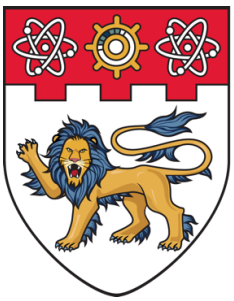

NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

# SCSE18-0377

# Single-domain Fine-grained Sentiment Analysis

Submitted in Partial Fulfillment of the Requirements
for the Degree of Bachelor of Computer Engineering
of the Nanyang Technological University

by

**Wang Yiqi**

**Supervisor: Associate Professor Pan, Sinno Jialin**

**Examiner: Associate Professor Erik Cambria**

School of Computer Science and Engineering

2019

# Abstract

Fine-grained sentiment analysis has attracted a great deal of attention recently due to its various applications and related challenging research topics. One of the main tasks is to extract both sentiment words and topic words from a sentence, where a sentiment word is an opinion expression and the topic word is an aspect on which the opinion word is expressed. In this project, I use a joint model which integrates recursive neural networks and sequence labelling models for explicit aspect and opinion terms extraction. Based on the experiment result, I also design an interactive application which utilizes the joint model and is able to take user input and extract aspect and opinion expressions from given text.

Word embedding is a popular natural language processing(NLP) method which aims at learning vector representations of words from documents. In this project, I use Yelp Challenge dataset for word embedding pre-training and SemEval Challenge 2014 dataset to evaluate my models. Previous studies have shown that a joint model of Recursive Neural Networks(RNN) and sequence labelling methods is promising for this task, because it learns high-level discriminative features and dually propagates information between aspect and opinion terms (Wang, Pan, Dahlmeier, & Xiao, 2016). Hence, I conduct experiments using Recursive Neural Network integrated with different sequence labelling methods respectively: Conditional Random Fields(CRFs) and Bi-directional LSTM(Bi-LSTM). The experimental result verifies the robustness of the joint models. Based on the well-tuned sentiment analysis models, an interactive application is built using a Python web framework, Flask. Two types of UI are designed for the application: one takes text input, the other takes url input. Analysis results are shown as sentences where aspect and opinion terms are highlighted in different colors.

# Acknowledge

I would like to thank Professor Sinno Jialin Pan, my supervisor, for his incisive advice, kind encouragement and prompt response whenever I asked a question.

I would like to thank Processor Erik Cambria, my examiner, for his time and effort spent in reviewing my work and providing valuable suggestion.

I would also like to thank Wenya Wang for her patience, invaluable help and professional knowledge.

I also want to thank Luo Jingqi for helping me with key concepts in NLP and Ding Ni for inspiring me to build application with Flask.

I want to thank Huang Tiancheng as well for reviewing my report and providing valuable suggestions on my writing styles.

Last but not least, I appreciate School of Computer Science and Engineering for providing all the help, support and opportunities over the past four years.

## Table of Contents

## List of Figures & Tables

# 1. Introduction

## 1.1. Background

Fine-grained sentiment analysis aims to extract important information, e.g. opinion targets, opinion expressions, target categories, and opinion polarities from user-generated content such as microblogs, reviews, etc. In this project, my main task is to extract explicit aspects of an entity and the corresponding opinion expressions from the text of the restaurant domain and laptop domain datasets. For example, in a restaurant review "*The wine list is interesting and has many good values.*", a sentiment analysis model is supposed to predict "*wine list*" as aspect term, with "*interesting*" and "*good values*" as opinion terms.

In previous works, one approach is to accumulate aspect and opinion terms from known or extracted seed collection and utilize syntactic rules between aspect and opinion terms (Qiu, Liu, Bu, & Chen, 2011). This method is based on the observation that opinion terms are always trying to modify aspect terms. In the above example, if we can tell *"interesting"* as opinion word from any seed collection, then *"wine list"* can probably be inferred as aspect word because "*interesting* " is the modifier of it. But this method highly relies on hand-coded rules and has its own restrictions such like opinion words are expected to be only adjectives. However, from the above sample, we can find that this assumption does not always hold because "*good values*" is a noun phrase but works as a modifier. Other approaches integrate both the lexicalization techniques and linguistic features into a HMMs model to handle contextual information for the tag assignment to words. (Jin & Ho, 2009) The tagging algorithm finds the most probable sequence of tags by using hand-crafted features. However, this approach requires extensive efforts to categorize hybrid tags and design hand-crafted features. Meanwhile, it also ignores higher order interactions between aspect and opinion terms.

Therefore, a joint model named Recursive Neural Conditional Random Fields (RNCRF) is created to overcome the limitations of the mentioned methods (Wang, Pan, Dahlmeier, & Xiao, 2016). In this project, I build two joint models based on Wang et al.'s work. The joint model contains two main components. The first component is a dependency tree recursive neural network, which learns a high-level

representation for each word in each sentence. The feature representation of each word is learned interactively through the dependency structure. The output of RNN is then fed into a sequence labelling model to learn the discriminative mapping from features to labels, i.e., aspects, opinions, or others. I used two different labelling methods for comparison. One is Conditional Random Fields (CRF), which is the same as Wang proposed. The other is Bi-directional LSTM, which is for efficiently using both past and future input features (Huang, Xu, & Yu, 2015).

## 1.2. Objectives

In this project, I first perform data pre-processing on the Yelp Challenge dataset and Amazon review dataset. Then I reimplement RNCRF (Wang, Pan, Dahlmeier, & Xiao, 2016). Legacy dependencies are updated and necessary refactoring is made due to the changes in Python syntax. I also build a new joint model with Bi-directional LSTM, which takes the higher-level representation generated by the pre-trained model as input and encodes context information from both directions within the sentence. Model evaluation is conducted using datasets from SemEval Challenge 2014. In addition, I also build an interactive application using Flask, where the user can input either text or Uniform Resource Locator (URL) then get sentiment analysis results. If the URL is passed to the application, the application will first perform text extraction task before proceeding to sentiment analysis.

## 1.3. Report Structure

This report consists of the following sections:

- section 1 provides background information and objective of this project;
- section 2 investigates the existing solutions for aspect-based sentiment analysis;
- section 3 is problem description;
- section 4 describes the detailed methodology used in the project, including word-embedding, dependency-tree RNN, Recursive Neural CRFs, and Bi-directional LSTM;

- section 5 shows how the experiment is conducted;
- section 6 shows demos of the application and implementation of the application;
- section 7 gives the conclusion of this project and provides future direction for the project;
- section 8 gives a brief description of the resources I have used in the project.

# 2. Related Work

## 2.1. Utilization of Dependency Relation in Sentiment Analysis

Semantic parser has been playing an important role in many NLP tasks. It breaks text into clauses from which people can actually deconstruct those clauses into concepts. The outcome of a parser is informative, such as part-of-speech(POS) tag for each individual word and the dependency relations between pairs of words within one sentence. Semantic parser has facilitated various NLP tasks. There is existing approach using the semantic parser for deconstructing text into NLP concepts, which then get represented through a vector space of common-sense knowledge (Poria, Cambria, Winterstein, & Huang, 2014). Considering each verb and its noun phrases associated by the semantic parser, one or more concepts can be extracted by using the POS-based bigram algorithm or the event concept extraction algorithm. The concepts are be fed to a common-sense reasoning algorithm for further processing.

The semantic parsing of one sentence has a tree structure in nature. Recursive Neural Networks(RNNs) have shown successful results in learning sequence and tree structures in many NLP tasks such as phrase continuous representation, sentence-level sentiment analysis based on word embedding and language parsing. The tree structures used for RNNs includes constituency tree and dependency tree. In this project, I use the dependency tree for RNN pre-training task, where each node in a dependency tree represents a word connected to its parent nodes through dependency relations. By using dependency-tree RNN(DT-RNN), we are able to extract word-level representations containing both syntactic relations and semantic meaning. The output of RNN is saved and then fed to a contextual learning model for sequence labelling.

## 2.2. Word Embedding

Word Embedding is one of the most popular and useful representations of document vocabulary. It is capable of capturing syntactic and semantic word relationships by learning low-dimensional continuously-valued vector representation of words. In

other words, it is a vector representation of a particular work. There are many word embedding techniques such as Skip-gram, GloVe, and Word2Vec, which have been proven to facilitate various NLP tasks. In this project, I utilize Word2vec (Mikolov, Chen, Corrado, & Dean, 2013) to construct such embedding, which can be obtained using two methods: Skip Grams and Common Bag of Words(CBOW). CBOW model takes the context of each word as the input and tried to predict the word corresponding to the context. Skip Grams, on the other hand, use target words to predict the context and then produce the representation. Both methods have their own advantages and disadvantages. According to Mikolov, Skip Gram works well with a small amount of data while CBOW is faster and has better representation for frequent words.

The assumption behind word embedding approaches is that by relying on statistical information such as word co-occurrence frequencies, word embedding approaches can learn distributional vector representations that capture semantic meaning of words. However, the representations learned by such algorithms can be very general and independent from any specific task. Therefore, they may not be optimal for a given NLP task, especially when prior knowledge is available. Training word embedding for specific tasks by incorporating prior knowledge has been proven to sufficiently improve the quality of word embedding for sentiment analysis task (Li, et al., 2017). In this project, I use manually annotated labels(aspect, opinion, etc.) as prior knowledge and training word embedding with a recursive neural network to improve the performance of sentiment analysis models.

## 2.3. Conditional Random Fields

Conditional Random Fields(CRFs) are a discriminative undirected probabilistic graphical model which encodes known relationships between observations and construct consistent interpretations. In graphical modelling, a distribution over many variables is represented as product of local functions that each depends on a much smaller subset of variables. CRFs take the concept of context into account and are widely used to predict the sequence of labels for a sequence of input samples.

In this project, I employed linear-chain CRFs where feature functions are introduced. In CRFs, each feature function is a function that takes as input a

sentence, the position of the current word, the label of the current word, and the labels of words within the context window of the current word and outputs a real-valued number. Each feature function is assigned with a weight. By summing up all the weighted features over all words in the sentence, we can score the labelling of a sentence.

## 2.4. Deep Learning for Sentiment Analysis

In recent years, sentiment analysis has become increasingly popular and deep learning methods have been proven to achieve great accuracy in related tasks.

Neural sequential models, especially Long Short-Term Memory (LSTM) networks are of growing interests for their capability of representing sequential information. By incorporating with attention mechanism, which takes an external memory and produces probability distribution qualifying the importance of input features, LSTM is able to achieve promising performance in aspect-based sentiment analysis (Ma, Peng, & Cambria, 2018).

Besides capturing sequential information, we can also make prediction based on the neighbor information of the current tag. There are two different approaches to make use of neighbor tag information. The first one is to focus on sentence-level instant of an individual time step, which is CRF. The second one is to predict a time distribution of tags and use decoding to find optimal tag sequences. This is where the word of maximum entropy classifier falls. In existing work, a combined model based on Recurrent Neural Networks (RNNs) architecture and word embeddings is proposed to explicitly extract opinion expressions and identify the aspect of the opinion (Liu, Joty, & Meng, 2015). The RNN model fine-tunes the word vectors during training to learn task-specific embeddings. The proposed model heavily depends on the quality of word embeddings, so it is necessary to perform word embeddings pre-training from several external sources.

## 2.5. Bidirectional LSTM for Sequence Tagging

LSTM uses purpose-built memory cells to store information, which makes it better at finding and exploiting long-range context. But one shortcoming of conventional

Recurrent Neural Networks(RNNs) is that they only utilize previous context information. It makes sense for many sequence prediction tasks such as time series forecasting where time only flows from the past to the future. However, in aspect-based sentiment analysis, the entire sentence is processed at once, so there is no reason not to exploit future context as well. We can thus use a bidirectional RNN with LSTM as proposed by Graves et al. (Graves, Mohamed, & Hinton, 2013). Bidirectional RNNs can be implemented by replacing each hidden sequence in traditional LSTM with the forward and backward sequences. The forward and backward pass over time is carried out in a similar way as the regular network, except that we need to unfold the hidden states for all time steps.

# 3. Problem Description

Given a training set of customer reviews in the restaurant domain, which can be denoted as S = $\{s_1, s_2, \ldots, s_N\}$, where N is the number of sentences. For any $s_i \in S$, it may contain a set of aspect terms $A_i = \{a_{i1}, \ldots, a_{im}\}$, where each aspect term can be either a single word or a sequence of words. It may also contain a set of opinion term $O_i = \{o_{i1}, \ldots, o_{il}\}$, where each opinion term can be a single word or a sequence of sentiment expressions. The task is to learn a classifier to extract set $A_i$ and set $O_i$ from each review sentence $s_i \in S$.

In this project, this task is formulated as a sequence tagging problem using IOB encoding scheme. There are in total 5 classes: BA (beginning of aspect), IA (inside of aspect), BO(beginning of opinion), IO(inside of opinion), O(outside of aspect/ opinion). Let L = {BA, IA, BO, IO, O}. Each review sentence $s_i$ can be taken as a sequence of words $s_i \rightarrow \{w_{i1}, \ldots, w_{in_i}\}$ where $n_i$ is the total number of words in the sentence. Each word $w_{ij} \in s_i$ is labeled as one of the 5 classes. To evaluate the performance of the models, a testing set of reviews is used as $S_i' = \{s_i', \ldots, s_{N'}'\}$, where $N'$ is the number of sentences in the testing set. For each sentence $s_i' \in S'$, the goal is to predict class label $y_{ir}' \in L$ for each word $w_{im}' \in s_i'$.

# 4. Methods

## 4.1. Recursive Neural CRFs

RNCRF consists of two components:

1). DT-RNN to learn the high-level representation of each word

2). CRF which takes the output of DT-RNN as input to capture contextual information within each sentence for explicit aspect and opinion term extraction.

### 4.1.1. Dependency-Tree RNN

Before the pre-training task starts, we first go through all the sentences in the training and testing set and save each word in the vocabulary. Then we associate each word $w_i$ in the vocabulary with a feature vector $x_i \in R^d$, which matches the $i^{th}$ column of the word embedding matrix $W_e \in R^{d \times v}$, where $d$ is the word embedding dimension and $v$ is the size of vocabulary.

For each sentence, we pass it to the Stanford CoreNLP server (Stanford CoreNLP – Natural language software, n.d.) and build dependency tree based on the parsing result. An example of raw parse is shown below, where each column represents word, pos-tag, index of parent word and relation. Note that Stanford CoreNLP always generate a 'root' word for each sentence and index 0 is reserved for 'root'.

```
The         DT    2     det
service     NN    5     nsubj
is          VBZ   5     cop
pretty      RB    5     advmod
good  JJ    0     ROOT
```

An example of dependency tree is shown in Figure 1.1, where each edge starts from the parent and points to the dependent.

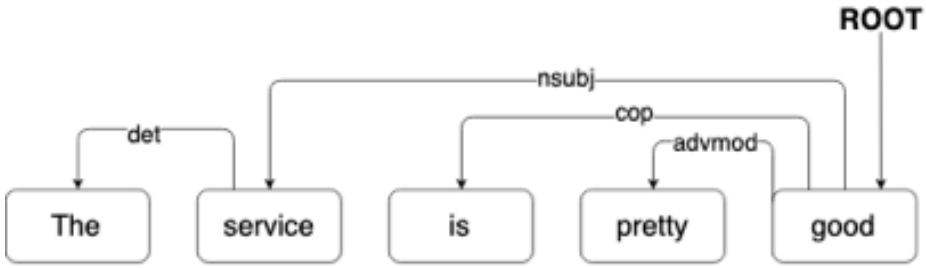


Figure 1. 1 Example of dependency tree

There are three types of nodes in a DT-RNN: leaf node, internal node and the root node. Each node $n$ associates with a word $w$ and a feature vector $x_w$. A common transformation matrix $W_v \in R^{d\times d}$ is used to map feature vector $x_w$ at node $n$ to its hidden vector $h_w \in R^d$, where $d$ is same as the embedding dimension. In addition, we also keep a dictionary $W_r$ which takes relations as keys and matrix $W_{r[rel]} \in R^{d\times d}$ as values.

The hidden vector $h_w$ at node $n$ is computed from its own feature vector, the transformation matrix $W_v$ and the hidden vectors of its children $h_{child(n)}$ with corresponding relation matrix $W_r[rel]$. For example, for the parse tree given in Figure 1.1, we can computer the hidden vector of each leaf word as follows,

$$h_{The} = tanh(W_v \cdot x_{The} + b),$$
$$h_{is} = tanh(W_v \cdot x_{is} + b),$$
$$h_{pretty} = tanh(W_v \cdot x_{pretty} + b)$$

where we use $tanh(\cdot)$ as activation function and $b$ represents bias term. Next, we can recursively generate hidden vectors for internal nodes as follows,

$$h_{service} = tanh(W_v \cdot x_{service} + Wr[det] \cdot h_{the} + b)$$

Finally, the hidden vector for the root node can then be generates as follow,

$$h_{good} = tanh(W_v \cdot x_{good} + W_r[nsubj] \cdot h_{service} + W_r[cop] \cdot h_{is} + W_r[advmod] \cdot h_{good} + b)$$

The generated DT-RNN is shown as Figure 1.2.

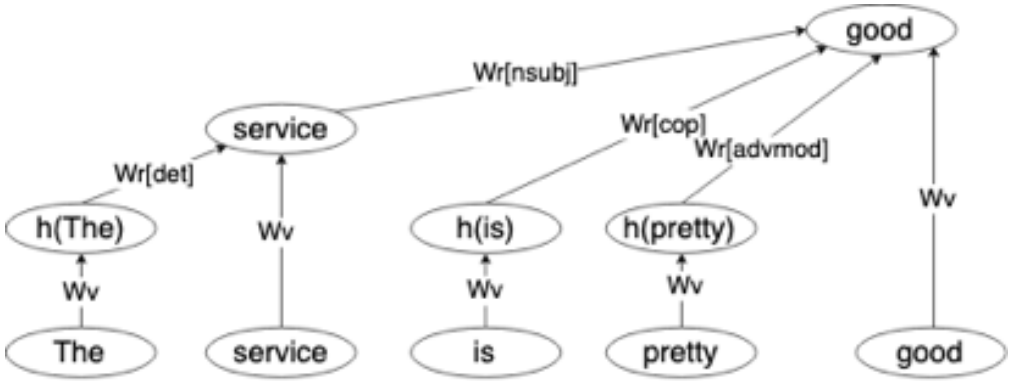


**Figure 1. 2 Example of DT-RNN**

In general, a hidden vector for any node $n$ associated with a word vector $x_w$ can be computer as follows,

$$h_n = f(W_v \cdot x_w + b + \sum_{k \in K_n} W_{r_{nk}} \cdot h_k)$$

where $K_n$ indicates the set of children of node $n$, $r_{nk}$ indicates the dependency relation between node $n$ and its child node $k$, and $h_k$ represents the hidden vector of child node $k$.

The parameters of DT-RNN, $\Theta_{RNN} = \{W_v, W_r, W_e, b\}$, are learned during training using backpropagation through structure(BPTS). (Goller & Küchler, 1996) BPTS is principally the same as general propagation as follow,

$$\delta^{(l)} = ((W^{(l)})^T \delta^{(l+1)}) \cdot f'(z^{(l)})$$
$$\frac{\partial}{\partial W^{(l)}} E_R = \delta^{(l+1)}(a^l)^T + \lambda W^{(l)}$$

But there are three differences resulting from the recursive tree structure: we need to sum derivatives of $W$ from all nodes; during forward propagation, the hidden vector of parent is computed using its children nodes, thus we need to split derivatives at children nodes and compute error with respective to each of them; last but not least, we need to add error message from parent node and the current node itself.

### 4.1.2. Integration with CRFs

The output of RNN, which is the hidden representation of each word in the review sentences, is fed into CRFs for structure prediction. For each sentence $s_i$, the input for CRF is denoted as $h_i$, where each column $\{h_{i1}, \ldots h_{in_i}\}$ matches to the hidden representation of word $\{w_{i1}, \ldots, w_{in_i}\}$ in the sentence. CRF then computes a sequence of output $y_i = \{y_{i1}, \ldots, y_{in_i}\}$ where $y_{in_j} \in L$. The entire structure of linear-chain CRF can be represented as an undirected graph $G = (V, E)$ with cliques $c \in C$(Sutton et al., 2010). In this project, I employed linear-chain CRF, which has two clique templates: unary cliques($U$), defined at each word as input-output connection, and pairwise cliques($P$), defined at edge connecting neighbors as adjacent output connections, as shown in Figure 1.3. The model aims at outputting $\hat{y}$ with maximum conditional probability $p(y \mid h)$. The probability is computer from potentials of cliques:

$$p(y \mid h) = \frac{1}{Z(h)} \prod_{c \in C} \psi_c(h, y_c),$$

where $Z(h)$ is the normalization term, $\psi_c(h, y_c)$ is potential of clique $c \in C$. Potential of clique $c$ is computed as $\psi_c(h, y_c) = exp < W_c, \phi(h, y_c) >$ , where weight vector $W_c$ is tied for both unary and pairwise cliques and $\phi(h, y_c)$ is a feature function for clique $c$. Hence, potential of clique $c$ is the exponential of a linear combination of feature vectors.

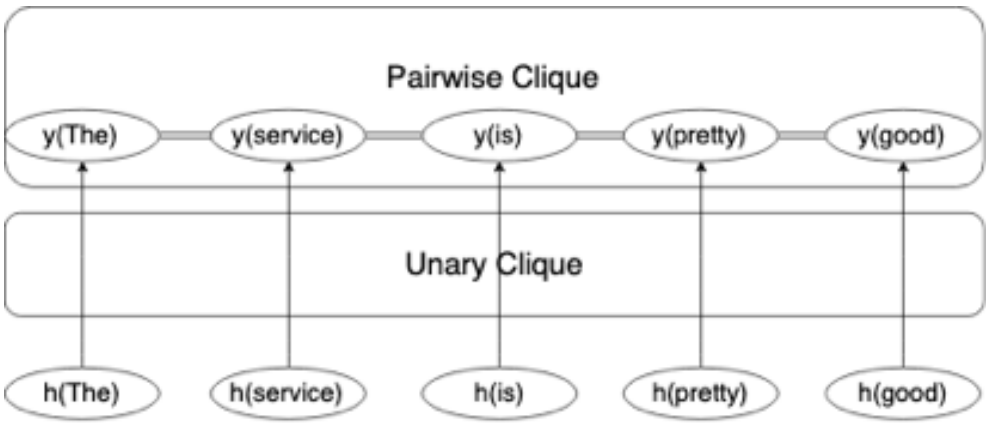


**Figure 1. 3 Example of RNCRF structure**

In this project, I use a context window of size 3 when computing unary potentials. Thus, the potential of unary clique at node $k$ can be written as

$$\psi_U(h, y_k) = exp((W_0)_{y_k} \cdot h_k + (W_{-1})_{y_k} \cdot h_{k-1} + (W_{+1})_{y_k} \cdot h_{k+1})$$

where $W_0$, $W_{-1}$ and $W_{+1}$ are CRF weight matrices for the current position $k$, the first position on the left side, and the first position on the right side. The subscript of weight matrix indicates the row corresponding to $y_k$. Therefore, the conditional distribution of the entire sequence y in Figure 1.3 is calculate as

$$p(y \mid h) = \frac{1}{Z(h)} exp(\sum_{k=1}^{5} (W_0)_{y_k} \cdot h_k + \sum_{k=2}^{5} (W_{-1})_{y_k} \cdot h_{k-1} + \sum_{k=1}^{4} (W_{+1})_{y_k} \cdot h_{k+1} + \sum_{k=1}^{4} V_{y_k, y_{k+1}})$$

where the first terms in the exponential are unary clique. The term $(W_0)_{y_k} \cdot h_k$ is counted five times because each word in the sentence will be taken as current position once through the entire computation. Similarly, only words from second position to the fifth position will take the first position on its left side into account, because the first word in the sentence does not have word on its left side. The last term is the pairwise clique with matrix $V$ representing pairwise state transition score.

## 4.2. Bi-LSTM

Recurrent neural networks (RNNs) has been employed to produce promising results on a variety of NLP tasks. Meanwhile, Long Short-Term Memory (LSTM) architecture, which uses purpose-built memory cells to store information, is better at finding and exploiting long range context.

In aspect-based sentiment analysis, we have access to both past and future input features at a given time. Therefore, we can utilize a bidirectional LSTM network. By doing so, we can process data in both directions within two separate hidden layers for a specific time step. A typical Bi-LSTM structure for sentiment analysis has four layers: input layer *x*, embedding layer *e*, hidden layer *h* and output layer *y*. In this project, we substitute input layer and embedding layer with the output of DT-RNN. Figure 1.4 shows how a Bi-LSTM connects the forward and backward hidden payer and finally update the output layer.

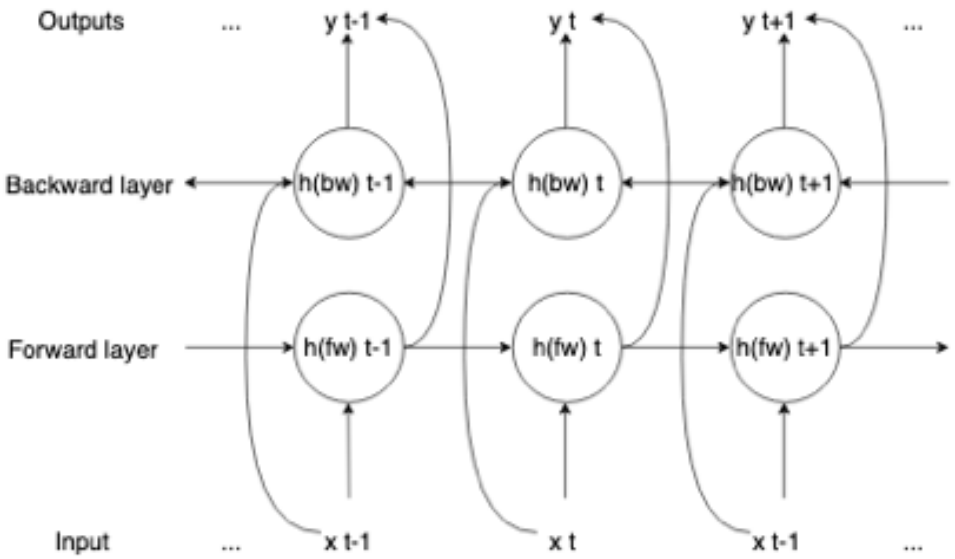


**Figure 1. 4 Illustration of Bi-LSTM**

The output of DT-RNN captures both syntax and semantics information at only word level. By using bidirectional LSTM, we are able to extract contextual representation of each word. The forward hidden sequence $\overrightarrow{h_t}$, backward hidden sequence $\overleftarrow{h_t}$ and output sequence $y$ are computer as follows.

$$\overrightarrow{h_t} = H(W_{x\overrightarrow{h}}x_t + W_{\overrightarrow{h}\overrightarrow{h}}\overrightarrow{h_{t-1}} + b_{\overrightarrow{h}})$$

$$\overleftarrow{h_t} = H(W_{x\overleftarrow{h}} x_t + W_{\overleftarrow{h}\overleftarrow{h}} \overleftarrow{h_{t+1}} + b_{\overleftarrow{h}})$$

$$y_t = W_{\overrightarrow{h}y} \overrightarrow{h_t} + W_{\overleftarrow{h}y} \overleftarrow{h_t} + b_y$$

The outputs of forward-hidden layer and backward-hidden layer are concatenated and then passed to the output layer for classification using *softmax*. Since there are in total 5 classes in L, we take a matrix $W \in R^{5*k}$ and bias $b \in R^5$, where k is the size of hidden units in LSTM. Then we compute a vector of score $s \in R^5 = W \cdot h + b$, where the $i_{th}$ column $s[i]$ can be interpreted as the score of class *i:*

$$P(y_t = i \mid s, \theta) = \frac{exp(w_i^T h_t)}{\sum_{i=1}^{5} exp(w_i^T h_t)}$$

$h_t = \phi(x_t)$ defines the transformations of $x_t$ through the hidden layers, and $w_i$ are the weights from the hidden layer to the output layer. We fit the model by minding the cross entropy of training data.

# 5. Experiment

## 5.1. Dataset

Models are evaluated on the dataset from SemEval Challenge 2014 task 4(subtask 1), including reviews from two domains: restaurant and laptop. There are 3,041 training reviews, 800 testing reviews in the restaurant domain and 3,045 training reviews, 800 testing reviews in the laptop domain. A summary of the SemEval Challenge dataset is shown in Table1. The original dataset only provides annotation labels for aspect terms. After extracting review text and corresponding aspect terms, I use manually annotated opinion terms for both restaurant domain and laptop domain datasets (Wang, Pan, Dahlmeier, & Xiao, 2016).

| Domain | Training | Testing | Total |
|---|---|---|---|
| Restaurant | 3,041 | 800 | 3,841 |
| Laptop | 3,045 | 800 | 3,845 |
| Total | 6,086 | 1,600 | 7,686 |

**Table 1 Overview of SemEval Challenge 2014 Task 4 Dataset**

## 5.2. Word2vec

For word vector initialization, I train word embeddings with word2vec (Mikolov, Chen, Corrado, & Dean, 2013) on the Yelp Challenge dataset for the restaurant domain and on the Amazon electronic domain reviews dataset (He & McAuley, 2016) for the laptop domain. The Yelp dataset contains more than 20M reviews and 674K vocabulary. As for the Amazon electronic domain dataset, it contains 1.7M reviews and 334K vocabulary. Both original datasets are in JSON format, so I need to convert them into CSV format and extract only review text column into txt file while dropping the rest. When passing data to word2vec, it is possible that we face memory issue if the amount of training data is large. Fortunately, we can solve this problem with memory-friendly built-in class “LineSentence” from word2vec as follows,

```
sentences = word2vec.LineSentence(input_file)
model = word2vec.Word2Vec(sentences, size=300,
window=3, min_count = 5)
model.wv.save_word2vec_format(output_file,
binary=False)
```

Note that sentences must be stored in lines in the input file since word2vec is expecting input as tokens. Word2vec might take a huge amount of memory while training. Considering the adequacy of the datasets and the relatively small size of evaluation datasets, I set min-count for word embedding as 5 to increase the efficiency of the entire process.

In this project, I experiment on word embedding of different dimension: the default dimension of word2vec as 100 and dimension as 300 with window size as 3. Based on the experiment result, I choose to use word embedding with dimension as 300 and window size as 3.

## 5.3. Dependency Parser

Besides word embedding, the data pre-processing in this project also includes building dependency trees.

The first step is to parse sentences into raw dependency trees. Dependency results are generated using CoreNLPDependencyParser from nltk, which is using Stanford CoreNLP via REST API. After locally hosting StanfordCoreNLPServer at one of the commonly used ports, we can get real-time parsing result as follows,

```
dep_parser =
CoreNLPDependencyParser(url='http://localhost:9000)
parse, = dep_parser.raw_parse('The service is pretty
good')
```

This is for the further development of the application. By doing so, the application is able to take real-time user input and pass individual sentence to the parser.

After getting the raw parsing result, I then proceed to construct dependency tree objects for each sentence. A dependency tree consists of a list of nodes, each corresponding to one word in the sentence. A node object has several fields: 'word',

'list of parents', 'list of children' and the corresponding index of 'word' in the vocabulary list.

## 5.4. Word Embedding

After obtaining word2vec and dependency trees, I proceed to the task of word embedding. During the word embedding process, each word within a dependency tree is assigned with a word vector. As mentioned in section 5.3, each node object has a field named "index of word". If a node has a valid "index of word", which means that the word assigned to this node has shown in the word2vec file, its word vector will be the corresponding trained word2vec word vector. Otherwise, a vector with 300 random elements will be assigned to the node.

## 5.5. DT-RNN

The pre-training on the parameters of DT-RNN is performed with cross-entropy error. I use mini-batch stochastic gradient descent with a batch size of 25. Experiments are conducted using different learning rate in [0.01, 0.015, 0.02, 0.025] for Adagrad optimizer. Based on the cross-validation result, I take 0.02 as the final learning rate. Eventually, the AdaGrad optimizer is initialized at 0.02 as learning rate, which runs 4 epochs for the restaurant domain and 5 epochs for the laptop domain. All the tuned parameters are saved for sequence labelling.

## 5.6. Conditional Random Fields

In this project, I experiment with two different sequence labelling methods, one is Conditional Random Fields(CRFs), another one is Bi-LSTM.

For CRFs, I utilize a linear-chain CRF with CRFSuite (Okazaki, 2007). I adopt context window size as 3 for both restaurant domain and laptop domain. I use the pre-trained hidden representation of the current word and its two neighbors as the input features to CRFs instead of hand-crafted features. The CRF model performs stochastic gradient descent with l2 regularization term, where the coefficient for l2 regularization is set to 1.

## 5.7. Bi-LSTM

The BRNN with LSTM is built with tensorflow. Based on previous research result (Kingma & Ba), I implement stochastic gradient descent(SGD) with an Adam optimizer with learning rate as 0.001, exponential decay rate for the first moment estimates as 0.9, exponential decay rate for the second moment estimates as 0.999 and epsilon as $10e^{-8}$.

I did grid search for dropout rate and hidden layer dimensions, choosing dropout rate within [0.1, 0.2, 0.3, 0.4, 0.5] and hidden layer dimensions within [50, 100, 150, 200]. The 5-fold cross-validation results for different combinations of hyperparameters are shown as follows:

### 5.7.1. Restaurant Domain

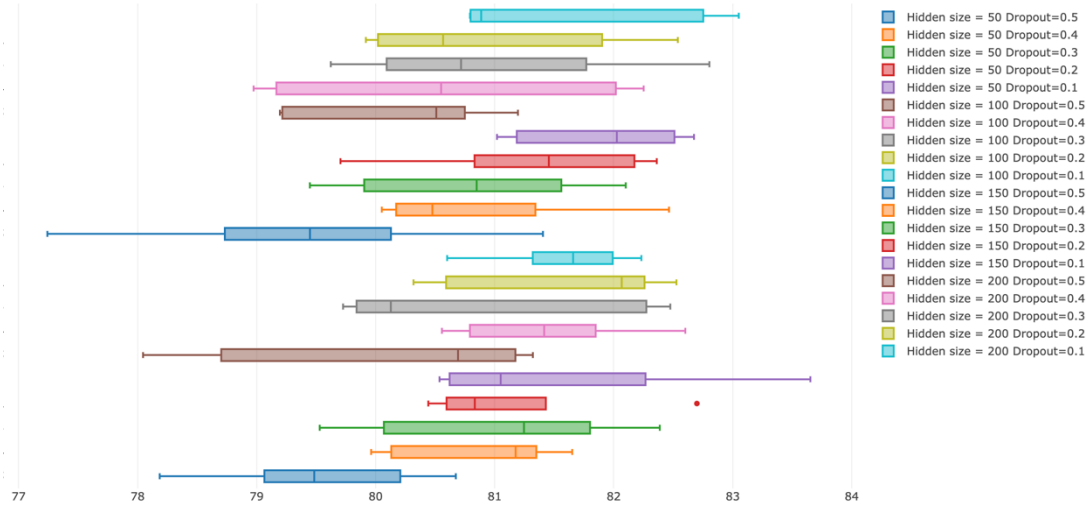


**Figure 2. 1 F-1 Score Distribution for Restaurant Aspect**

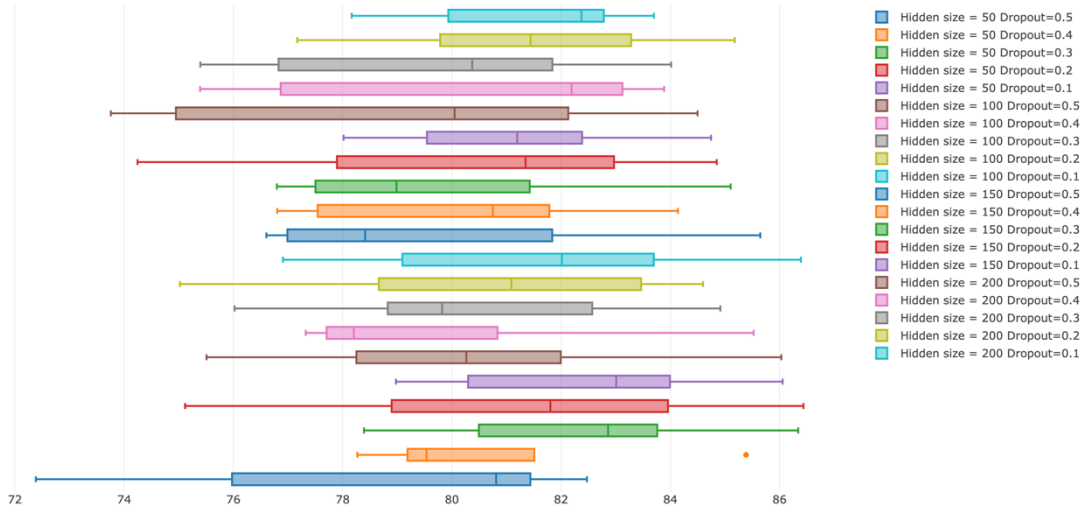


**Figure 2. 2 F-1 Score Distribution for Restaurant Opinion**

Please note that the order of box plot is bottom up but the order of labels on the right-hand side is top down. Taking Figure 2.2 as an example, the blue box at the bottom represents the distribution of cross-validation results for hidden size as 50 and dropout rate as 0.5 in the restaurant domain.

While tuning hyperparameters, I prefer to choose the group of hyperparameters whose average f1 score is relatively high but f1 score variance is relatively low. However, after combining the results from Figure 2.1 and 2.2, I realize it is not possible to have ideal outcomes for both restaurant aspect and restaurant opinion at the same time. For example, when the size of hidden units is 150 and the dropout rate is 0.1, the 5-fold cross validation result has the highest average f1 score and lowest variance for aspect term. But when it comes to opinion terms, the average accuracy drops immediately with increasing variance. Thus, after trading off between the prediction of aspect term and opinion term, I eventually choose 50 as the size of hidden units and 0.1 as the dropout rate for restaurant domain. Although the variance of aspect term prediction of this combination is relatively high, its overall prediction accuracy still makes it the best candidate among all the other choices.

### 5.7.2. Laptop Domain

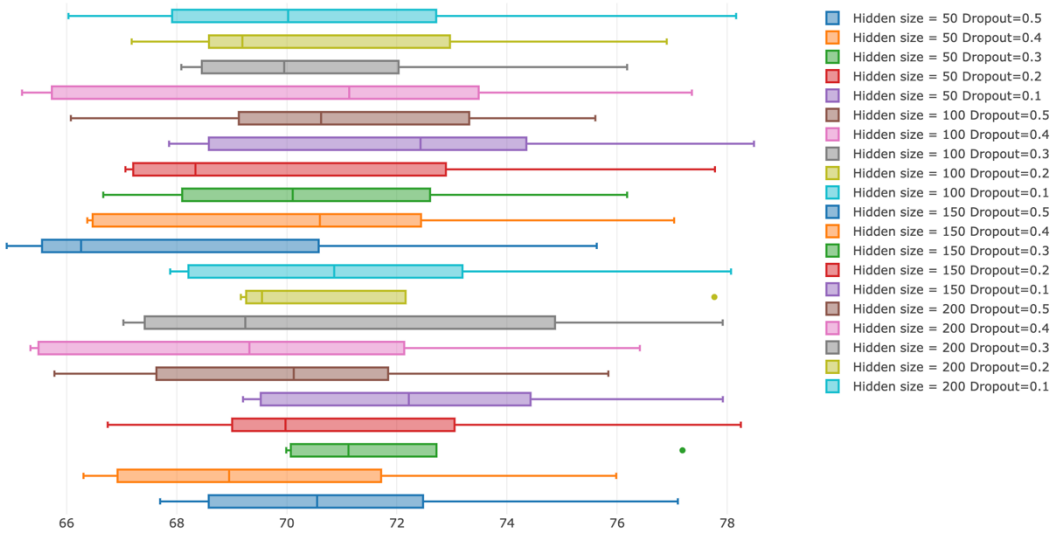


**Figure 2. 3 F-1 Score Distribution for Laptop Aspect**

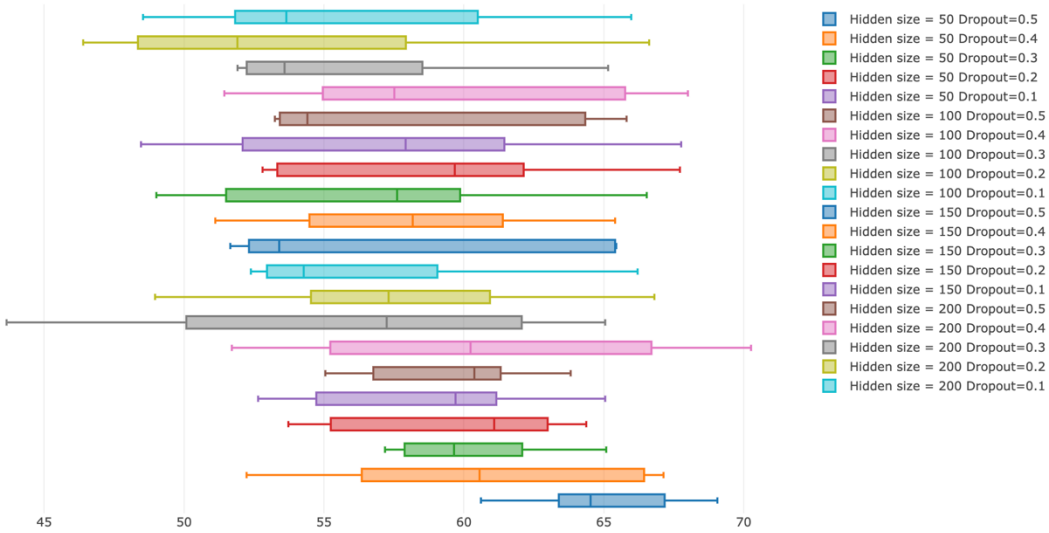


**Figure 2. 4 F-1 Score Distribution for Laptop Opinion**

Figure 2.3 and 2.4 show the f1 score distribution of 5-fold cross-validation in the laptop domain. Same as tuning hyperparameters for restaurant domain, after comparing the average accuracy and variance from the box plots, I choose two sets of parameters: 50 as the size of hidden units and 0.3 as the dropout rate; 50 as the size of hidden units and 0.1 as the dropout rate. Based on the experiment result, I choose 50 as the size of hidden units and 0.3 as the final dropout rate for the laptop domain.

## 5.8. Evaluate Schema

When evaluating the performance of fine-grained sentiment analysis models, we always look for one beginning-of-aspect(BA) item followed by zero or more inside-of-aspect(IA) items, which is the same when it comes to opinion terms. In this project, I use evaluation metrics which score if a model is able to identify the exact span of an aspect or opinion term and if it is able to assign the correct entity type(aspect or opinion) regardless of the boundaries. In order to calculate precision and recall, I use the scoring categories proposed by MUC (Chinchor, 1992):

- COR: the model's prediction and the golden-standard annotation agree
- INC: the model's prediction and the gold-standard annotation disagree
- PAR: the model's prediction and the gold-standard annotation are not identical but have some overlapping text.
- MIS: there is a gold standard entity that is not predicted by the model
- SPU: the model predicts an entity that does not exist in the gold-standard

For both the boundaries and the type, the following measures are calculated:

- The number of correct answers: *COR*
- The number of annotations in the gold-standard which contribute to the final score:

  $Possible(POS) = COR + INC + PAR + MIS = TP + FN$
- The total number of annotations produced by the system:

  $Actual(ACT) = COR + INC + PAR + SPU = TP + FP$

Evaluation results will be reported using the standard precision/recall/f-score metrics:

- $P(Precision) = \frac{COR + 0.5 * PAR}{ACT}$
- $R(Recall) = \frac{COR + 0.5 * PAR}{POS}$
- $F1 = \frac{2 * P * R}{P + R}$

## 5.9. Experiment Results

The comparison results are shown as follow,

| | Restaurant | | Laptop | |
|---|---|---|---|---|
| **Models** | **Aspect** | **Opinion** | **Aspect** | **Opinion** |
| RNCRF(baseline) | 84.05 | 80.93 | 76.83 | 76.76 |
| LSTM(baseline) | 81.15 | 80.22 | 72.73 | 74.98 |
| RNCRF | **84.14** | 82.56 | **77.04** | **80.17** |
| Bi-LSTM | 81.49 | **83.21** | 73.04 | 68.86 |

**Table 2 Comparison results in terms of F1 scores**

Here I use results of RNCRF and LSTM from Wang et al. (2016) as the baselines.

From Table 2 we can see that RNCRF is promising for fine-grained sentiment analysis task. It outperforms its baseline RNCRF model with 2%, 4.44% absolute improvement for opinion extraction in the restaurant domain and laptop domain, respectively. The aspect extraction results in both domains are also comparable to the baseline RNCRF model. The improvement is caused by the following reasons. First, the semantic parser has evolved. The original RNCRF is using parser version 3.5.1, but the latest parser has become version 3.8.1. The two versions of parser generate different parsing for the same sentence where both the dependency relation and edge construction differ. New dependency relations are introduced in the latest version of semantic parser, which brings more information and helps to make prediction. Second, I utilize the concept of context window in word embedding which encodes contextual information at the beginning of the prediction task. In the original implementation of RNCRF, context window is not considered until word vectors are fed to CRFs for sequence labelling. However, I use context window size of 3 in word embedding, which can be taken as pre-training word vector using the context of the training document.

As mentioned in section 2, the performance of LSTM heavily depends on word embedding. I use self-produced word embedding rather than well-trained word embeddings using large corpus or extensive external resources. But since bidirectional LSTM is used, information is dually propagated between aspect and

opinion terms. The performance of Bi-LSTM is comparable to the baseline model for aspect extraction in both domains and is even better with 3.7% improvement for opinion extraction in the restaurant domain. By incorporating dependency information and dual propagation information between aspect and opinion terms, Bi-LSTM can still well learn the hidden representation without conducting sophisticated word embedding.

Comparing these two models, I also find that RNCRF is able to provide a promising prediction result even with a small dataset. For example, when I use a sample dataset of only 100 restaurant review sentences, RNCRF is still able to achieve f1 scores above 70 for both aspect and opinion extraction. However, Bi-LSTM can only make f1 score around 40 for aspect extraction and less than 20 for opinion extraction. Nevertheless, Bi-LSTM still has its own strength, which is the training speed. The entire learning process of RNCRF may take hours, while Bi-LSTM only takes minutes. Moreover, the performance of Bi-LSTM can have big improvement with larger training dataset, which makes it a good candidate for the prediction of a large dataset.

In conclusion, RNCRF is a robust model with promising performance regardless of the size of the dataset. Bi-LSTM, on the other hand, achieves slightly lower accuracy in this project.

# 6. Application

I use Flask to build a sentiment extractor interface. Flask is a micro web framework written in Python. Since the models are implemented with Python, it is convenient to wrap them into a Python-based framework. This application is built based on RNCRF model in the restaurant domain. Experiments are conducted first, then model parameters which achieves the best performance are dumped for the use of UI development.

## 6.1. Front-End

I made two types of UI, one to take text input, another one to take URL input. They are shown as follows.

The front-end takes user request and renders HTML formatted string based on the labelling result produced by the back-end. Front-end is in charge of data processing. If the URL is passed, it will first extract paragraphs from a given URL using urllib and BeautifulSoup (RealPython).

```
try:
     url = request.form['url']
     sauce = urllib.request.urlopen(url).read()
except:
     # handle exception
if url:
     soup = bs.BeautifulSoup(sauce, 'lxml')
     sentences = []
     text = []
     for paragraph in soup.find_all('p'):
          if paragraph.text[0] not in whitespace:
               text.append(paragraph.text.strip())
     # split text into sentences and save in a list
```

The text is then converted into a list of sentences and passed to back-end for sequence labelling, which is the same as directly passing text into the application. In the generated string, aspect terms are highlighted in orange while opinion terms are highlighted in purple. The highlighting is implemented using HTML tags. Take

aspect term highlighting as an example: if a word is labelled as aspect, additional tags will be added to it:

```
<strong><span style='color:orange'>
     Aspect-word
</span></strong>
```

The reason why I want to build two UI is that customer review from a website are not as 'cleaned' as our training data, the way of expression varies and can have implicit aspect and opinion terms, which are hard for the model to extract. Therefore, it is easier to test the model by inputting text.

The illustration of text UI and URL UI are shown as follows,

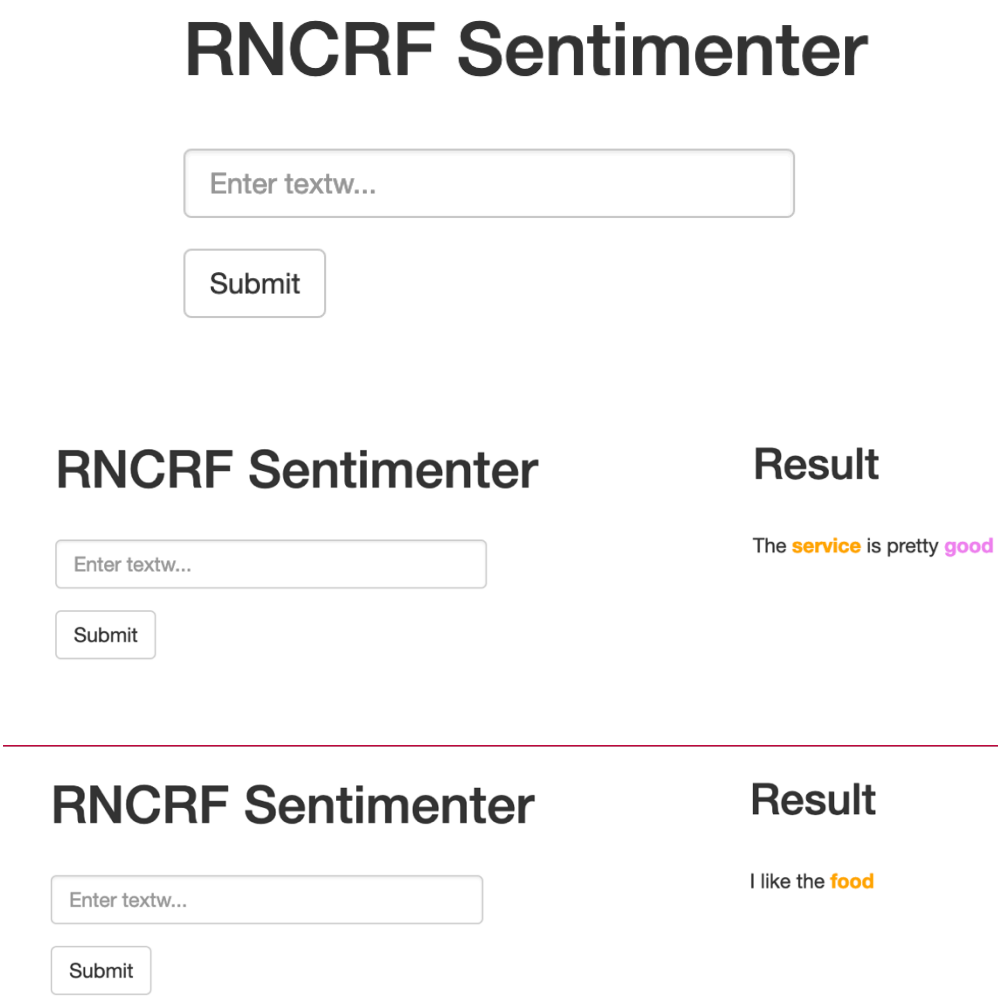


**Figure3. 1 Text Input UI**

Since this application is built for restaurant-domain, I take one restaurant review page from Yelp for demonstration (FatPapas).

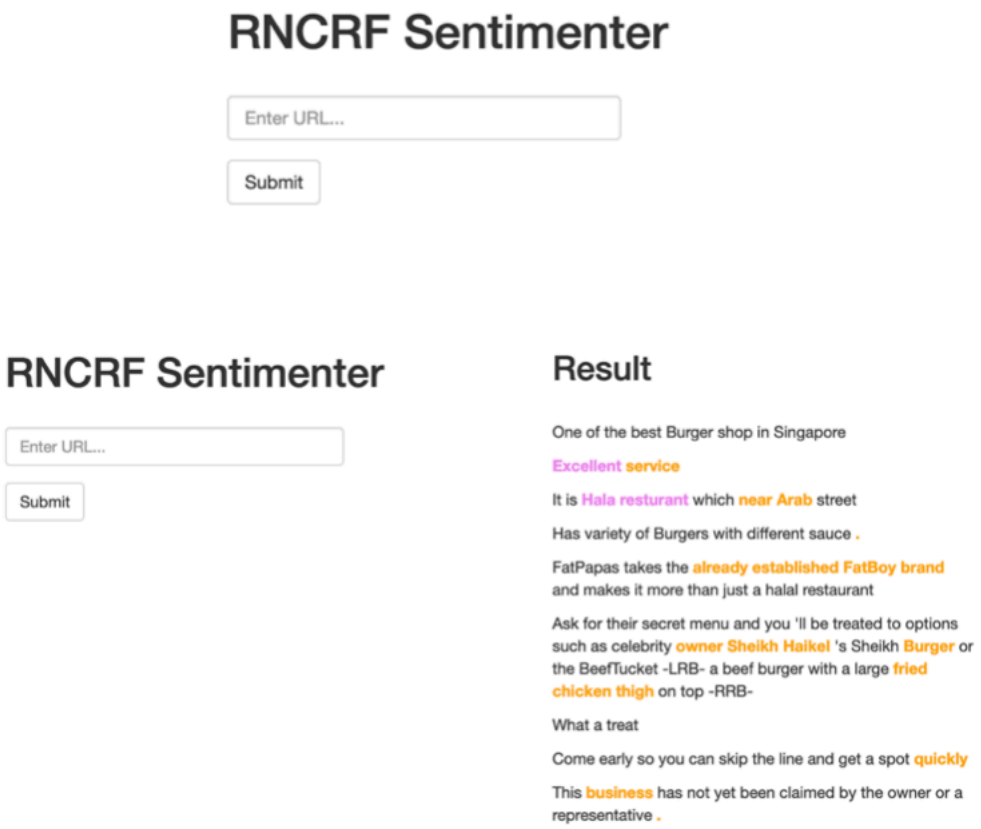


**Figure3. 2 URL Input UI**

## 6.2. Back-End

The backend is in charge of sequence labelling.

When the app is started, a series of preparation is done before the user input any data. First, tuned parameters and trained model are loaded, word embedding dictionary is also generated from the word2vec file. Next, a CoreNLP parser instance is created and hosted locally at port 9000.

All of the above objects are then used to instantiate a ‘Sentimenter’ object. The ‘Sentimenter’ object takes a pre-trained CRF model, parameters, and sentences for labelling from front-end. It passes each sentence to the parser instance and builds a dependency tree based on the parsing result. For each dependency tree, Sentimentor

gets the hidden representation of each node by forward propagating information from children to its parent. The final input of each sentence for labelling is a list of word features, where each word is represented by a dictionary of features. I use a context window size of 3, which is the same as the pre-trained CRF model.

The entire workflow is shown in Figure 3.3.

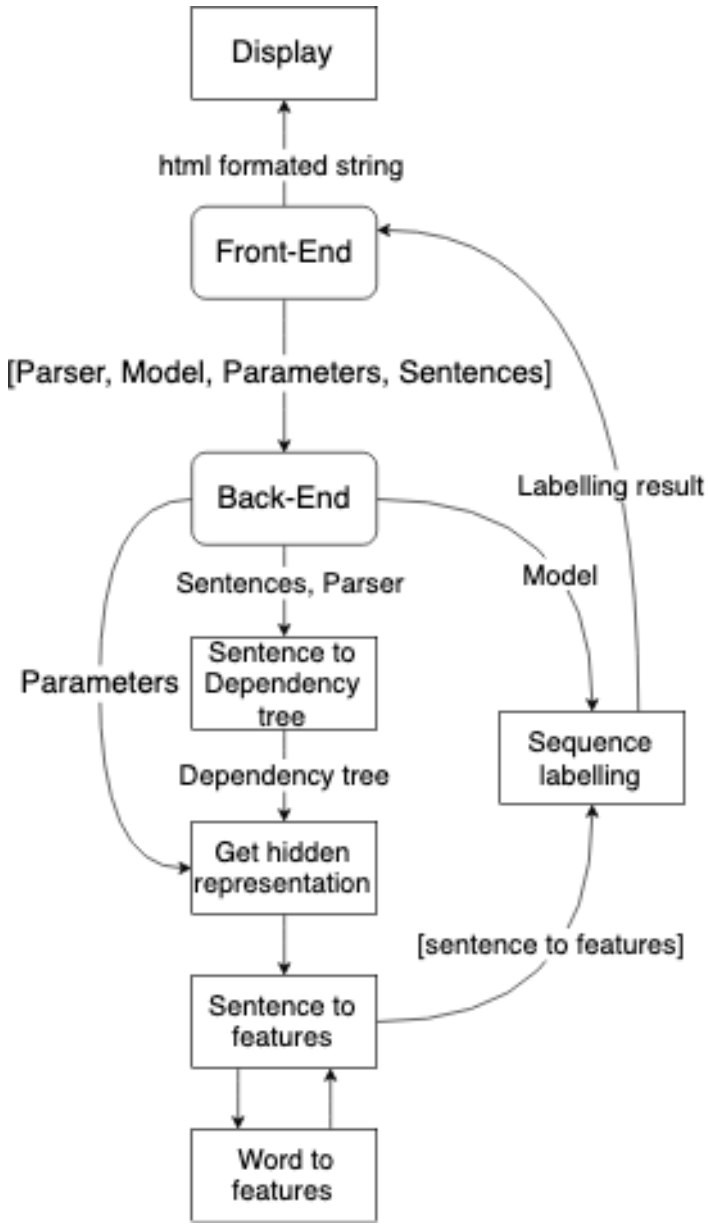


**Figure3. 3 Illustration of application word flow**

The sentiment analysis is handled by Sentimentor object and therefore it is convenient to upgrade the application with a better model. For instance, if another RNCRF model is proved to have better performance than the current RNCRF model,

I can simply replace the parameter files and load the better model. It is also possible to build Sentimenter based on other methods. In this project, I also build a Bi-LSTM sentiment extractor with tensorflow. By importing new dependencies and restoring tensorflow model, I can also use Bi-LSTM for labelling in the Sentimenter instance.

# 7. Conclusion

## 7.1. Conclusion

An entire process of fine-grained sentiment analysis consists of several steps, including data collection, data cleaning, data pre-processing, word embedding, pre-training, and model evaluation. In this project, I use review datasets from the restaurant domain and laptop domain to do a series of experiments and eventually come up with a sentiment extraction application. The experiment result has shown that encoding dependency paths can overcome the shortage of word embedding. The joint-model combines the advantages of RNN and CRF, thus learns syntactic, semantic and contextual information from a user-generated sentence. The text UI and URL UI are developed to demonstrate the result of the tuned model. Text UI allows us to play with it while URL UI provides a more generalized application for the analysis model.

## 7.2. Future Works

The models and application built in this project still have some aspects to improve. First, this project is conducted in single-domain, only the restaurant domain and laptop domain reviews are used for training and testing. The model can be extended to cross-domain in the future. Second, the Bi-LSTM model in this project has only one layer while RNNs always have the potential to go “deep”. Therefore, future work can be conducted to see if deep Bi-LSTM will improve the performance of Bi-LSTM model. Third, the application is only a prototype at this moment, especially the URL application. Further improvement can be made to enhance the text extraction performance. Last but not least, more information is in fact available in the parsing result, for example, POS tags. Further experiments can be conducted using POS tags as extra features to see if performance can be improved.

# 8. Material Resources

In this section, I will give a brief description of the software resources used in the project for those who might follow and carry out the same project in the future.

- Programming Language:
  - Python==3.6.8 :: Anaconda, Inc.
- Development Environment
  - Jupyter notebook==5.7.4
- Toolkit
  - Stanford CoreNLP==3.9.2
- Python Dependencies
  - genism.word2vec for word embedding training
  - numpy==1.16.2
  - tensorflow==1.12.0
  - sklearn_crfsuite==0.3
  - joblib==0.13.2 for dumping and loading CRF Suite
  - Flask==0.10.1
  - beautifulsoup4==4.7.1
  - urllib3==1.24.1 for accessing and reading from given URL
- Cloud Platform

  It will be useful and save plenty of time if we can use cloud services to run models, especially when our own devices are not capable to conduct large volume experiments such as K-fold cross-validation. There are many options available on the Internet. In this project, I tried Google Cloud Platform and Kaggle and would like to recommend Kaggle for those conducting experiments with Gigabytes of data.

# Works Cited


Li, Y., Pan, Q., Yang, T., Wang, S., Tang, J., & Cambria, E. (2017). *Learning Word Representations for Sentiment Analysis. Cogn Comput.* Retrieved from https://doi.org/10.1007/s12559-017-9492-2

Ma, Y., Peng, H., & Cambria, E. (2018). Targeted Aspect-Based Sentiment Analysis via Embedding Commonsense Knowledge into an Attentive LSTM. *AAAI Conference on Artificial Intelligence.* North America.

Liu, P., Joty, S., & Meng, H. (2015). Fine-grained Opinion Mining with Recurrent Neural Networks and Word Embeddings. *EMNLP*, (pp. 1433-1443).

Wang, W., Pan, S., Dahlmeier, D., & Xiao, X. (2016). Recursive Neural Conditional Random Fields for Aspect-based Sentiment Analysis. *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, 616-626.

Qiu, G., Liu, B., Bu, J., & Chen, C. (2011). Opinion word expansion and target extraction through double propagation. *Computational Linguistics*, 37(1):9-27.

Jin, W., & Ho, H. (2019). A model lexicalized hmm-based learning framework for web opinion mining. *ICML*, 465-472.

Huang, Z., Xu, W., & Yu, K. (2015). *Bidirectional LSTM-CRF Models for Sequence Tagging.* Retrieved from arXiv.org e-Print archive: arXiv: 1508.01991

Graves, A., Mohamed, A.-r., & Hinton, G. (22 Mar, 2013). *Speech Recognition with Deep Recurrent Neural Networks.* Retrieved from arXiv.org e-Print archive: Cite as: arXiv:1303.5778

*Stanford CoreNLP – Natural language software*. (n.d.). Retrieved from Stanford CoreNLP: https://stanfordnlp.github.io/CoreNLP/index.html

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781.

He, R., & McAuley, J. (2016). Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering. *WWW*.

Kingma, P. D., & Ba, J. (n.d.). *Adam: A Method for Stochastic Optimization.* Retrieved from arXiv.org e-Print archive: arXiv: 1412.6980

Okazaki, N. (2007). *CRFSuite: a fast implementation of conditional random fields (CRFs).* Retrieved from http://www.chokkan.org/software/crfsuite/

*FatPapas.* (n.d.). Retrieved from Yelp: https://www.yelp.com/biz/fatpapas-singapore?osq=fatpapa

RealPython. (n.d.). *Flask by Example – Text Processing with Requests, BeautifulSoup, and NLTK.* Retrieved from RealPython: https://realpython.com/flask-by-example-part-3-text-processing-with-requests-beautifulsoup-nltk/

Goller, C., & Küchler, A. (1996). Learning task-dependent distributed representations by back-propagation through structure. *ICNN*, (pp. 347-352).

Chinchor, N. (1992). MUC-4 evaluation metrics. *MUC4 '92 Proceedings of the 4th conference on Message understanding* (pp. 22-29 ). Virginia: McLean.

Poria, S., Cambria, E., Winterstein, G., & Huang, G.-B. (6 June, 2014). *Sentic patterns: Dependency-based rules for concept-level sentiment analysis.* Retrieved from Knowledge-Bases Systems: www.elsevier.com/locate/knosys