_{k=1}^{n} exp\left(y_k^{(4)}\right)} \tag{3.29}$$

3.2.2.3 Model Training

The training target is to maximize the following likelihood, where $\mathbb{Q}$ is the training questions pairs set, and stands for whether the questions are identical or not:

$$\theta \mapsto \sum_{Q \in \mathbb{Q}} \log P(I|Q, \theta) \tag{3.30}$$

The model applies the stochastic gradient descent method [74] to optimize the training process above. For each iteration, the model randomly select a sample (Q, I), process a gradient iteration according to formula below, where α is the learning rate.

$$\theta \mapsto \theta + \alpha \frac{\partial \log P\left(I|Q, \theta\right))}{\partial \theta} \tag{3.31}$$

Additionally, we use a trick proposed by Plaut and Hinton [75] that is widely used when training neural networks. The model will initialize all of the parameters in the neural network from a uniform distribution. The magnitude of the maximum or minimum equals the square root of the node number from the previous layer.

Chapter 4

System Architecture and Evaluation

The overall structure of the whole system is shown in Figure 4.1. We used a Python wrapper for Stanford CoreNLP, py-corenlp [76] to communicate with the Stanford CoreNLP Server. For each sentence, the Stanford CoreNLP Server returns the lemmatized and tokenized texts as well as the syntax parse tree and dependency parse tree. The Question Classification Module classifies wh-questions into different question types and polar questions into different triplet sets. We used PySpark, a python implementation of Spark, to send questions in the same group to the same node. Then within each node, we combined question pairs, sent those pairs to the semantic similarity module, and calculated the final predicted similarity score. At last, we used 10 machines to build the Spark cluster. Each machine has 8×2.6G CPU and 32GB memory.

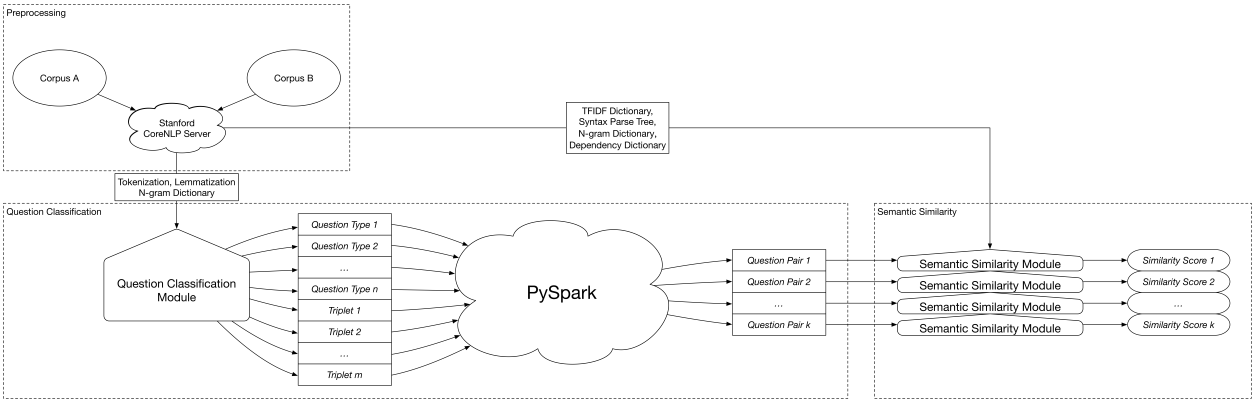


Figure 4.1: Overall Structure of the System

The main problem is that our task is a real-world challenge and there is no golden answer for the identification result. To resolve this problem, we decided to use existing datasets from other sources to train and evaluate the system, and then apply the NLSY and Add Health dataset to it to see the outcome. No matter how the second experiment performs, the first one will, to some extent, show the quality of the system.

4.1 Question Classification Module

We put the structure of the question classification module again here just for reference. The following two sections show how we setup the experiment and the evaluation results.

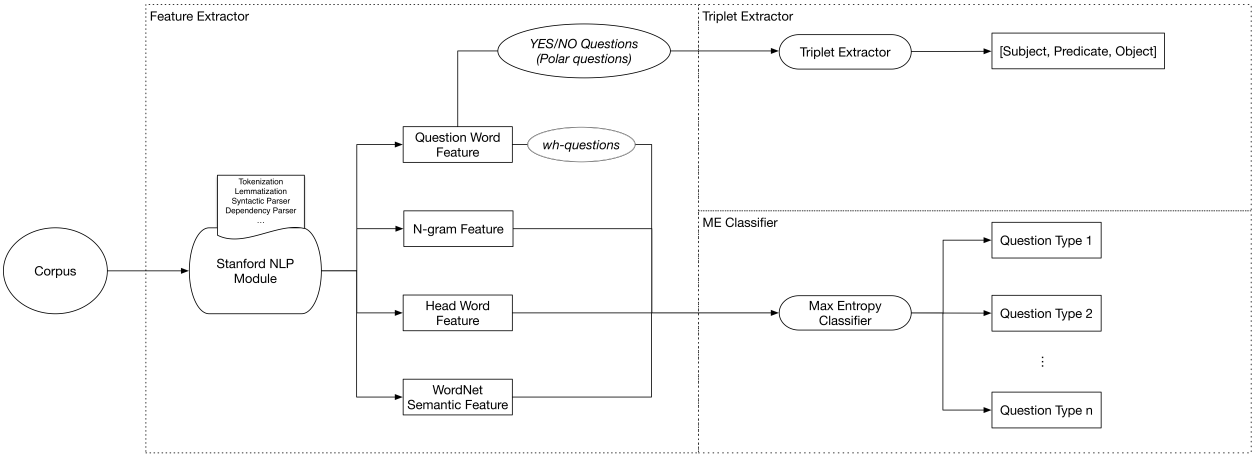


Figure 4.2: The Structure of Question Classification Module

4.1.1 Experiment Setup

4.1.1.1 Preprocessing

All input from the corpus is preprocessed via py-corenlp with Stanford CoreNLP 3.8.0. Each question is lemmatized and tokenized. We also return the syntactic parse tree and dependency parse tree for the use of the Semantic Similarity Module. Each entry contains the preprocessed sentences and their syntactic or dependency trees. The system will first calculate the corpus-related features: N-gram dictionary, Dependency-triplet dictionary, and TF-IDF value dictionary. These values are maintained with the processed-texts and related parse tree and then passed to the Semantic Similarity module.

4.1.1.2 Training

For the ME classifier in the question classification module, we adopted Experimental Data for Question Classification [77] proposed by Li [1]. It contains about 15,000 questions with predefined classes. The distribution of the dataset is shown in the Table 4.1.

Table 4.1: Experimental Data Description

Class	#	Class	#	Class	#	Class	#
ABBREV.	241	letter	30	description	774	**NUMERIC**	2480
abb	46	other	2089	manner	766	code	22
exp	195	plant	38	reason	543	count	985
ENTITY	3682	product	130	**HUMAN**	3424	date	650
animal	365	religion	9	group	529	distance	86
body	54	sport	165	individual	2691	money	183
color	119	substance	124	title	67	order	15
creative	595	symbol	31	description	137	other	154
currency	6	technique	111	**LOCATION**	2376	period	197
dis.med.	291	term	271	city	376	percent	82
event	173	vehicle	68	country	425	speed	36
food	266	word	71	mountain	67	temp	15
instrument	37	**DESCRIPTION**	3304	other	1307	size	275
lang	45	definition	1221	state	201	weight	23

4.1.1.3 Testing

The triplet extractor is syntax-based, and it is processed independently of the actual dataset. Once the ME classifier was trained, we applied another dataset from the Quora Question Pairs Datasets [78] for testing the QC module. The dataset consists of over 400,000 lines of potential question duplicate pairs. For the testing purpose, we randomly sampled 3,000 pairs of the question pairs that are already marked duplicated. This empirical evaluation measured how many question pairs were successfully categorized into the same group using the ME classifier.

Table 4.2: Experimental Data for Question Classification Result

Type	#Quest	**wh + head word**		**+ unigram**		**+ wordnet**	
		6 classes	50 classes	6 classes	50 classes	6 classes	50 classes
what	1322	86.4	84.2	87.1	85.3	88.9	86.2
which	102.6	90.3	89.9	92.3	92.1	95.4	94.2
when	392.2	100	93.1	100	94.3	100	95.6
where	356.6	94.3	92.6	95.2	93.5	96.3	93.1
who	233.8	91.1	90.2	94.2	92.3	94.2	92.3
how	488.3	95.3	83.7	96.7	85.1	96.7	89.8
why	94.2	100	100	100	100	100	100
rest	10.3	77.1	43.2	78.7	43.2	78.7	43.2
total	3000	91.46	87.3	92.42	88.51	**93.45**	**89.87**

4.1.2 Experiment Results

Huang [29] measured the contribution of individual feature types. His results showed that the wh-word and word features are highly related to the model accuracy. Also, unigram performed much better than bigram and trigram. The WordNet Direct hypernym performed better than the indirect hypernym.

Based on our literature summary, we selected wh-word and head word features as the baseline and incrementally added the unigram feature and WordNet direct hypernym feature. Table 4.2 shows the question classification accuracy of the ME classifier. We repeated the process 10 times and averaged the measures. The baseline using the wh-word and head word provided 91.46% accuracy for the coarse classes and 87.3% for the fine classes. Adding unigram and WordNet features increases accuracy 0.96% and 1.03%, respectively, for the coarse classes and 1.21% and 1.36%, respectively, for the fine classes. The best accuracy achieved for 6 classes was 93.45%, and for 50 classes it was 89.87%. We made conclusions similar to Huang, in that our results indicated that these features all positively contribute to the model prediction.

Table 4.3 shows the results of the experiment evaluated on the Quora Question Pairs Dataset. We sampled 3,000 question pairs which were only wh-questions, in order to test the ME classifiers accuracy. We observed that the ME classifier performed better predicting 5 classes than 30 classes, which is natural because the chance to be classified in different classes will be lower with a smaller

Table 4.3: Overall Performance for ME Classifier on Quora Dataset (10 times)

#Total Pairs	5 Classes		30 classes	
	Correct Pairs	Accuracy	Correct Pairs	Accuracy
3000	1420	94.67%	1249	83.27%

Table 4.4: Polar Questions Classification Result

# of Pairs	**subject**		**subject+predicate**		**subject+predicate+object**	
1,000	Set Number	64	Set Number	227	Set Number	493
	Max Set Size	42	Max Set Size	10	Max Set Size	7
	Mean Set Size	15.63	Mean Set Size	4.41	Mean Set Size	2.03
	Min Set Size	3	Min Set Size	1	Min Set Size	1
	Correct Pairs	916	Correct Pairs	747	Correct Pairs	311
	Accuracy	91.60%	Accuracy	74.70%	Accuracy	31.10%

number of classes. The accuracy for 5 classes was 94.67%, and it was 83.27% for 30 classes. The unsuccessful categorizations are mainly due to the following issues: 1) There is inherent ambiguity in classifying a question; 2) there is inconsistent labeling in the training data and test data; and 3) the parser can produce an incorrect parse tree, which would result in wrong head word extraction.

We also sampled 1,000 duplicate polar questions from the Quora Question Pairs Dataset to test the practicability of the triplet extractor. The result is shown in Table 4.4. Our classification module performed differently based on the level of information that was used for triplet indexing. We can find the details of different classification performances using the triplet as the key. If we only categorize the questions by subject, 124 different sets are grouped, and the rate for successfully categorizing two questions to the same set is 91.60%. If we use $[subject, predicate]$ as the key, on the other hand, the set number increases by 103, but the correct pairs classified decreases by 169. Moreover, if we further include object, the set number increases by 266, but the accuracy decreases from 74.7% to 31.1%. Therefore, as we increase the amount of information that has been used for indexing, our model showed more bias.

Apart from the accuracy for these two parts, we would like to calculate the time difference before and after we applied the system. The table below shows that the comparison times after the coarse classes classification are only 21.66% of the number of comparison times without clas-

Table 4.5: Latencies of WE classifiers (3000 pairs)

without classification	**5 classes**		**30 classes**	
times	times	cost percent	times	cost percent
9000000	1949230	21.66%	407552	4.53%

Table 4.6: Latencies of Triplet Extractor

w/o classification	**subject**		**subject+predicate**		**subject+predicate+object**	
times	times	cost percent	times	cost percent	times	cost percent
1000000	112896	11.29%	22700	2.27%	24157	2.42%

sification, and the number of comparison times for the fine classes classification is just 4.53% of comparison times without classification.

For polar questions, note that the sets grouped by different keys are not evenly distributed. Usually there are several sets that contain many more questions than others. For example, if we only use subject as the group key, then the words *I*, *You*, and It will have more candidates than others. To calculate the approximate comparison times, we just use the max set size as the set size. This will guarantee the upper bound of the time cost. The calculation result is shown in Table 4.6.

These results show that, even when we treat each set as the maximum size, the time saved using the triplet is large. They also suggest that it may be preferable to just use subject as the group key because it already saves enough time and including predicate and object will lower the accuracy. The classification results for the Quora Question Pairs Dataset are shown in Table 4.7.

Table 4.7: Classification Results for Quora Question Pairs Dataset

Class	#	Correct #	Percentage	Class	#	Correct #	Percentage
ABBREV.	124	60	96.77%	**ENTITY**	921	438	95.11%
abb	55	20	72.73%	animal	27	11	81.48%
exp	69	20	57.97%	body	23	9	78.26%
DESCRIPTION	782	385	98.47%	color	28	13	92.86%
definition	234	110	94.02%	creative	10	2	40.00%
description	289	130	89.97%	currency	21	10	95.24%
manner	199	92	92.46%	dis.med.	35	16	91.43%
reason	60	30	100.00%	event	113	43	76.11%
HUMAN	349	153	87.68%	food	64	30	93.75%
group	100	42	84.00%	instrument	37	12	64.86%
individual	129	48	74.42%	lang	28	13	92.86%
title	67	20	59.70%	letter	10	3	60.00%
description	53	21	79.25%	other	282	120	85.11%
LOCATION	492	234	95.12%	plant	11	2	36.36%
city	174	74	85.06%	product	42	13	61.90%
country	123	57	92.68%	religion	19	6	63.16%
mountain	67	22	65.67%	sport	53	25	94.34%
other	107	40	74.77%	substance	37	15	81.08%
state	21	7	66.67%	symbol	3	0	0.00%
NUMERIC	332	150	90.36%	technique	23	11	95.65%
code	23	10	86.96%	term	34	16	94.12%
count	10	3	60.00%	vehicle	14	6	85.71%
date	17	7	82.35%	word	7	2	57.14%
distance	49	20	81.63%				
money	43	20	93.02%				
order	15	3	40.00%				
other	58	22	75.86%				
period	20	9	90.00%				
percent	43	20	93.02%				
speed	12	6	100.00%				
temp	13	5	76.92%				
size	12	5	83.33%				
weight	17	8	94.12%				

4.2 Semantic Similarity Module

We performed experiments to calculate the Semantic Similarity of question pairs. Instead of using the question pairs from the evaluation of the QC module, which is aligned with the actual data pipeline, we evaluated the model on unrelated multiple question/sentence pair datasets. The structure of the module is illustrated in Figure 4.3 for convenience.

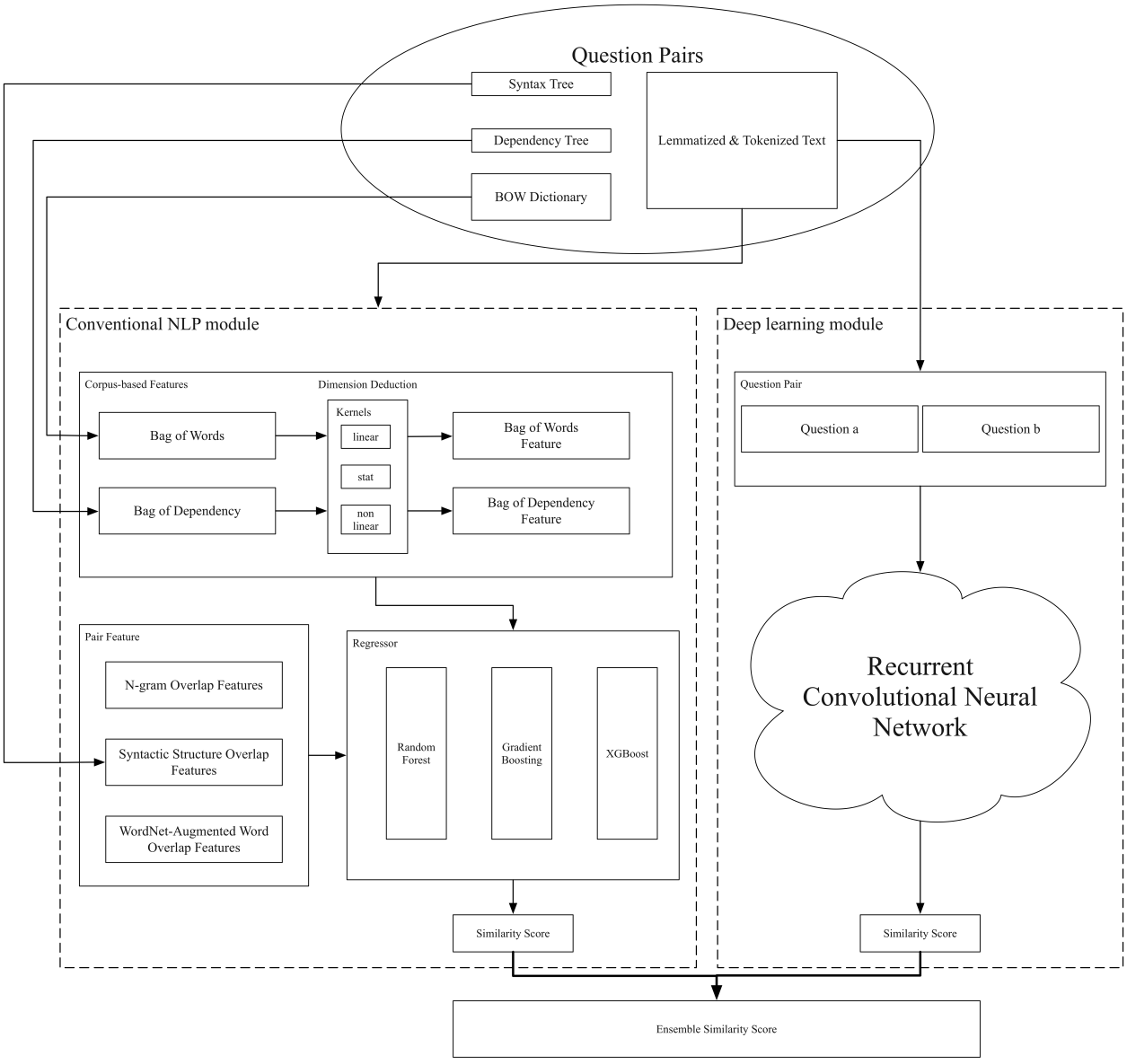


Figure 4.3: The Structure of Semantic Similarity Module

4.2.1 Experiment Setup

4.2.1.1 Datasets

To train and evaluate the Semantic Similarity module, we adopted datasets from multiple sources. In addition to the Quora Question Pairs dataset, we used the Microsoft Research Paraphrase Corpus [79], and the Sentences Involving Compositional Knowledge (SICK) dataset [80].

The Microsoft Research Paraphrase Corpus contains 5,801 pairs of sentences with 4,076 pairs for training and the remaining 1,725 pairs for testing. The training set contains 2,753 true paraphrase pairs and 1,323 false paraphrase pairs; the test set contains 1,147 true and 578 false paraphrase pairs.

The SICK dataset consists of 9,927 sentence pairs with 4,500 for training, 500 as a development set, and the remaining 4,927 in the test set. The sentences are drawn from image video descriptions. Each sentence pair is annotated with a relatedness $score \in [1, 5]$, with higher scores indicating that the two sentences are more closely-related.

4.2.1.2 Preprocessing

For the whole system, there is no need for preprocessing because the texts have already been preprocessed in the QC module. However, because both the QC module and the Semantic Similarity module are trained and tested separately, we have the similar preprocessing steps here. Before passing the data to the system, we use the Stanford CoreNLP Library to do lemmatization, tokenization, syntax parse, and dependency parse. Also, we use NLTK to calculate TF-IDF value for each word in advance.

4.2.1.3 Recurrent Convolutional Neural Network Setup

For the RCNN model, word vector, as a distributional word representation, is more appropriate as the input to the neural network. There are several pretrained word vectors. Word2Vec [55], GloVe [56] and Paragram [81]. Here we selected the most popular, Word2Vec, as the initial word vector in our model.

The hyperparameter of the neural network should be tuned based on different datasets and scenarios. We chose the most commonly used parameters. Specifically, we set learning rate, $\alpha = 0.01$, number of hidden layers $H = 100$, the dimensionality of word vector, $|e| = 300$, and the dimensionality of context vector, $|c| = 50$.

4.2.2 Experiment Evaluation

4.2.2.1 Conventional NLP Module

Table 4.8 shows the features evaluation for the conventional NLP module. The model trained with BoW features shows reasonable accuracy, just over 76%. Overall, the model trained with BoD and syntactic features shows relatively low accuracy. We thought the reason is that BoD and syntactic features are more helpful when combined with word-based features such as BoW, N-gram, and similar features, and using these features alone does not provide important semantic information. However, adding these features improved the performance. As shown in Table 4.8, adding the BoD feature to the BoW feature increased the model accuracy by 1.18% on the Quora dataset with RF classifier. Adding N-grams and syntactic features to WordNet Word Overlap features increased the accuracy by 5.15%. Our experiment showed that using all features including BoW, BoD, N-gram, syntactic, and WordNet-augmented features had the best performance. This was the case because our features were designed to reect distinctive aspects of the data that were highly related to the similarity. N-gram and BoD features focused on the context information, and the syntactic structure overlap feature targeted POS tag information.

We also found that the SVM classifier did not provide high accuracy in our context. This result might be because the SVM classifier is more sensitive to noise during data collection, and the Quora dataset is labeled by customers, so there is no guarantee that the duplicate labels are always correct. That the SVM classifier performed worse over the Quora dataset than over the MSR and SICK datasets supports this argument. Moreover, we did an experiment on the performance of the SVM classifier. The results shown in Table 4.9 indicate that the model without the SVM classifier can achieve higher accuracy. Therefore, we did not apply the SVM classifier in our system.

Table 4.8: Feature Evaluation for Conventional NLP Module

		Quora				MSR			
		RF	**SVM**	**GB**	**XGB**	**RF**	**SVM**	**GB**	**XGB**
Single features	BoW features	0.7618	0.5663	0.7398	0.7823	0.7297	0.5608	0.6579	0.6731
	BoD features	0.6204	0.4275	0.5726	0.558	0.5646	0.4309	0.5972	0.5446
	N-gram Overlap Features	0.6679	0.5302	0.716	0.7613	0.6831	0.4449	0.6318	0.6654
	Syntactic Structure Overlap Features	0.5045	0.3924	0.559	0.6069	0.5265	0.4157	0.5263	0.5791
	WordNet-Augmented Word Overlap Features	0.743	0.386	0.7353	0.6494	0.6444	0.3952	0.6415	0.718
Corpus based features	BoW + BoD	0.7736	0.5051	0.7584	0.7884	0.7429	0.5892	0.7001	0.7239
Pair based features	N-gram + Syntactic + WordNet	0.7945	0.5959	0.8109	0.7263	0.7196	0.6079	0.7518	0.8066
All features		0.8046	0.5991	0.7497	0.775	0.7519	0.6132	0.8246	0.7813
		SICK				**Quora+MSR+SICK**			
		RF	**SVM**	**GB**	**XGB**	**RF**	**SVM**	**GB**	**XGB**
Single features	BoW features	0.7064	0.6565	0.7846	0.7207	0.7689	0.6417	**0.7855**	0.7195
	BoD features	0.5224	0.4152	0.5219	0.4433	0.511	0.4107	0.518	0.5293
	N-gram Overlap Features	0.6906	0.5697	0.6807	0.6608	0.799	0.6177	0.7133	0.7334
	Syntactic Structure Overlap Features	0.5551	0.4213	0.552	0.5026	0.5656	0.4492	0.5267	0.5419
	WordNet-Augmented Word Overlap Features	0.732	0.3543	0.6222	0.7513	0.7194	0.3781	0.699	0.69
Corpus based features	BoW + BoD	0.7657	0.5786	0.8031	0.7482	0.786	0.5861	0.7877	0.7465
Pair based features	N-gram + Syntactic + WordNet	0.7102	0.5844	0.7078	0.7355	0.6585	0.5459	0.6799	0.7603
All features		0.8498	0.7305	0.7763	0.8247	0.774	0.6325	0.7816	0.808

Table 4.9: Performance Evaluation on SVM Classifier

		Accuracy (w/o SVM)	Accuracy (w/ SVM)
Single features	BoW features	0.7358	0.7035
	BoD features	0.542	0.5117
	N-gram Overlap Features	0.7003	0.6603
	Syntactic Structure Overlap Features	0.5455	0.5141
	WordNet-Augmented Word Overlap Features	0.6955	0.6162
Corpus based features	BoW + BoD	0.762	0.7127
Pair based features	N-gram + Syntactic + WordNet	0.7385	0.6997
All features		0.7968	0.7586

To evaluate the applicability of our approach, we completed the experiment both on a single dataset and on integrated datasets. We found that the overall accuracy is not decreased even if we mixed up three datasets. This means that our model is able to process more general corpus instead of specific domains, which gives us confidence that it would perform well on the NLSY and Add Health datasets as well.

4.2.2.2 Recurrent Convolutional Neural Network

Our RCNN module takes the sentence pair $[S_1, S_2]$ as the input. Each sentence consists of a word sequence $w_1, w_2, \cdots, w_n$. The w_i is the pretrained word vector. We select Word2Vec as our initial word vector. The output of the network is the binary classification on whether two sentences are semantically identical.

The difference between the RCNN and CNN models is how they represent the context. The CNN model uses a fixed-size window to represent the context information, whereas the RCNN model uses a recurrent structure to build context information with any length of distance. The performance of the CNN model is inuenced by the window size. If the window is too narrow, then it will lose the long-distance information, and if the window is too wide, then the data will be sparse and the number of parameters will increase, making the training process more difficult.

Figure 4.4 depicts F1 scores genereated by the CNN with various window sizes, and the dashed line represents the performance of the RCNN model, which was unrelated to the window size and

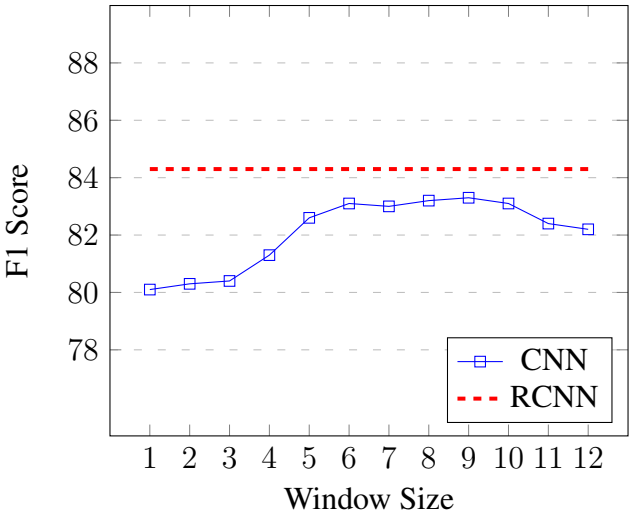


Figure 4.4: Performance of RCNN & CNN

was plotted for comparison. From the figure, we found that as the window size increased, the performance of the CNN first increased. When the window size was 9, the CNN achieved the best performance, with an F1 score of 83.3. However, as the window size kept increasing, the performance of the CNN started to decrease. We thought it was because of the data sparsity and potential overfitting. In comparison, the RCNN model performed steadily because it did not depend on the window size. The F1 score of the RCNN model is 84.3. In our context, the RCNN model demonstrated a higher F1 score compared to the highest F1 score with the CNN. This result indicates that the RCNN allows the model to cope with longer texts, and it introduces less noise than the CNN when it uses the longer window size.

4.2.2.3 Ensemble Module

In this section, we evaluate the ensemble module. Based on the experiments performed in Section 4.2.2.1 and Section 4.2.2.2, we used all features and the XGB classifier for a conventional NLP module. For the RCNN module, we set $\alpha = 0.01$, $H = 100$, $|e| = 300$, and $|c| = 50$. We performed an experiment on the both individual and mixed datasets: Quora, SICK and MSR. The evaluation result is shown in the Table 4.10. We found that the ensemble model performed better than individual ones. The overall accuracy for the ensemble module is 0.8481, which is the best

Table 4.10: Performance Evaluation for Ensemble Model

	Quora	SICK	MSR	Quora+SICK+MSR
Conventional NLP Module + XGB	0.775	0.7813	0.8247	0.8084
RCNN	0.8413	0.8279	0.8372	0.843
Ensemble	0.8422	0.841	0.843	**0.8481**

performance. It shows that the combination not only improves the performance but also increases the robustness for modeling similarity of heterogeneous sources.

Chapter 5

Conclusion and Future Work

In this paper, we presented an integrated system that enables efficient semantic integration on heterogeneous sources. The system is designed to cope with domain-independent input.

RQ-1 To integrate semantically similar questions from multiple questionnaires, our system is equipped with a Question Classification module that can quickly subdivide the questions to coarse categories. Without element-wise comparison, it can significantly reduce the latencies required to find semantically and syntactically alike questions.

RQ-2 To calculate a similarity score for semantically and syntactically alike questions, our system built a hybrid system by combining conventional NLP techniques and the Recurrent Convolutional Neural Network model. The overall accuracy was as high as 0.8481. The extensive experimental results show that the combination not only improves the performance but also increases the robustness for modeling the similarity of heterogeneous sources.

While the proposed system is good enough to process multiple questionnaires integration, there is room for improvements to the system. This involves addressing the following possible aspects:

1. Few questions in a questionnaire dataset may contain specific "questionnaire-characteristics." For example, *"Since [date of last interview] what months have you lived with your [mother (figure)/father (figure)]?"* (NLSY-R4024144). The question may contain fill-in fields which will introduce a degree of error. Either a deep data cleaning process or more detailed feature engineering may resolve the issue.

2. To further integrate multiple questionnaire databases, we need to analyze the candidate answers as well. For example, answers to the question *"How tall are you?"* may have different units in different datasets. One may use inches and another may use centimeters. Fur-

thermore, the system should be able to convert one measurement index to another for easy integration.

3. The system should also be integrated with an error-trace module. When there are misclassified question-pairs, the system should be able to trace back to point out which step most likely causes this error. Since the QC module and STSTS module are evaluated independently, we did not build such a module and would like to leave it to future work.

Bibliography

[1] Xin Li and Dan Roth. Learning question classifiers. In *Proceedings of the 19th international conference on Computational linguistics-Volume 1*, pages 1–7. Association for Computational Linguistics.

[2] Håkan Sundblad. *Question classification in question answering systems*. Thesis, 2007.

[3] Babak Loni. A survey of state-of-the-art methods on question classification. 2011.

[4] Natsuda Laokulrat. A survey on question classification techniques for question answering. , 2(1), 2016.

[5] Sangmi Lee Pallickara, Shrideep Pallickara, Milija Zupanski, and Stephen Sullivan. Efficient metadata generation to enable interactive data discovery over large-scale scientific data collections. In *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*, pages 573–580. IEEE, 2010.

[6] Matthew Malensek, Sangmi Lee Pallickara, and Shrideep Pallickara. Galileo: A framework for distributed storage of high-throughput data streams. In *Utility and Cloud Computing (UCC), 2011 Fourth IEEE International Conference on*, pages 17–24. IEEE, 2011.

[7] Cameron Tolooee, Sangmi Lee Pallickara, and Asa Ben-Hur. Mendel: A distributed storage framework for similarity searching over sequencing data. In *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 790–799. IEEE, 2016.

[8] Dennis Gannon, Beth Plale, Marcus Christie, Yi Huang, Scott Jensen, Ning Liu, Suresh Marru, Sangmi Lee Pallickara, Srinath Perera, Satoshi Shirasuna, et al. Building grid portals for e-science: A service oriented architecture. *High Performance Computing and Grids in Action*, 2007.

[9] Yogesh L Simmhan, Sangmi Lee Pallickara, Nithya N Vijayakumar, and Beth Plale. Data management in dynamic environment-driven computational science. In *Grid-based problem solving environments*, pages 317–333. Springer, 2007.

[10] Sangmi Lee Pallickara, Shrideep Pallickara, and Marlon Pierce. Scientific data management in the cloud: A survey of technologies, approaches and challenges. In *Handbook of Cloud Computing*, pages 517–533. Springer, 2010.

[11] Sangmi Lee Pallickara, Shrideep Pallickara, and Milija Zupanski. Towards efficient data search and subsetting of large-scale atmospheric datasets. *Future Generation Computer Systems*, 28(1):112–118, 2012.

[12] Scott Jensen, Beth Plale, Sangmi Lee Pallickara, and Yiming Sun. A hybrid xml-relational grid metadata catalog. In *Parallel Processing Workshops, 2006. ICPP 2006 Workshops. 2006 International Conference on*, pages 8–pp. IEEE, 2006.

[13] Sangmi Lee Pallickara and Marlon Pierce. Swarm: Scheduling large-scale jobs over the loosely-coupled hpc clusters. In *SWARM: Scheduling Large-Scale Jobs over the Loosely-Coupled HPC Clusters*, pages 285–292. IEEE, 2008.

[14] Matthew Malensek, Sangmi Pallickara, and Shrideep Pallickara. Analytic queries over geospatial time-series data using distributed hash tables. *IEEE Transactions on Knowledge and Data Engineering*, 28(6):1408–1422, 2016.

[15] Matthew Malensek, Sangmi Pallickara, and Shrideep Pallickara. Fast, ad hoc query evaluations over multidimensional geospatial datasets. *IEEE Transactions on Cloud Computing*, 5(1):28–42, 2017.

[16] Matthew Malensek, Sangmi Pallickara, and Shrideep Pallickara. Polygon-based query evaluation over geospatial data using distributed hash tables. In *Proceedings of the 2013 IEEE/ACM 6th International Conference on Utility and Cloud Computing*, pages 219–226. IEEE Computer Society, 2013.

[17] Matthew Malensek, Sangmi Pallickara, and Shrideep Pallickara. Evaluating geospatial geometry and proximity queries using distributed hash tables. *Computing in Science & Engineering*, 16(4):53–61, 2014.

[18] Matthew Malensek, Walid Budgaga, Ryan Stern, Shrideep Pallickara, and Sangmi Pallickara. Trident: Distributed storage, analysis, and exploration of multidimensional phenomena. *IEEE Transactions on Big Data*, 2018.

[19] Thilina Buddhika, Matthew Malensek, Sangmi Lee Pallickara, and Shrideep Pallickara. Synopsis: A distributed sketch over voluminous spatiotemporal observational streams. *IEEE Transactions on Knowledge and Data Engineering*, 29(11):2552–2566, 2017.

[20] Walid Budgaga, Matthew Malensek, Sangmi Pallickara, Neil Harvey, F Jay Breidt, and Shrideep Pallickara. Predictive analytics using statistical, learning, and ensemble methods to support real-time exploration of discrete event simulations. *Future Generation Computer Systems*, 56:360–374, 2016.

[21] Walid Budgaga, Matthew Malensek, Sangmi Lee Pallickara, and Shrideep Pallickara. A framework for scalable real-time anomaly detection over voluminous, geospatial data streams. *Concurrency and Computation: Practice and Experience*, 29(12):e4106, 2017.

[22] Thilina Buddhika and Shrideep Pallickara. Neptune: Real time stream processing for internet of things and sensing environments. In *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 1143–1152. IEEE, 2016.

[23] Thilina Buddhika, Ryan Stern, Kira Lindburg, Kathleen Ericson, and Shrideep Pallickara. Online scheduling and interference alleviation for low-latency, high-throughput processing of data streams. *IEEE Transactions on Parallel and Distributed Systems*, 28(12):3553–3569, 2017.

[24] Wes Lloyd, Shrideep Pallickara, Olaf David, Jim Lyon, Mazdak Arabi, and Ken Rojas. Performance modeling to support multi-tier application deployment to infrastructure-as-a-

service clouds. In *Proceedings of the 2012 IEEE/ACM Fifth International Conference on Utility and Cloud Computing*, pages 73–80. IEEE Computer Society, 2012.

[25] Yuanjie Liu, Shasha Li, Yunbo Cao, Chin-Yew Lin, Dingyi Han, and Yong Yu. Understanding and summarizing answers in community-based question answering services. In *Proceedings of the 22nd International Conference on Computational Linguistics-Volume 1*, pages 497–504. Association for Computational Linguistics.

[26] Long Chen, Dell Zhang, and Levene Mark. Understanding user intent in community question answering. In *Proceedings of the 21st International Conference on World Wide Web*, pages 823–828. ACM.

[27] Fan Bu, Xingwei Zhu, Yu Hao, and Xiaoyan Zhu. Function-based question classification for general qa. In *Proceedings of the 2010 conference on empirical methods in natural language processing*, pages 1119–1128. Association for Computational Linguistics.

[28] Guangyu Feng, Kun Xiong, Yang Tang, Anqi Cui, Jing Bai, Hang Li, Qiang Yang, and Ming Li. Question classification by approximating semantics. In *Proceedings of the 24th International Conference on World Wide Web*, pages 407–417. ACM.

[29] Zhiheng Huang, Marcus Thint, and Zengchang Qin. Question classification using head words and their hypernyms. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 927–936. Association for Computational Linguistics.

[30] Thamar Solorio, Manuel Pérez-Coutino, Manuel Montes-y Gémez, Luis Villasenor-Pineda, and Aurelio López-López. A language independent method for question classification. In *Proceedings of the 20th international conference on Computational Linguistics*, page 1374. Association for Computational Linguistics.

[31] Minh Le Nguyen, Nguyen Thanh Tri, and Akira Shimazu. Subtree mining for question classification problem. In *IJCAI*, pages 1695–1700.

[32] Dell Zhang and Wee Sun Lee. Question classification using support vector machines. In *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, pages 26–32. ACM.

[33] Naoaki Okazaki, Yutaka Matsuo, Naohiro Matsumura, and Mitsuru Ishizuka. Sentence extraction by spreading activation through sentence similarity. *IEICE TRANSACTIONS on Information and Systems*, 86(9):1686–1694, 2003.

[34] Jung-Hsien Chiang and Hsu-Chun Yu. Literature extraction of protein functions using sentence pattern mining. *IEEE Transactions on Knowledge and Data Engineering*, 17(8):1088–1098, 2005.

[35] Thomas K Landauer, Peter W Foltz, and Darrell Laham. An introduction to latent semantic analysis. *Discourse processes*, 25(2-3):259–284, 1998.

[36] Peter W Foltz, Walter Kintsch, and Thomas K Landauer. The measurement of textual coherence with latent semantic analysis. *Discourse processes*, 25(2-3):285–307, 1998.

[37] Curt Burgess, Kay Livesay, and Kevin Lund. Explorations in context space: Words, sentences, discourse. *Discourse Processes*, 25(2-3):211–257, 1998.

[38] Yuhua Li, David McLean, Zuhair A Bandar, James D O'shea, and Keeley Crockett. Sentence similarity based on semantic nets and corpus statistics. *IEEE transactions on knowledge and data engineering*, 18(8):1138–1150, 2006.

[39] Rada Mihalcea, Courtney Corley, and Carlo Strapparava. Corpus-based and knowledge-based measures of text semantic similarity. In *AAAI*, volume 6, pages 775–780.

[40] Jesús Oliva, José Ignacio Serrano, María Dolores del Castillo, and Ángel Iglesias. Symss: A syntax-based measure for short-text semantic similarity. *Data & Knowledge Engineering*, 70(4):390–405, 2011.

[41] Aminul Islam and Diana Inkpen. Semantic similarity of short texts. *Recent Advances in Natural Language Processing V*, 309:227–236, 2009.

[42] Long Qiu, Min-Yen Kan, and Tat-Seng Chua. Paraphrase recognition via dissimilarity significance classification. In *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*, pages 18–26. Association for Computational Linguistics.

[43] 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*, pages 468–476. Association for Computational Linguistics.

[44] Richard Socher, Eric H Huang, Jeffrey Pennin, Christopher D Manning, and Andrew Y Ng. Dynamic pooling and unfolding recursive autoencoders for paraphrase detection. In *Advances in neural information processing systems*, pages 801–809.

[45] Yangfeng Ji and Jacob Eisenstein. Discriminative improvements to distributional sentence similarity. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 891–896.

[46] Delia Rusu, Lorand Dali, Blaz Fortuna, Marko Grobelnik, and Dunja Mladenic. Triplet extraction from sentences. In *Proceedings of the 10th International Multiconference" Information Society-IS*, pages 8–12.

[47] Apache opennlp. http://opennlp.apache.org/. (Accessed on 05/19/2018).

[48] The stanford natural language processing group. https://nlp.stanford.edu/software/lex-parser.shtml. (Accessed on 05/19/2018).

[49] Berkeley parser. https://github.com/slavpetrov/berkeleyparser. (Accessed on 05/19/2018).

[50] Dan Klein and Christopher D Manning. Accurate unlexicalized parsing. In *Proceedings of the 41st annual meeting of the association for computational linguistics*.

[51] Michael Collins. Head-driven statistical models for natural language parsing. *Computational linguistics*, 29(4):589–637, 2003.

[52] George A Miller. Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41, 1995.

[53] Mark Stevenson and Yorick Wilks. Word sense disambiguation. *The Oxford Handbook of Comp. Linguistics*, pages 249–265, 2003.

[54] Michael Lesk. Automatic sense disambiguation using machine readable dictionaries: how to tell a pine cone from an ice cream cone. In *Proceedings of the 5th annual international conference on Systems documentation*, pages 24–26. ACM, 1986.

[55] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[56] Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543.

[57] Frane ari, Goran Glava, Mladen Karan, Jan najder, and Bojana Dalbelo Bai. Takelab: Systems for measuring semantic text similarity. In *Proceedings of the First Joint Conference on Lexical and Computational Semantics-Volume 1: Proceedings of the main conference and the shared task, and Volume 2: Proceedings of the Sixth International Workshop on Semantic Evaluation*, pages 441–448. Association for Computational Linguistics.

[58] Kai Wang, Zhaoyan Ming, and Tat-Seng Chua. A syntactic tree matching approach to finding similar questions in community-based qa services. In *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, pages 187–194. ACM.

[59] Michael Collins and Nigel Duffy. Convolution kernels for natural language. In *Advances in neural information processing systems*, pages 625–632.

[60] scikit-learn: machine learning in python scikit-learn 0.19.1 documentation. http://scikit-learn.org/stable/. (Accessed on 05/23/2018).

[61] dmlc/xgboost: Scalable, portable and distributed gradient boosting (gbdt, gbrt or gbm) library, for python, r, java, scala, c++ and more. runs on single machine, hadoop, spark, flink and dataflow. https://github.com/dmlc/xgboost. (Accessed on 05/23/2018).

[62] Richard Socher, Jeffrey Pennington, Eric H Huang, Andrew Y Ng, and Christopher D Manning. Semi-supervised recursive autoencoders for predicting sentiment distributions. In *Proceedings of the conference on empirical methods in natural language processing*, pages 151–161. Association for Computational Linguistics.

[63] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1631–1642.

[64] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12(Aug):2493–2537, 2011.

[65] Nal Kalchbrenner and Phil Blunsom. Recurrent convolutional neural networks for discourse compositionality. *arXiv preprint arXiv:1306.3584*, 2013.

[66] Siwei Lai, Liheng Xu, Kang Liu, and Jun Zhao. Recurrent convolutional neural networks for text classification. In *AAAI*, volume 333, pages 2267–2273.

[67] Richard Socher, Andrej Karpathy, Quoc V Le, Christopher D Manning, and Andrew Y Ng. Grounded compositional semantics for finding and describing images with sentences. *Transactions of the Association of Computational Linguistics*, 2(1):207–218, 2014.

[68] Richard Socher, Cliff C Lin, Chris Manning, and Andrew Y Ng. Parsing natural scenes and natural language with recursive neural networks. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 129–136.

[69] Jeffrey L Elman. Finding structure in time. *Cognitive science*, 14(2):179–211, 1990.

[70] Kunihiko Fukushima and Sei Miyake. *Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition*, pages 267–285. Springer, 1982.

[71] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[72] Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. A convolutional neural network for modelling sentences. *arXiv preprint arXiv:1404.2188*, 2014.

[73] Yoon Kim. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*, 2014.

[74] Léon Bottou. Stochastic gradient learning in neural networks. *Proceedings of Neuro-Nmes*, 91(8):0, 1991.

[75] David C Plaut and Geoffrey E Hinton. Learning sets of filters using back-propagation. *Computer Speech & Language*, 2(1):35–61, 1987.

[76] smilli/py-corenlp: Python wrapper for stanford corenlp. https://github.com/smilli/py-corenlp. (Accessed on 05/23/2018).

[77] Learning question classifiers. http://cogcomp.org/Data/QA/QC/. (Accessed on 05/23/2018).

[78] First quora dataset release: Question pairs - data @ quora - quora. https://data.quora.com/First-Quora-Dataset-Release-Question-Pairs. (Accessed on 05/23/2018).

[79] Download microsoft research paraphrase corpus from official microsoft download center. https://www.microsoft.com/en-us/download/details.aspx?id=52398. (Accessed on 05/24/2018).

[80] Sick. http://clic.cimec.unitn.it/composes/sick.html. (Accessed on 05/24/2018).

[81] John Wieting, Mohit Bansal, Kevin Gimpel, and Karen Livescu. Towards universal paraphrastic sentence embeddings. *arXiv preprint arXiv:1511.08198*, 2015.