NORTHWESTERN UNIVERSITY

# Scaling Classroom Education with Peer Review: A Natural Language Processing Approach

A DISSERTATION

SUBMITTED TO THE GRADUATE SCHOOL

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

for the degree

DOCTOR OF PHILOSOPHY

Field of Computer Science

By

Zheng Yuan

EVANSTON, ILLINOIS

June 2020

# ABSTRACT

Scaling Classroom Education with Peer Review: A Natural Language Processing Approach

Zheng Yuan

Peer review is a commonly used tool to manage large classes. It allows students to grade and provide feedback to each other based on rubrics provided by instructors. Peer review has been proved to be effective in improving students' learning outcomes by many research. During providing peer review, students are exposed to more solutions to the same question, which makes students more innovative. Also, peer review reduces instructors' or TAs' workload on providing feedback despite teaching big classes.

However, peer review still has shortcomings. First, peer review increases students' workload on reviewing other submissions. To collect enough peer reviews for submissions, each peer usually needs to review several submissions per homework. Some peers may spend more time on peer reviewing than finishing the homework. Second, since peers typically lack the subject matter mastery of the instructor, peer grades exhibit both bias and variance, which makes consensus grade estimation a challenging task.

This dissertation addresses these limitations of peer review using Natural Language Processing techniques. Specifically, this dissertation proposes novel neural models that predict important parts of submissions. By doing so, peers can save time by only focusing on the essential parts. Our models take advantage of textual reviews and review labels to improve prediction accuracy. This dissertation also proposes novel peer grading methods, which enhance peer grading accuracy by using historical instructor grades to estimate peer bias, and textual review comments to estimate review quality.

## Acknowledgements

The Lord is my shepherd; I shall not want. (Psalm 23) Give thanks to the Holy one, for his salvation and unchanging love.

I want to thank my wife, Jing Yang, for her fully support in my life. She always cooks delicious food. In our marriage, I learned how to love people.

Special thanks to my parents for raising me. Without their support, I could never have the opportunity to study in the US.

I want to express my sincere gratitude to my advisor Prof. Doug Downey. He always encourages me when I am frustrated and guides me when projects did not go well. Also, I greatly appreciate his invaluable and thought-provoking training in NLP and machine learning. This dissertation could not have been finished without his help and guidance.

Meanwhile, I also thank my dissertation committees, Prof. Jason Hartline and Prof. Eleanor O'Rourke. Our weekly meeting was inspirational and helpful. Prof. Hartline has crystal clear thoughts and pointed out a lot of valuable experiments and research questions. Prof. O'Rourke is very kind and had inspired me to think more about the learning outcomes of peer review.

I am grateful for the contributions of my lab mates, David Demeter, Mike D'Arcy, Rony Mohammed Alam, Yiben Yang, and Thanapon Noraset. Nor is one of my best friends. We also worked out together and discussed a lot of research questions.

# Table of Contents

CHAPTER 1

# Introduction

Peer review [17] is a widely used tool in classrooms, in which students are asked to review each other's submissions, and reviews are aggregated to produce consensus assessments of each submission. Due to the explosion enrollment in computer science major[56], it is hard for instructors or TAs to grade and provide feedback for every homework submission. Peer review is a common solution to scaling the size of classroom education. Peer review allows students to grade and give feedback to each other's submission. It has several advantages, including reducing instructors' or TAs' workload on grading[41], improving students' writing ability[97], providing helpful guidance to students[18], and enabling more prompt feedback to students on their submissions [60].

A general peer review process in a classroom setting includes the following steps:

**1:** Submission collection: students submit individual or group submissions.

**2:** Review matching: determine which peers should review which submissions.

**3:** Peer feedback collection: Peers provide their numeric grades and textual comments according to certain rubrics.

**4:** Consensus grade estimation: compute submission grades based on peer grades.

**5:** Instructor evaluation (optional): To ensure the quality of peer feedback, sometimes instructors grade a portion of the submissions, and also the review comments. Peers are rewarded for assigning grades that are similar to the instructor's grade, and for providing helpful comments.

However, challenges in peer review exist [1]. First, although peer review reduces instructors' or TAs' workload, the workload of peers increased because they need to review others' submissions. Second, since peers are not well-trained graders, peer grades are not as accurate as instructors' or TAs' grades. (We use the term of instructor and TA interchangeably.) In this dissertation, we propose novel NLP (Natural Language Processing) solutions to the two shortcomings mentioned above.

### 1.1. Highlighted Text Prediction and Named Entity Typing in Open Domain

Text annotation [113] is the practice and the result of adding a note or gloss to a text, which may include highlights or underlining, comments, footnotes, tags, and links. It helps people better understand an article [72, 87]. Doing text annotation by machine (automatic text annotation) is an essential task in Natural Language Processing. Automatic text annotation includes many tasks, such as text summarization [71, 27, 29, 36, 9, 59, 54, 101, 19, 55], review generation [65, 8, 79, 25, 80, 130, 116, 91, 127, 134], important sentence prediction [135, 48, 63], to name a few. It is valuable for many other natural language processing tasks, such as relation extraction [69, 129, 78, 131, 66, 50, 52, 88, 70, 133], question answering [122, 11, 30, 89, 110, 119, 125, 111, 132, 112], knowledge base population [51, 4, 15, 32, 35, 38, 49, 74, 76, 86], and co-reference resolution [96, 31, 22, 26, 23, 83, 13, 20, 43, 120]. Automatic text annotation also helps people reading articles easier

[117]. For example, by highlighting sentences people can easily get the gist of articles. By annotating entities and drawing relationships, people can better understand connections within articles.

One of the tasks in this dissertation aims to reduce peers' workload by highlighting important text of submissions, so that peers can save reading time by skipping un-highlighted parts. We collect submissions and peer review data in EECS-349 machine learning class at Northwestern University. The task we are interested in is to predict the important parts of the submissions we collected. This problem can be viewed as a sequence labelling (tagging) problem [103] where the model predicts a binary label (highlighted or not) for each token. Since we have collected textual reviews and review labels corresponds to each highlighted sentence during peer review process using PDF annotation tools, we can take advantage of these extra information to improve prediction accuracy. Textual reviews can be viewed as labels for highlighted sentences. To investigate the problem of treat a text sequence as a label, we first study the problem of NET (Named Entity Typing) in open domain setting [126].

NET [77] is the task of labeling a given entity mention in text with a semantic label. NET is valuable for many natural language processing tasks, such as relation extraction, question answering, knowledge base population, and co-reference resolution. Traditional NET systems [115, 12, 2, 16] usually pre-define a set of labels and train a model to learn entity typing. One of the shortcomings of NET is that it is hard to incorporate new labels to the already trained NET models. The task of Open Named Entity Typing (ONET) is fine-grained NET when the set of target types is not known in advance. Note that the common part between ONET and highlighted text prediction is that both of them view

unseen texts as labels. In ONET, the unseen texts are type names, while in highlighted text prediction unseen texts are peer reviews. Thus, by investigating ONET problem, we could get more insights of highlighted text prediction.

## 1.2. Consensus Grade Estimation

How to estimate consensus grade accurately based on peer grades is a challenging problem in peer review [1]. Since peers typically lack the subject matter mastery of the instructor, peer grades exhibit both bias and variance, which makes consensus grade estimation a challenging task. A variety of previous work [3, 85, 94, 93, 6, 7] has proposed peer grading methods to model peer biases and variances.

However, existing methods have two limitations. First, they do not model systematic peer bias. That is, most peers tend to all overestimate, or all underestimate, then the consensus grades computed by the methods will be higher or lower than the ground truth grades. Second, existing methods only consider peer grades and are not able to take advantage of textual review comments. In addition to numeric grades, peers provide textual comments that point out problems in submissions or suggestions for improvement. Comments that are high-quality could indicate that a peer review merits higher weight in the consensus grade. In this dissertation, we explore several methods that can not only detect and adjust systematic peer bias based on historical data but also improves consensus grade estimation accuracy using available textual reviews.

## 1.3. Contributions and Outline

Contributions of this dissertation are as follows.

**Preliminary Work: Open Domain Named Entity Typing:** In Chapter 2 and [126], we introduce Open domain Named Entity Typing (ONET), the task of typing named entity in open domain setting. In ONET, the set of target types is not known in advance. We propose a novel neural architecture for ONET. We show that the proposed neural architecture outperforms two baseline models and achieves weighted average ROC_AUC score of 0.870 and 0.780 on unseen types for FIGER(GOLD) and MSH-WSD data. We also analyze which factors drive the system's performance, finding that the presence of training types that are more similar to an unseen target type improves accuracy on that type.

**Highlighted Text Prediction:** In Chapter 3, we present several neural architectures that predict highlighted text in homework submission PDFs. Our methods are inspired by the over-labeling technique [75], which takes advantage of available information, textual reviews and review labels, to improve prediction performance. Our results show that by using review labels our model beats baseline model while using textual review does not have much advantage on highlighted text prediction. By doing highlighted text prediction, peers can potentially save time on peer review process by focusing on important parts of submissions and skim the unrelated parts.

**Consensus Grade Estimation:** In Chapter 4, we explore different methods to improve consensus grades estimation accuracy. We first investigate one of state-of-the-art peer grading algorithms and propose several ways to improve its performance. We also present peer grading methods, which improve peer grading accuracy by using historical instructor grades to estimate peer bias and textual review comments to estimate review

quality. Our method models peer biases in maximum-likelihood fashion using the differences between peer grades and instructor grades in the past. It models the quality of a peer review using its textual comments and puts more weight on better peer reviews when computing submission grades. To the best of our knowledge, our method is the first work to explore using textual reviews to improve peer grading performance. We also show the possibility of further improves the estimation performance using co-training method. In addition, Chapter 4 also investigates the learning outcomes of different peer review processes. This chapter releases the limitation of inaccurate peer grades on peer review tools.

CHAPTER 2

# Preliminary Work: Open Domain Named Entity Typing

This chapter is a preliminary work chapter. In order to see how likely highlighted text prediction is going to work with the help of textual reviews, we first tackle a simpler task: named entity typing in open domain. Entity typing labels a mention in a sentence with types. In open domain setting, type names can be any words or phrases, which may not even appear in training phase. As mentioned before, the connection between open domain named entity typing and highlighted text prediction is that both of them treat unseen texts as labels. This chapter treats entity types, usually one word, as labels, while next chapter treats textual reviews as labels.

In this chapter, section 2.1 introduces the task of ONET. In section 2.2, we cover related work. 2.3 formally defines ONET problem. Section 2.4 illustrates a neural network architecture for ONET problem. Section 2.5 shows our results. Section 2.6 concludes this chapter.

## 2.1. Introduction of ONET

Named Entity Typing (NET) is the task of labeling a given entity mention in text with a semantic label. NET is valuable for many natural language processing tasks, such as relation extraction, question answering, knowledge base population, and co-reference resolution.

Traditional NET focused on a small set of mutually-exclusive types, such as person, location, and organization [115, 12, 2, 16]. More recent work has generalized NET to much larger type systems that include fine-grained types, e.g. book, artist, city, and so on [90, 106, 67, 124]. In fine-grained NET, often the target types are non-disjoint ("Paris" is both a city and a location, for example) and thus fine-grained NET is a multi-label task, i.e. each mention may be assigned multiple positive labels.

Existing fine-grained NET techniques have a limitation: they require labeled training mentions for each target type. For types that are not in the training set—referred to as *unseen* types—NET systems cannot output the labels. This is a limitation because in many cases, we want to know whether a given entity belongs to a specific type that is *not* present in our limited training dataset. Learning whether an entity belongs to an unseen type is an instance of "zero-shot learning," where no training examples exist for a given output label [82, 108, 14]. While existing hypernym discovery techniques can identify whether a given phrase belongs to a type, hypernym discovery is context insensitive, whereas NET is context sensitive. For example, given the sentence "I went to Chicago last year," hypernym discovery would output all possible types of "Chicago" (city, band, movie, etc.). Whereas in NET we must only assign types that are correct for "Chicago" in the given context, e.g. city and location. Unsupervised Named Entity Recognition [45] is context-sensitive and in principle can handle entities of unseen types, but these techniques require mutually exclusive entity categories—i.e. they cannot apply to the multi-label setting encountered in fine-grained NET.

Thus, we introduce the task of Open Named Entity Typing (ONET), which is fine-grained NET when the set of target types is not known in advance. We propose a novel

neural architecture for ONET. The intuition behind our approach is simple: we represent mentions and types in our model using word embeddings pre-trained on unlabeled text, and leverage regularities in the embedding space to extend to unseen types. Specifically, our model projects the embeddings of mentions and types into a common space. We train the model such that in the common space, the representations of correct types are close to the mention representation, while the representations for incorrect types are far away. Our hypothesis is that regularities in the pre-trained embedding space will allow this approach to extend to unseen types. That is, if our model learns to map mentions of the type "musician" to be close to the representation for the word "musician," it will also map mentions of the type "drummer" to be close to the representation of the word "drummer," even if the latter type never appears in training. Our architecture extends previous work [106] by adding a new component, type embeddings, and a method for comparing the type embedding to a given mention embedding. In addition, our model is able to incorporate mention- and pattern-based features as optional components.

We refer to our model as OTyper. We evaluate OTyper on a common benchmark NET dataset, FIGER(GOLD) [67] and MSH-WSD dataset[53]. The results show that OTyper outperforms two baseline models and achieves weighted average ROC_AUC score of 0.870 and 0.780 on unseen types for FIGER(GOLD) and MSH-WSD. We also analyze which factors drive the system's performance, finding that the presence of training types that are more similar to an unseen target type improves accuracy on that type.

In summary, our contributions include:

**1:** We introduce a new task of Open Named Entity Typing (ONET), which is fine-grained NET when the set of target types is not known in advance.

**2:** We propose a neural network model, OTyper, for the ONET task and experimentally demonstrate that the technique outperforms baseline approaches on unseen types.

**3:** We also establish performance baselines for the ONET task. To the best of our knowledge, OTyper is the first model for ONET.

## 2.2. Related Work

Named Entity Typing is a long-standing task in Natural Language Processing [34, 21, 28, 95, 98, 61]. Most work in NET is performed in the context of the *Named Entity Recognition* (NER) task, which includes NET as a subtask. In NER, systems must first find and delimit entity mentions, and then perform NET, i.e. assign each mention to a type. Using a variety of supervised approaches, existing NET methods can achieve high accuracy and recall. Most of these systems only deal with traditional NET over three categories: person, location, organization.

Many end tasks such as relation extraction and question answering can benefit from finer-grained entity typing, which has become a focus in recent years. Unlike traditional NET, fine-grained NET considers hundreds or thousands of types, and each entity mention can be assigned to more than one type. Ling and Weld [67] introduced 113 entity types derived from Freebase and released a dataset, FIGER, now a commonly-used benchmark for fine-grained NET. They used semantic features to train a multi-instance multi-label distant supervision classifier on FIGER. In this chapter, we use FIGER as our evaluation dataset. Lee et al. [64] defined 147 fine-grained named entity types and used a CRF on fine grained NET for question answering. Yogatama et al. [124] introduced embedding

methods that use a ranking loss and learn a joint representation of features and labels, which allows for information sharing among related labels. Rabinovich et al. [90] considered a large number of types and applied a linear model for fine-grained NET. Shimaoka et al. [105] proposed the first model for fine-grained entity typing that learns to recursively compose representations for the context of each entity using an attention model. Shimaoka et al. [106] combined the attention model with mention features and hierarchical label-sharing parameters in a NET system, and achieved the current state-of-the-art result on the FIGER(GOLD) dataset. By adopting a universal schema approach, Yao et al. [123] operates over the union of all types from its input sources, and is able to classify over 16,000 types. However, all these methods assume a pre-defined a set of fine-grained types that is known at training time—they are not applicable to ONET, where the target types do not appear in the training dataset.

Huang et al. [45, 44] proposed an unsupervised entity-typing framework by combining symbolic and distributional semantics. They use domain knowledge bases as an additional data resource. Their approach creates a knowledge graph and knowledge representation based on domain knowledge base, and then clusters entity embeddings and named entities. Their system produces only one type label for each mention, while in ONET each mention can correspond to many types. Also, domain knowledge bases are not always available, which is a limitation for their system. By contrast our OTyper approach does not use a knowledge base, only pre-trained embeddings.

Similar to named entity typing, hypernym discovery [107, 102] also labels entities with their hypernyms. Hypernym discovery is the task of, given an NP $e$, finding a set of NPs $c_i$ such that each $c_i$ is a hypernym of $e$ [99]. Hypernym discovery is often powered by

Hearst patterns [39], which we also employ as features in OTyper. The primary difference between our ONET task and hypernym discovery is that ONET is context sensitive—we must not only assign types to each entity name, but also determine which of those types are relevant to the particular sense of the entity name used in a given context.

## 2.3. ONET Definition

We now formally define Open Named Entity Typing (ONET). As in traditional NET, in ONET we take as input a set of *mentions*. Each mention is an occurrence of a named entity in text, along with its surrounding context. We are also given a set of target types, and the the task is to associate each mention with its correct types. An ONET system is trained on a labeled dataset of annotated mentions. The key difference between ONET and NET is that in ONET, some of the target types (i.e. the test types) may not occur at all in the training phrase. We use the symbol $t_i$ to denote a type in training or testing, and $m_i$ to denote a mention. Each mention $m_i$ contains two parts: the entity $e_i$ and its textual context on the left and right, referred to as $cl_i$ and $cr_i$. In our remaining notation, we use lowercase letters for scalars, bold lowercase for vectors and uppercase letters for matrices and constants. We define *seen types* to be the types in the training dataset. *Unseen types* are those that do not exist in training dataset but are found in test dataset. While in principle test types may include both seen types and unseen types, in our experiments we focus on unseen types.

## 2.4. OTyper: A Neural Network Architecture for ONET

### 2.4.1. Overview

OTyper outputs a vector for each $m_i$, where each element in the vector is a probability estimate that $m_i$ belongs to a type. Our intuition is that if mention embeddings are appropriately mapped into a common space with type embeddings, then the mention embeddings will be nearby the correct type embeddings and far away from the incorrect type embeddings—even for types that do not explicitly occur in the training set.

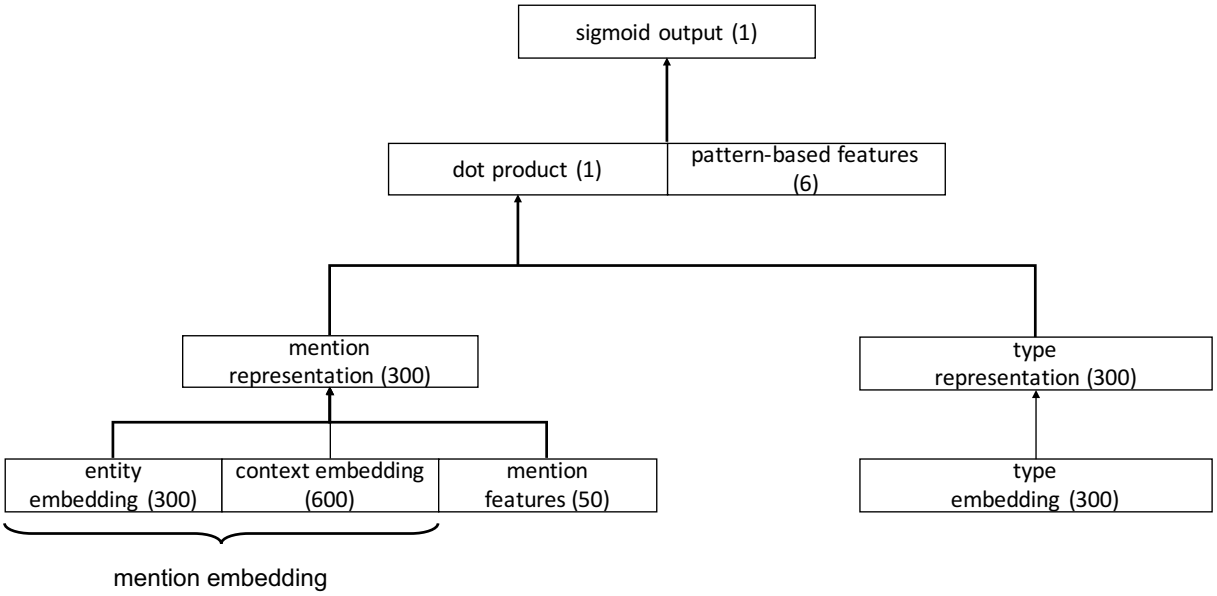


Figure 2.1. Neural architecture for OTyper. The number of neuronal units is provided for each component in parentheses

Figure 2.1 shows the neural network architecture of OTyper. The number of dimensions is listed in parentheses after the name. The arrows represent fully interconnected weight matrices with one exception—the mapping from the mention and type representation to the dot product is a dot product operation of these two representations. Also,

OTyper uses a fixed identity matrix for the weights from the type embedding to the type representation. OTyper has a single logistic function output for each <mention, type> pair. Thus, OTyper can have a different number of types in training and testing phase, which is critical for ONET. OTyper maps the mention embedding into a common space with the type embedding and is trained to minimize the dot-product distance between the mention embedding and each of its correct type embeddings.

The high level mathematical formulation of OTyper is as follows:

$$\mathbf{a}_{\text{rep}} = [\mathbf{f}_{\text{m-emb}}, \mathbf{f}_{\text{m-fet}}] * W_{\text{mention}} \tag{2.1}$$

OTyper concatenates the mention features ($\mathbf{f}_{\text{m-fet}}$) with the mention embedding ($\mathbf{f}_{\text{fet}}$) and projects it into the common space in Equation 2.1.

$$\mathbf{b}_{\text{rep}} = \mathbf{f}_{\text{t-emb}} * W_{\text{type}} \tag{2.2}$$

Equation 2.2 projects the type embedding, $\mathbf{f}_{\text{t-emb}}$, into the common space. As mentioned above, in OTyper, the type projection is an identity transformation; exploring alternative transformations is an item of future work.

$$d = \mathbf{a}_{\text{rep}} \cdot \mathbf{b}_{\text{rep}} \tag{2.3}$$

Then, the dot product between the projected mention and type is computed in Equation 2.3.

$$l = [d, \mathbf{f}_{\text{p-fet}}] * W_l + b \tag{2.4}$$

Pattern features vector ($\mathbf{f}_{\text{p-fet}}$) are concatenated with the dot product. In equation 2.4, the concatenation vector is then fed into a hidden layer, which outputs a scalar.

$$\hat{y} = sigmoid(l) \tag{2.5}$$

The last layer transforms the hidden layer output with a logistic function, to produce a probability estimate that the given mention is of the given type. To train the model, OTyper minimizes the cross-entropy loss on the training data.

Note that in the training phase, we feed seen type embeddings into OTyper, while during the test phrase, we feed target types which can include unseen types.

In 2.4.2 and 2.4.3, we introduce mention and type embeddings. The features that OTyper uses are illustrated in 2.4.4.

### 2.4.2. Mention embedding

The mention embedding in OTyper contains two parts: an entity embedding and a context embedding. OTyper adopts the mention embedding approach proposed for NET in [106], which we describe in this section.

Our entity embedding simply averages the individual word embeddings for the entity, as shown in equation 2.6. $m_i^{(j)}$ denotes the $j^{th}$ word of $i^{th}$ mention. $ml_i$ is the number of words of $i^{th}$ mention. The function $emb(\cdot)$ returns the word embedding.

$$\mathbf{f}_{\text{e-emb}}(m_i) = \frac{1}{ml_i} \sum_{j=1}^{ml_i} emb(m_i^{(j)}) \tag{2.6}$$

To generate a context embedding, OTyper trains two bi-LSTMs [33] on both sides of context. An attention model is applied to the output states of the bi-LSTMs to get weighted summations on both sides, which are concatenated for form the context embedding.

Equations 2.7 to 2.10, from [106], describe how our mention embedding is computed. Equations 2.7 to 2.9 formulate how OTyper computes its context embedding. We use $\overrightarrow{o}^l_{ij}$ and $\overleftarrow{o}^l_{ij}$ to denote the bidirectional output states of bi-LSTMs for the $j^{th}$ word in the left side context of $m_i$, $j \in \{1, \ldots, C\}$, with analogous quantities for the right side.

$$\mathbf{e}^l_{ij}(cl_{ij}) = \tanh(W_e * \begin{bmatrix} \overrightarrow{\mathbf{o}}^l_{ij} \\ \overleftarrow{\mathbf{o}}^l_{ij} \end{bmatrix}) \tag{2.7}$$

$$\tilde{a}^l_{ij} = \exp(W_a * \mathbf{e}^l_{ij}) \tag{2.8}$$

$$a_{ij}^{l} = \frac{\tilde{a}_{ij}^{l}}{\sum_{j=1}^{C}(\tilde{a}_{ij}^{l} + \tilde{a}_{ij}^{r})} \tag{2.9}$$

To get the attention weights in Equation 2.9, the attention model trains a two-layer feed-forward neural network as in Equations 2.7 and 2.8. The scalar $a_{ij}^{l}$ is the weight for the left context output state for the $j^{th}$ word. The right context scalars are analogous. The context embedding is shown in Equation 2.10.

$$\mathbf{f}_{\text{c-emb}}(cl_i, cr_i) = \sum_{j=1}^{C}(a_{ij}^{l} * \begin{bmatrix} \overrightarrow{\mathbf{o}}_{ij}^{l} \\ \overleftarrow{\mathbf{o}}_{ij}^{l} \end{bmatrix} + a_{ij}^{r} * \begin{bmatrix} \overrightarrow{\mathbf{o}}_{ij}^{r} \\ \overleftarrow{\mathbf{o}}_{ij}^{r} \end{bmatrix}) \tag{2.10}$$

The mention embedding (Equation 2.11) is the concatenation of the entity and context embedding.

$$\mathbf{f}_{\text{m-emb}}(m_i, cl_i, cr_i) = [\mathbf{f}_{\text{e-emb}}(m_i), \mathbf{f}_{\text{c-emb}}(cl_i, cr_i)] \tag{2.11}$$

We also tried the two alternative context models proposed in [106], one based on averaging the context embeddings and the other based on LSTMs without attention. We found the attention embedding performed the best, so we use it in OTyper.

### 2.4.3. Type embeddings

In equation 2.12, OTyper computes type embeddings by simply averaging the word embeddings of the words comprising the type name:

$$\mathbf{f}_{\text{t-emb}}(t_i) = \frac{1}{tl_i}\sum_{j=1}^{tl_i} emb(t_i^{(j)}) \tag{2.12}$$

where $t_i^{(j)}$ denotes the $j^{th}$ word of type $t_i$. $tl_i$ is the number of words in $t_i$.

### 2.4.4. Features

Shimaoka et al. [106] showed substantial F1 score improvement in NET by incorporating mention features such as syntactic features, word shape features and topic features. We also use these mention features in OTyper. Following [106], we represent mention features as binary vectors and use a trainable linear projection to map the binary vector to lower dimension. We use $f_{\text{m-fet}}(m_i)$ to represent mention features for mention $i$.

Pattern-based features are helpful for ONET. For example, given an entity-type pair, $<e, t>$, if the number of textual occurrences of "$e$ is a $t$" is low in a large corpus, it is less likely that $e$ is of type $t$. We use two kinds of pattern based features in OTyper : entity-type features and type-only features. We use a set of hypernym patterns applied to a large corpus taken from the web-is-a database [102]. We compute features based on the number of pattern matches for different entities and types. Entity-type features capture the pattern matching information of a specific $<e, t>$ pair. Specifically, entity-type features include: the number of matches, the number of distinct matched patterns, and the number of matched URL domains. Type-only features marginalize entity-type features over types, and can be viewed as a prior over types — i.e. types that appear in more patterns may be the type labels that OTyper should be output more often, all else being equal. Equations 2.13, 2.14 are the formulas for entity-type features ($\mathbf{f}_{\text{e-t-fet}}$)

and type-only features($\mathbf{f}_{\text{t-o-fet}}$). $n_m$, $n_p$ and $n_u$ are the number of matches, the number of distinct matched patterns and the number of URL domains for $<e, t>$ separately. Pattern based features (in Equation 2.15) combines entity-type features with type-only features.

$$\mathbf{f}_{\text{e-t-fet}}(e_i, t_i) = [n_m(e_i, t_i), n_p(e_i, t_i), n_u(e_i, t_i)] \tag{2.13}$$

$$\mathbf{f}_{\text{t-o-fet}}(t_i) = \sum_i \mathbf{f}_{\text{e-t-fet}}(e_i, t_i) \tag{2.14}$$

$$\mathbf{f}_{\text{p-fet}}(e_i, t_i) = [\mathbf{f}_{\text{e-t-fet}}(e_i, t_i), \mathbf{f}_{\text{t-o-fet}}(t_i)] \tag{2.15}$$

## 2.5. OTyper Experimental Result

This section evaluates OTyper and answers three questions:

**1:** How well can OTyper label unseen types?

**2:** How much do similar training types help—does unseen type accuracy correlate with how similar the training type embeddings are to the unseen type embeddings?

**3:** How much does each feature impact the accuracy of OTyper?

### 2.5.1. Experimental setup

We evaluate OTyper on two datasets: FIGER(GOLD) and MSH-WSD.

**1: FIGER(GOLD)** is a benchmark dataset for fine-grained NET. The training and development FIGER(GOLD) data were generated from Wikipedia text [67]. The FIGER(GOLD) test dataset consists of manually annotated newspaper articles. Each mention has 113 binary type labels, where the types were derived from Freebase. There are 200,000 mentions in training dataset, and 10,000 mentions for development.

**2: MSH-WSD** is a word sense disambiguation dataset that was automatically collected from the Unified Medical Language System (UMLS) Metathesaurus and the manual MeSH indexing of MEDLINE [53]. We type each MSH-WSD mention using UMLS tags. Specifically, we define the correct types for a mention to be all of its ancestors in the UMLS taxonomy tree. We randomly select 80% of the mentions for training, 10% for development and 10% for test. There are 1387 different types in MSH-WSD.

We use published mention features for the FIGER(GOLD) dataset from [106]. There are no mention features available for MSH-WSD. The pattern features used in our experiments are extracted from the web-is-a database [102]. Mention and type embeddings are taken from pre-trained GloVe-840B [84] word embeddings.

For all experiments, the dimension of the common space is 300. As mentioned above, we set $W_{type}$ to be the identity matrix. The rest of hyper-parameters are the same as in [106]. The projection of the mention features is 50 dimensional. The Adam optimizer [58] with a learning rate 0.001 is applied to minimize the loss. The model is trained for 5 epochs with batch size 1,000. Dropout with keep probability of 0.5 is applied to the

mention embeddings and mention features. The context window size is 10. We present the test performance of the model that performed best of the development set.

Our evaluation metric is weighted AUC-ROC score, which computes an AUC-ROC score for each type and then averages the scores weighted by the type frequency.

### 2.5.2. Unseen type labeling

To evaluate how well OTyper labels unseen types, we split test types into 10 parts and do 10-fold cross-validation on the types. For each fold, only type labels *other than* the test types are utilized in the training and development set.

We compare OTyper with two baseline models. Pattern-based methods are a canonical approach for identifying hypernym relations. So, our first baseline is a pattern-based model. It uses the number of matches in web-is-a for <e,t> as its score [100], and forms a relatively strong baseline. The second baseline is based on word embeddings. It trains a logistic regression model to classify the vector difference between the entity embedding for $e$ and the type embedding for $t$. Vector differences between embeddings have been shown to reflect relation information [73]. In addition, to provide an upper bound on accuracy, we also evaluate OTyper in a fully supervised setting in which all types are available in training.

| Model | AUC-ROC |
|---|---|
| Pattern baseline | 0.639 |
| Embedding baseline | 0.413 |
| OTyper | 0.870 |
| Supervised upper bound | 0.943 |

Table 2.1. Unseen type weighted AUC-ROC comparison on FIGER(GOLD). OTyper outperforms the baselines.

Table 2.1 presents the results on FIGER(GOLD). The results show that using only pattern-based features or embeddings cannot solve the ONET task, because ONET is context-sensitive whereas the pattern features and embeddings are context-insensitive. OTyper achieves an AUC-ROC of 0.870 and outperforms the two baselines by substantial margins. The pattern baseline is better than the embedding baseline in these experiments. The supervised upper bound model gets an AUC-ROC score of 0.943. It is notable that OTyper can score relatively close to the performance of the supervised model, given that OTyper must solve the much more challenging ONET task in which the test types do not occur in the labeled training examples.

| Model | AUC-ROC |
|---|---|
| Pattern baseline | 0.005 |
| Embedding baseline | 0.350 |
| OTyper | 0.780 |
| Supervised upper bound | 0.891 |

Table 2.2. Unseen type-weighted AUC-ROC comparison on the MSH-WSD dataset. OTyper outperforms the baselines.

Table 2.2 shows the results on MSH-WSD. Again, OTyper achieves much higher weighted type AUC-ROC score compared to the baselines. However, the AUC-ROC is not as high as in FIGER(GOLD) because MSH-WSD has a much higher number of types than FIGER(GOLD), and also lacks mention features, which makes the ONET task harder than in FIGER(GOLD). Further, since MSH-WSD type comes from biomedical domain, they rarely show up in the Web corpus used to form the web-is-a database. Thus, pattern-based features are sparse. In fact, out of 51 million <e, t> pairs in MSH-WSD only 817 of them have non-zero pattern matches. This implies that the pattern-based features are not informative here, and is the reason that the pattern-based baseline gets

almost zero AUC-ROC score on MSH-WSD data. Due to the feature limitations of MSH-WSD, we only use FIGER(GOLD) in the following experiments.

Note that although OTyper achieves high weighted type AUC scores, if we evaluate on an F1 metric over unseen types, OTyper achieves a score of 0.26. This indicates that OTyper works well for ranking the unseen types, but not that well for classification.

In addition, for each fold of the 10-folds cross-validation, our model holds out about 4 types. So, to predict a test type, our model lost the labels of three more types in training dataset. These missing types in training data have negative effect on test type prediction, especially when the missing types are similar to the test type. We will investigate this more in 2.5.3.

### 2.5.3. Influence of training types

This subsection attempts to explain the performance of OTyper by analyzing how much the unseen type AUC-ROC correlates with how similar the training type embeddings are to the target unseen type embedding. We use cosine similarity in Equation 2.16 to measure the distance between types:

$$\textit{cos-similarity}(\mathbf{a}, \mathbf{b}) = \frac{\mathbf{a} \cdot \mathbf{b}}{\|\mathbf{a}\|_2 \|\mathbf{b}\|_2} \tag{2.16}$$

For these experiments, we focus on the 11 of our 41 test types that occur at least ten times in our test dataset. For each of the 11 types, we hold the type out from training and development sets and train three different OTyper models. In the first model, the training and development types contain all other types other than the test type. We name

this model *All*. In the second and third models, the three most and least similar types from the test type are removed from the training and development datasets. We call them *top-3* and *bot-3* model. IfOTyper is utilizing similar types to the test type in the training set as a source of information, we would expect *top-3*, which removes the most similar types, to substantially underperform *bot-3*, which removes dissimilar types.

| Type name | All | Top-3 | Bot-3 |
|---|---|---|---|
| location | 0.739 | 0.692 | 0.742 |
| person | 0.885 | 0.82 | 0.897 |
| organization | 0.880 | 0.819 | 0.811 |
| city | 0.917 | 0.856 | 0.921 |
| country | 0.857 | 0.787 | 0.893 |
| company | 0.849 | 0.611 | 0.842 |
| sports team | 0.862 | 0.827 | 0.873 |
| athlete | 0.957 | 0.897 | 0.953 |
| building | 0.929 | 0.885 | 0.894 |
| educational institution | 0.888 | 0.720 | 0.904 |
| time | 0.793 | 0.863 | 0.801 |
| **Average** | **0.869** | **0.798** | **0.866** |

Table 2.3. Performance of OTyper when holding out types that are similar (*Top-3*) or dissimilar (*Bot-3*) to the target type, compared to keeping all training types (*All*). We report AUC-ROC on the FIGER(GOLD) dataset. On average, removing similar types hurts performance, whereas removing dissimilar types has negligible impact.

Table 2.3 shows the type AUC-ROCs of all three models. On average, *bot-3* achieves 0.068 higher AUC-ROC score than *top-3*. The result suggests that to predict unseen types, the similar types in the training data are more informative than dissimilar types. We also observed that average AUC-ROC of *All* is almost the same with bot-3. So, dissimilar types do not affect unseen type prediction.

### 2.5.4. Feature analysis

This section investigates whether having mention- and pattern-based features is helpful in predicting unseen types. We remove each feature from our model separately and evaluate our model using 10 fold cross-validation as in 2.5.2. The results are summarized in Table 2.4.

| Model | Weighted AUC-ROC |
|---|---|
| OTyper | 0.870 |
| OTyper (- mention features) | 0.863 |
| OTyper (- entity-type features) | 0.842 |
| OTyper (- type-only features) | 0.848 |

Table 2.4. Impact of features on the FIGER(GOLD) dataset. The pattern-based features are most valuable for this dataset.

Table 2.4 shows that removing any of the features results in some AUC-ROC decrease. AUC-ROC does not decease very much when mention features are removed. We did not do a statistical significant test on these AUC-ROC scores. however, given there are more than two millions mentions in FIGER(GOLD), tiny change of AUC score is a big difference. As additional analysis, we find that when mention features are removed, the AUC-ROCs increases for 18 unseen types, decreases for 21 unseen types, and remains unchanged for the other two. Thus, there appears to be no clear advantage in using mention features. The pattern-based features are found to be more informative for FIGER(GOLD) dataset. Even without these features, OTyper still performs well. We believe that this is because, a) world embedding vectors consists meaning fully information of types; b) OTyper learns to map type representation vector space to mention representation vector space.

### 2.5.5. Supervised setting

Our focus is on ONET in which types can be unseen. In a supervised setting, where all types are seen, we would expect OTyper performs similar to that of the recent state-of-the-art model in [106]. To verify this, we evaluate OTyper against the model from [106] (NFGEC for short) on FIGER(GOLD) using the same evaluation set-up in that work, i.e. strict accuracy, loose macro F1, and loose micro F1 scores. For this experiment, all 113 types are seen types.

For NFGEC, we use the attentive context encoder with mention features, which is the setting that achieves the highest FIGER(GOLD) accuracy and F1 scores in [106]. We use two feature settings for OTyper. The first setting includes only mention features. This setting uses exactly the same input data as NFGEC, i.e. mention embeddings, context embeddings and mention features. The second setting includes all features in OTyper, i.e. mention, entity-type and type-only features. Table 2.5 shows the results. When OTyper uses the same input data with NFGEC, OTyper gets slightly lower accuracy and F1 scores than NFGEC. When all features are included, OTyper gets similar accuracy and F1 scores with NFGEC. Overall, OTyper performs comparably to NFGEC in the supervised setting.

| Model | Acc. | F1 Macro | F1 Micro |
|---|---|---|---|
| NFGEC | 0.597 | 0.790 | 0.754 |
| OTyper(mention features) | 0.584 | 0.776 | 0.752 |
| OTyper(all features) | 0.595 | 0.779 | 0.759 |

Table 2.5. Comparison in the supervised setting. OTyper achieves comparable F1 to the state-of-the-art NFGEC.

## 2.6. Conclusion

In this chapter, we introduced the task of Open Named Entity Typing (ONET), which is NET when the set of target types is not known in advance. We proposed OTyper, a neural network architecture for ONET. OTyper relies on type embeddings in order to extend to unseen types. The experimental results demonstrate that on unseen types OTyper outperforms two baseline models and achieves a weighted AUC-ROC of 0.870 on the benchmark FIGER(GOLD) dataset, and a score of 0.78 on the MSH-WSD dataset. OTyper can serve as a baseline model for future ONET systems. Finally, our analysis revealed that similar training types provide more information for unseen type prediction than dissimilar training types do.

The Investigation of ONET helps us to better understand the task of highlighted text prediction. As for highlighted text prediction with textual reviews, instead of treats types as labels, it actually treats textual reviews to be labels for entire highlighted sentence. This chapter shows that well pre-trained word embedding vector provides information for types, although these types do not exists in training data. In the next chapter, to get meaningful representation of unseen reviews we use pre-trained word embedding and sequence-to-sequence model, which is an over-labeling technique [75].

CHAPTER 3

# Highlighted Text Prediction

The previous chapter shows that by leveraging the information in word embedding, it is possible to type named entities using unseen labels. In this chapter, we move one step forward, which is highlighted text prediction using over-labeling technique. We collect peer review data in EECS-349 machine learning class at Northwestern University. We ask students to highlight important regions (topic, data set, feature, algorithm) of submissions and provide feedback on the highlighted text. To encourage peers highlight the correct span of text, graders graded the quality of peer reviews. In this way, we have a peer review data set to train a neural network for highlighted text prediction.

The organization of this chapter is as follows: data collection is included in section 3.1. Section 3.2 talks about the background knowledge of highlighted text prediction. Section 3.3 shows our methods to predict the highlighted text and evaluate the results.

## 3.1. Peer Review Data Collection

To the best of our knowledge, there is no public peer review data set for highlighted prediction yet. The only large peer review data is [57]. However, the peer reviews in this data set are global peer reviews not sentence level reviews.

### 3.1.1. Project assignment overview

We collect data in EECS-349 Machine Learning class at EECS department at Northwestern University. In the course project of EECS 349, students work in groups. Each group proposes a machine learning task they want to address. Then, they collect data and train a machine learning model to solve their proposed tasks. The course project lasts for five weeks. Students submit project proposals in PDF format in the first week. In the third week, students submit status reports. At the end of the fifth week, students design a web site to demonstrate their work.

We ask students to do peer review for project proposals and status reports. By reviewing project proposals, students are exposed to more machine learning topics and techniques, which will help them to adjust their plan in this course project. Project status peer review helps student evaluate their progress of the course project by comparing with the progresses of other groups. To encourage students to provide good quality reviews, five graders graded student's peer review based on their quality and quantity of peer reviews. Note that grading peer review quality increases grading workload, but our goal here is to train a model to automatically identify important texts. Once the model is trained, it could save peer review time in the future.

### 3.1.2. Peer review process

For project proposal, students are asked to provide peer reviews on different aspects of randomly assigned submissions: topic, dataset, feature, and machine learning algorithm. For status report, students provides comments on dataset, feature, machine learning algorithm, and preliminary result. In this chapter, we call these aspects *review labels*.

To provide peer comments of these aspects, students first highlight the corresponding span of text in PDF file, then add comments on the highlighted part. Figure 3.1 is an comment example of the topic of a project proposal. We pixelate the name of the peer reviewer and the grader.

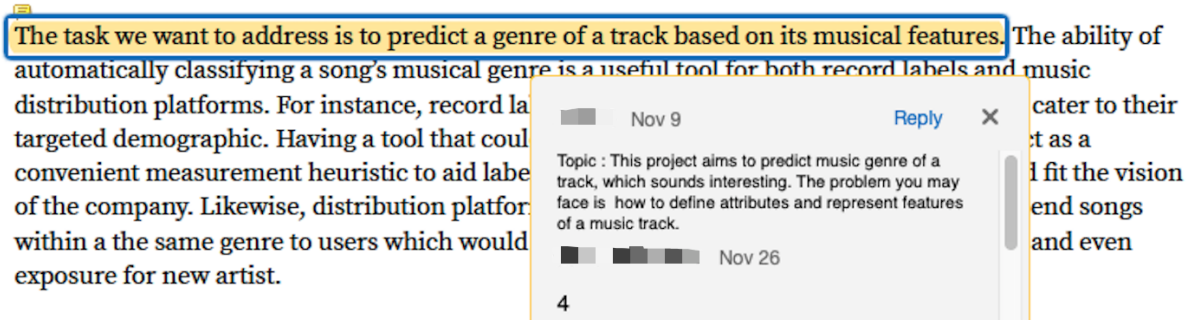


Figure 3.1. Topic peer comment example

Note that to specify the aspect of peer review. Peer comments starts with the name of the aspect. For the example in Figure 3.1, it starts with "Topic :"

Students are also allowed to provide other aspects of reviews, like Figure 3.2, by highlighting a sentence and provide free-style textual comments.

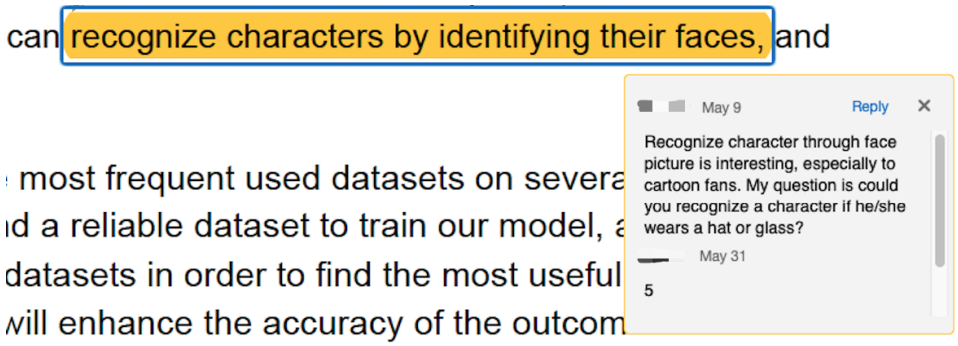


Figure 3.2. Free style peer comment example

If students want to provide global comments, they can put their comments in 'sticky notes' without highlighting any text. Figure 3.3 shows an example of a global comments.

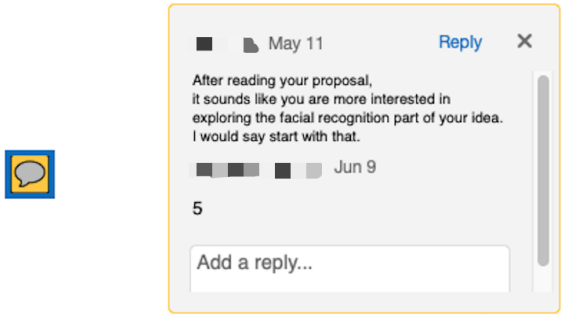


Figure 3.3. Global peer comments example

Peer reviews were graded by graders according to two aspects: quantity and quality. Students received full grades on quantity if they provide seven or more comments on a submission. Quality scores were measured by graders based on the helpfulness of peer reviews. The grading scale for peer reviews is from zero to five.

### 3.1.3. Statistical information of peer review data

There were 220 students in EECS-349 2018 Spring. Students worked in groups for the course project. Overall, there were 90 project groups. Each group submitted one project proposal and one project status report. In peer review process, each student got three project proposals and project status reports to do peer review. So, we collected 1.32K annotated PDFs. Each PDF has eight comments on average. The total number of annotated comments is about 10.56K.

## 3.2. Background of Highlighted Text Prediction

This section describes background and key steps to highlighted text prediction, which includes, peer review extraction, seq2seq model, and sequence tagging.

### 3.2.1. Peer review extraction

Our collected data are annotated PDFs. We need to extract annotations from those raw PDFs. There are two kinds of annotation: pop-up notes, which are used for peer reviews of highlighted text, and sticky notes, which are used for global peer reviews. These peer reviews are extracted using Java itextpdf package[47]. Each pop-up note has a label, which denotes the category (aspect) of it, such as topic, dataset, etc. After peer review extraction, a list of training examples is created for the task of highlighted text prediction. Each training example consists: peer review content, highlighted span of text (None, for global peer reviews), category(None, for global peer reviews), and text of the entire PDF.

### 3.2.2. Seq2seq model

Sequence-to-sequence (Seq2Seq) [109] is a neural network model, which takes a sequence as input and outputs a sequence. It has achieved state-of-the-art results on many NLP tasks, including machine translation, speech recognition, and text summarization, etc. Seq2seq model consists of two parts: encoder and decoder. Usually, both of them are RNN. Encoder maps an input sequence into an encoder state. The decoder takes encoder state as inputs and generates an output sequence. Figure 3.4 illustrate the structure of seq2seq model. This model takes sequence ABC as input, and the output is WXYZ. The blue box is the encoder state, which is also the beginning state of the decoder.

Attention mechanism has been widely used to improve the performance of seq2seq as an optional component. During decoding, attention mechanism takes previous states as a query vector. A weighting array is computed based on this query vector. Then, a weighted sum RNN states in the encoding phase is used to generate the current decoding state. In

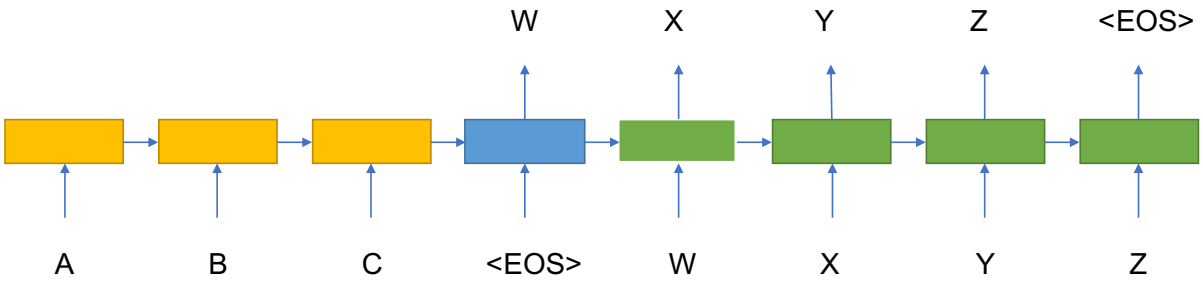


Figure 3.4. Sequence-to-sequence model architecture

such a way, the decoder is able to focus on different encoding RNN states. This work uses attention mechanism. Note that the attention mechanism is not shown in seq2seq figures.

### 3.2.3. Sequence tagging

Highlighted text prediction can be viewed as a sequence tagging problem. Each token can be tagged as either 0 or 1, where 1 denotes the token is highlighted, 0 means the token is not highlighted. For example, in Figure 3.1 labels from *The* to *features* are labeled as 1s, while labels of the rest tokens are all 0s. A traditional way to do sequence tagging is using LSTM [40] plus a CRF [62] layer on the top. Figure 3.5 shows the architecture of a common LSTM+CRF sequence tagging neural model. As is shown, the input of each LSTM cell is each token in a sentence. The LSTM output is then fed into a CRF layer. In order to train this model, cross-entropy loss is minimized. The data we collected from EECS-249 actually contains more data, textual reviews and review labels. Can we take advantage of these data to improve highlighted text prediction performance? This is the problem we will solve in the next section.

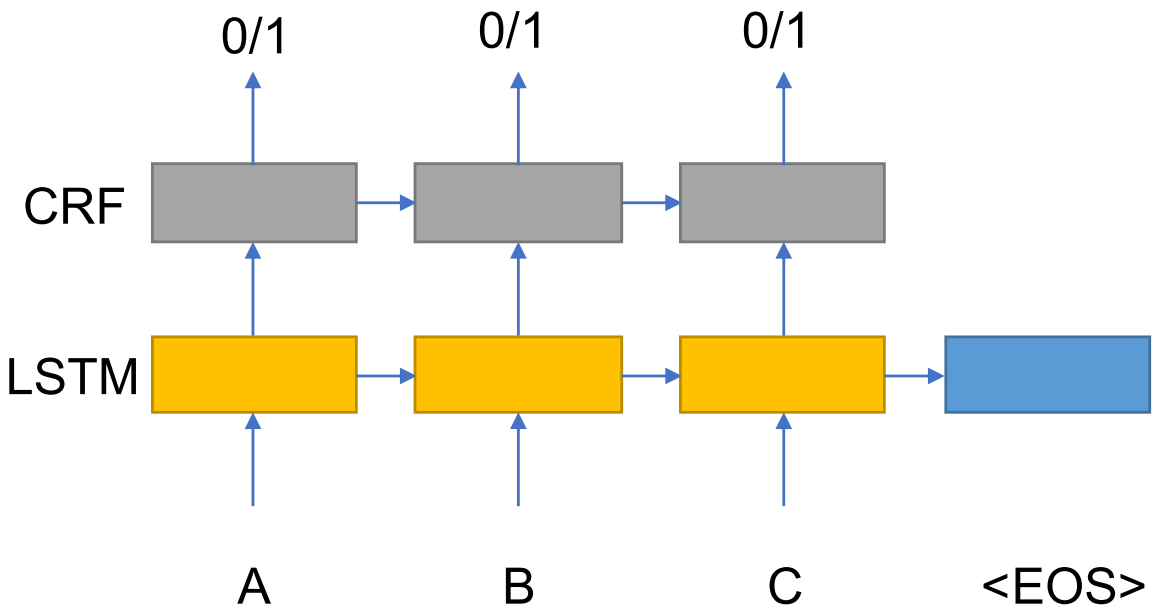


Figure 3.5. Traditional neural network architecture for sequence tagging

### 3.3. Highlighted Text Prediction Models

This section describes our methods of highlighted text prediction. Note that it is not appropriate to directly use OTyper to solve this task. If we consider mention as the highlighted text and the type as textual peer review, it only works well for highlighted text. It is not possible to combine non-highlighted text in this model. So, we propose two methods, that are different from OTyper, for highlighted text prediction. One is to take advantage of textual reviews to predict highlighted text. The other is to use review labels provided by peers to improve prediction performance. Note that both of these models adopt the idea of over-labeling [75]. The idea of over-labeling is to let the model predict something harder than the target. Then, convert the harder task to the target task in the test phase. The intuition is that by doing the harder task, models can learn to

create more sophisticated features, which results in better prediction performance. In our case, the hard tasks are: when predicting highlighted tokens, our model also minimizes the probability of generating textual reviews and predicts review labels given highlighted texts.

**Over-labeling(review text):** Figure 3.6 illustrates the neural architecture of highlighted text prediction using textual reviews as the extra input. The idea is to sum up seq2seq loss and the binary label loss per token as the training loss. Note that the review sequences for non-highlighted sentences are empty strings.

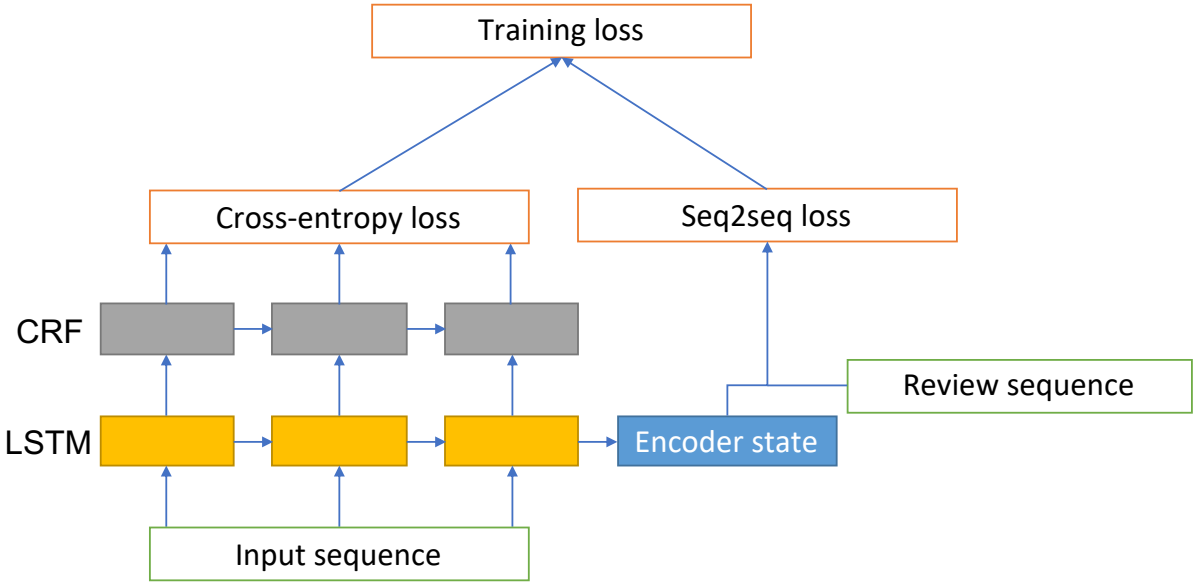


Figure 3.6. Highlighted text prediction using textual review as extra input

**Over-labeling(review label):** We also propose another model which takes peer topic labels as the extra input. Figure 3.7 shows the architecture. This model combines two losses, binary label cross-entropy loss and multi-label cross-entropy loss. The multi-label are the aspects of reviews.

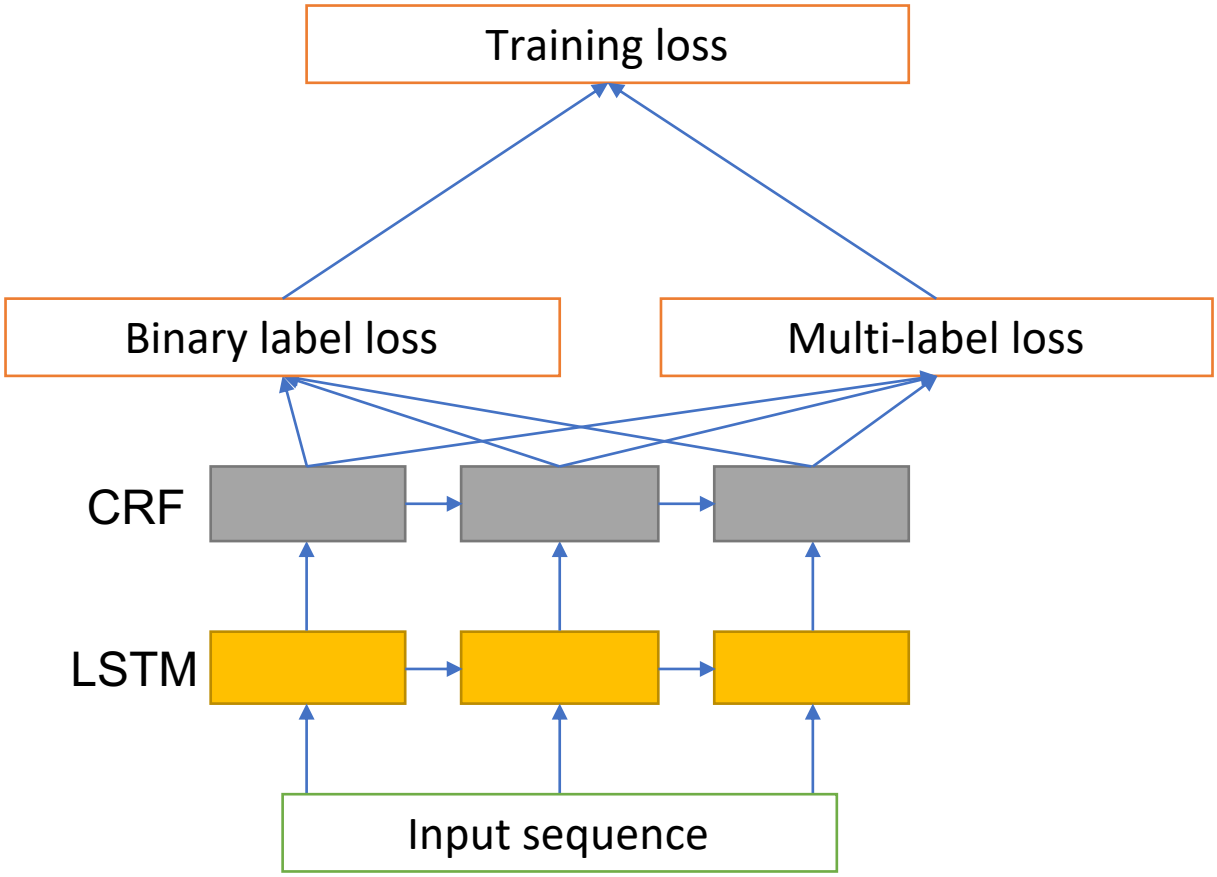


Figure 3.7. Highlighted text prediction using review labels as extra input

We evaluate the F1 score of the proposed models using the data we collected at EECS-349 Machine Learning class. Table 3.1 shows the token accuracy of the proposed models. We also compare our models with a baseline model. The baseline model predicts the binary label using LSTM+CRF architecture as we mentioned earlier in this chapter.

| Model | Over-labeling(review text) | Over-labeling(review label) | baseline |
|---|---|---|---|
| F1 score | 0.6935 | 0.7259 | 0.7143 |

Table 3.1. F1 score comparison of highlighted text prediction

As we can see from Table 3.1, using over-labeling(review label) provides the best token accuracy. This result implies using review labels helps the model on feature learning and results in better performance. Since the difference of token accuracy between over-labeling(review label) and baseline is small. We further run a Fisher's exact test compares if there is a significant difference between these two methods. The p-value is 0.0141 (smaller than 0.05), which indicates the difference between baseline and over-labeling(review label) exists. The textual review over-labeling did not achieve a good F1 score. We believe there are several reasons for this. First, generating meaningful reviews for homework submissions needs to understand submissions very well, which means domain knowledge is required. However, domain knowledge for homework is hard to get and encode. Our model does not take domain knowledge into consideration. Thus, it may not be able to do well on the seq2seq part. Instead of introducing domain knowledge, a cheaper way to incorporate external information into our model is to use massive pretrained models. Exploring pretrained model is an item of future work. Second, training a seq2seq model usually requires millions of training data. The data that we collected in class has only several thousand reviews, which may not be enough to train a seq2seq model well. Third, since the reviews of non-highlighted text are empty, most of the time the decoder predicts empty strings, which may impact the performance of seq2seq when it is trained on the highlighted text.

## 3.4. Conclusion

To reduce peer review workload, this chapter proposed highlighted text prediction models using over-labeling technique. The proposed models can be used to predict important span of text of submissions. Peers can save time by focusing on the important parts and skip the rest. Due to unpromising results and expansive cost, we did not evaluated the whether our methods reduce peer review effort in real classroom setting. Also, we want to note that the proposed methods are specific for one course. It may not be easily generalizes to other courses. Thus, a more generalized model, which could predict important regions of any submissions, is needed to increase practicality.

CHAPTER 4

# Consensus Grade Estimation

Since peers typically lack the subject matter mastery of the instructor, peer grades exhibit both bias and variance, which makes consensus grade estimation a challenging task. How to estimate consensus grades based on peer grades is a challenging task [1]. In this chapter, we explore peer grading techniques by first analyzing and improving a state-of-the-art peer grading algorithm. Then, we proposed novel peer grading methods, which outperforms state-of-the-art and baseline methods. Finally, we discuss learning outcomes of different peer reviewing processes.

## 4.1. Related Work

Alfaro and Shavlovsky [3] proposed Vancouver algorithm, which measures each peer's grading accuracy, by comparing the grades assigned by the peer with the grades by other peers to the same submissions and gives more weight to the peer grades with higher measured accuracy. The basic idea of Vancouver is like EM (Expectation-Maximization) algorithm. Vancouver estimates consensus grades by weighted averaging peer grades based on peer reputation, which is represented by the variance of each peer. Then, Vancouver updates peer variances based on the peer grades and consensus grades. Vancouver algorithm runs iteratively until the consensus grades and peer variances converge. Vancouver algorithm assumes that all peers are non-biased. However, peers are not trained graders. Thus, non-biased peer assumption is hardly met in reality. Piech et al. [85] proposed a

probabilistic method to do peer grade estimation. They propose three peer grades generation models and use Gibbs sampling to do the inference. Their methods estimates grader biases and reliabilities based on peer grades. Raman and Joachims [94, 93] propose methods for ordinal peer grading. They claim that besides cardinal grades [3, 85], ordinal information should be take into consideration during peer grading. They proposed both ordinal and cardinal methods for peer grading. According to their results [94], their ordinal enriched method achieved better results than the probabilistic method proposed in [85].

However, all of the previous works of peer grading have two shortcomings. First, they cannot deal with systematic peer biases. For example, if most of peers overestimate submission grades, the consensus grade estimated by peer grading algorithm will be higher than ground truth grades. Second, previous works did not take textual peer reviews, which is practically available and abundant, into consideration. Actually, quality of textual peer reviews reflect how much effort a peer spend on reviewing a submission to some extent. Intuitively, a detailed and constructive textual review indicates the peer reviewer understands the submission well and the corresponding peer grade is more likely to be accurate. Thus, peer grading may be improved if textual reviews can be taken into consideration. Also, we note that we target a classroom setting, where classes have on the order of 50-100 students and the instructor feedback can cover a significant proportion of the students, rather than the large MOOC setting with much more student data (but less proportionally instructor feedback).

Finally, our use of textual peer review comments is related to work in automated grading. Automated Essay Scoring (AES) [104] is the process that computer evaluate the

score of the writing. The input of AES is a text document and a rubric. AES outputs the predicted score of the writing. Automatic peer review grading is related to AES since peer review grading also predicts scores for peer reviews, which is a kind of text document. However, the different between AES and peer review grading is that peer review grading is context sensitive. Specifically, the score of the same review could be different for different documents. Different AES systems uses different grading strategies. Project Essay Grader(PEG) [81] takes some graded essays as training data. To predict the final score, PEG computed the coefficients from the training examples. Intelligent Essay Assessor (IEA) [46] grades essays using Latent Semantic Analysis, which predicts the distribution of word semantics. Electronic essay rater (E-rater) [5] grade essays by identify specific lexical and syntactical features.

### 4.2. Peer Grading Data Collection

To evaluate peer grading performance, we collected four peer review data sets from an EECS-336 Algorithm Design in EECS department at Northwestern University from the following quarters: 2017 Spring, 2017 Fall, 2019 Spring, and 2019 Fall. The instructor assigned one to two homework per week in EECS-336. Students worked in groups of one or two. Students submitted their submissions to Canvas, the learning management system used at Northwestern University. After submissions were collected, Canvas learning management system assigned several peers to review each submission. Each peer reviewed three submissions. Peers provided their feedback (textual reviews and peer grades) on Canvas. The instructor also graded a portion of the submissions. In addition, Canvas makes ensure that each peer has at least one submission that is also graded by TA. Peers

were graded based on how close their grades are to the instructor's grades. The difference between EECS-336 data sets is that in the 2019 Spring and Fall, peers are asked to provide separate textual reviews on three aspects for each homework (including correctness of the algorithm, correctness of the proof, and clarity of the writing). Then, the instructor also graded a part of the textual reviews by hand. In the 2017 Fall and Spring, peers were asked to provide one overall textual review, and the instructor did not evaluate the quality of textual reviews. Only the accuracy of the peer grades are evaluated based on the distance to TA grades.

To help isolate how method performance depends on specific characteristics of the peer grading distribution, we also create three synthetic data sets (syn-asymbias, syn-symbias, syn-unbias). We simulate a class with 90 students and 30 homework. For each homework, each student submits one submission and reviews three submissions. Ground truth grades of submissions are uniformly random sampled from $[50, 100]$. Each peer has a peer variance, which is uniformly sampled from $[2, 20]$, and a peer bias. Peer biases of syn-asymbias and syn-symbias are uniformly sampled from $[0, 20]$ and $[-10, 10]$. Peer biases of syn-unbias are zeros (unbiased). Peer grades are sampled from normal distributions: $N(\textit{peer bias} + \textit{ground truth},\ \textit{peer variance})$. Note that the synthetic data sets do not have textual peer reviews. Table 4.1 brief summarises the data sets. Table 4.2 shows biases of synthetic data sets.

| data set | students num | homework num | review num | text reviews | text review evaluation |
|---|---|---|---|---|---|
| 2019 Spring | 49 | 14 | 1783 | ✓ | ✓ |
| 2019 Fall | 65 | 14 | 2360 | ✓ | ✓ |
| 2017 Spring | 98 | 17 | 4064 | ✓ | × |
| 2017 Fall | 92 | 17 | 3068 | ✓ | × |
| synthetic | 90 | 30 | 13500 | × | × |

Table 4.1. Data sets summary

| | syn-asymbias | syn-symbias | syn-unbias |
|---|---|---|---|
| Peer bias | Uniform(0, 20) | Uniform(-10, 10) | 0 |

Table 4.2. Synthetic data peer biases

### 4.3. Vancouver Algorithm Case Study

This section illustrates how Vancouver algorithm, one of state-of-the-art peer grading algorithms, computes consensus grades. The basic idea of Vancouver is like Expectation-Maximization (EM) algorithm. At high-level, Vancouver computes consensus grades based on peer accuracy, which is represented by the variance of each peer. Then, Vancouver again updates peer variance based on the peer grades and the consensus grades. Vancouver algorithm runs iteratively until the consensus grades and peer variances converge. The assumption of Vancouver algorithm is that although different peers may spend different amounts of time to do peer review, each peer put equal effort to review each assigned submissions. For example, a peer spends three hours to review three submissions, he or she will spend one hour to review each.

When calculating consensus grades, Vancouver leaves out each peer at a time. The intuition is that, by doing so, a bad peer will not impact the consensus grades and bad peers will be detected. Similarly, when calculating peer variances, it lefts out each submission at a time. Without this left out strategy, we found that the results of Vancouver

show that some peers have very high qualities meanwhile the rest peers' qualities are very low. Thus, the estimated high quality peer grades dominate the final submission grades. However, those estimates may not be accurate and thus it is not optimal to have those peers dominate.

Algorithm 1 and 2 illustrates pseudo-code of the Vancouver algorithm proposed by Alfaro and Shavlovsky in [3]. Specifically, algorithm 1 is the iteration part of Vancouver. In each iteration, it calculates consensus grades and peer variances, while left out peers or submissions at each step. Thus, algorithm 1 has multiple peer variances and consensus grades. Based on the outputs of algorithm 1, algorithm 2 calculates unique variance for each peer and unique consensus grade for each submission.

Vancouver algorithm sets all peer variances to be 1.0 as the initialization value (line 1). In each EM iteration (line 2), line 3-7 computes the submission variances by summing all peer variances who reviewed the submission, except the left out peer. Line 8-13 then calculates the consensus grades. Vancouver algorithm weighted averages peer grades ($grade_{peer}$) based on peer qualities (peer variances). Line 14-20 updates peer qualities based on estimated consensus grades. Vancouver estimates the quality of a peer based on the difference between peer grades and estimated consensus grades (line 16) and the submission variance (line 17). Peer qualities are weighted averaged different, which weights are the submission variance and values are square difference in line 16. After running EM for several iterations, peer variances and consensus grades converge. Then, algorithm 1 returns the final peer variances and consensus grades.

After the EM part, algorithm 2 calculates unique variances for each peer and unique consensus grades for each submission. Algorithm 2 is similar to algorithm 1, except that

**Algorithm 1:** Original Vancouver Algorithm Pseudo-code, part 1

```
1  Initialize all peer variances (p-var) to 1.0;
2  for id_iteration in range(0, Max Iterations) do
3      for each submission id := id_sub do
4          for each peer id who reviewed submission id := id_p do
5              var_sub[id_sub][id_p] = sum_{id_p' != id_p} var_p[id_p'][id_sub]
6          end
7      end
8      Initialize all consensus grade (grade_con[id_sub][id_p]) to zero.
9      for each submission id := id_sub do
10         for each peer id who reviewed submission id := id_p do
11             grade_con[id_sub][id_p] =
                 sum_{id_p' != id_p} grade_peer[id_sub][id_p'] * var_p[id_p'][id_sub] / var_sub[id_sub][id_p']
12         end
13     end
14     for each peer id := id_p do
15         for each submission id which is reviewed by id_p := id_sub do
16             d_sum = sum_{id_sub' != id_sub} (grade_peer[id_sub'][id_p] - grade_con[id_sub'][id_p])^2 *
                 var_sub[id_sub'][id_p]
17             v_sum = sum_{id_sub' != id_sub} var_sub[id_sub'][id_p]
18             var_p[id_p][id_sub] = d_sum / v_sum
19         end
20     end
21 end
22 return var_sub, grade_con, var_p
```

it doesn't left out peers or submissions. Line 1-3 computes submission variances. Line 4-6 computes consensus grades. Line 7-11 computes peer variances. Since submission variances is an internal variable, Vancouver only returns consensus grades and peer qualities (peer variances) in the end.

**Algorithm 2:** Original Vancouver Algorithm Pseudo-code, part 2

**for** *each submission id* := $id_{sub}$ **do**
  $var_{final-sub}[id_{sub}] = \sum_{id_{p}'} var_p[id_{p}'][id_{sub}]$
**end**
**for** *each submission id* := $id_{sub}$ **do**
  $grade_{final-con}[id_{sub}] =$
  $\sum_{id_{p}'} grade_{peer}[id_{sub}][id_{p}'] * var_p[id_{p}'][id_{sub}]/var_{final-sub}[id_{sub}]$
**end**
**for** *each peer id* := $id_p$ **do**
  $d_{sum} = \sum_{id_{sub}'} (grade_{peer}[id_{sub}'][id_p] - grade_{con}[id_{sub}'][id_p])^2$
  $v_{sum} = \sum_{id_{sub}'} var_{sub}[id_{sub}'][id_p]$
  $var_{final-p}[id_p] = d_{sum}/v_{sum}$
**end**
return $grade_{final-con}$, $var_{final-p}$

## 4.4. Vancouver Algorithm Improvement

In this section, we first point out two parts in the original Vancouver algorithm that can be improved. Then, we propose our ways of improvement and evaluate the performance.

### 4.4.1. Peer variance prior

Original Vancouver algorithm calculates peer variance based on the difference between peer grades and estimated consensus grades. We observe that, especially for the first few iterations, Vancouver produces super high peer variance (several thousand). In reality, peer variances are usually low. Thus, if we could take the prior estimate of peer variance into consideration (**peer variance prior**), not only Vancouver will converge faster, but it will also produce more accurate estimation in the end.

We assume the peer grade generation process is as follows:

1. For peer $i$, sample $q_i$ from a normal distribution: $N(\mu, {\sigma_x}^2)$ (prior).

2. $q_i$ is the variance for peer $i$, i.e. the grade of assignment $j$ given by peer $i$ obeys normal distribution: $N(A_j, q_i)$, where $A_j$ is the ground truth score of assignment $j$.

Since peer $i$ grades multiple assignments, we further assume the observed $q_i'$s obeys normal distribution: $N(q_i, {\sigma_y}^2)$(likelihood). Since Vancouver estimates peer variance at each iteration, we then adjust these estimated peer variance using the prior per iteration.

According to Bayesian statistics, we have both prior and likelihood function to be normal. We assume the variance of the likelihood is known (${\sigma_y}^2$). Therefore, posterior is also a normal distribution (conjugate), which obeys: $N((\frac{1}{{\sigma_x}^2} + \frac{n}{{\sigma_y}^2})^{-1}(\frac{\mu}{{\sigma_x}^2} + \frac{\sum_{i=1}^{n} x_i}{{\sigma_y}^2}), (\frac{1}{{\sigma_x}^2} + \frac{n}{{\sigma_y}^2})^{-1})$. The prove can be found at [114].

### 4.4.2. Homework variance

Homework variance helps to estimate peer variance better, i.e. we should not punish a peer much for a high variance homework if the peer's estimation is much different from the consensus estimation. Actually, the homework variances come from the summation of peer variances. If we think about the source of the signal, homework variance is a duplicated signal with peer variance. Therefore, adding homework variances does not help with peer variance estimation. We propose to remove the homework variance weighting part (**plain weighting**). Algorithm 3 and 4 illustrate the improved Vancouver algorithm. (Using peer variance prior and plain weighting during peer variance computation).

In the first part of the algorithm, instead of computing peer variances by weighted average the squared errors based on submission variance, line 16-18 calculates the plain average of the squared errors. Line 19-20 adjusts the peer variances using prior peer

**Algorithm 3:** Improved Vancouver Algorithm Pseudo-code, part 1

```
1  Initialize all peer variances (p-var) to 1.0;
2  for id_iteration in range(0, Max Iterations) do
3      for each submission id := id_sub do
4          for each peer id who reviewed submission id := id_p do
5              var_sub[id_sub][id_p] = sum_{id_p' != id_p} var_p[id_p'][id_sub]
6          end
7      end
8      Initialize all consensus grade (grade_con[id_sub][id_p]) to zero.
9      for each submission id := id_sub do
10         for each peer id who reviewed submission id := id_p do
11             grade_con[id_sub][id_p] =
                 sum_{id_p' != id_p} grade_peer[id_sub][id_p'] * var_p[id_p'][id_sub] / var_sub[id_sub][id_p']
12         end
13     end
14     for each peer id := id_p do
15         for each submission id which is reviewed by id_p := id_sub do
16             var_p[id_p][id_sub] = sum_{id_sub' != id_sub} (grade_peer[id_sub'][id_p] -
                 grade_con[id_sub'][id_p])^2 * var_sub[id_sub'][id_p] /
17             v_sum = sum_{id_sub' != id_sub} 1.0
18             var_p[id_p][id_sub] = d_sum / v_sum
19             weight = sigma_y^2 / (sigma_y^2 + sigma_x^2 * v_sum)
20             var_p[id_p][id_sub] = weight * mu + (1 - weight) * var_p[id_p][id_sub]
21         end
22     end
23 end
24 return var_sub, grade_con, var_p
```

variance, $\mu$. We set the prior peer variance to be 10. The prior variance and the likelihood variance are all set to be 5.

Since submission variances are removed during peer variance computation, part 2 calculates the plain average of squared errors in line 8-10.

**Algorithm 4:** Improved Vancouver Algorithm Pseudo-code, part 2

**for** *each submission id* := $id_{sub}$ **do**
  $var_{final-sub}[id_{sub}] = \sum_{id_{p}'} var_p[id_{p}'][id_{sub}]$
**end**
**for** *each submission id* := $id_{sub}$ **do**
  $grade_{final-con}[id_{sub}] =$
  $\sum_{id_{p}'} grade_{peer}[id_{sub}][id_{p}'] * var_p[id_{p}'][id_{sub}]/var_{final-sub}[id_{sub}]$
**end**
**for** *each peer id* := $id_p$ **do**
  $d_{sum} = \sum_{id_{sub}'}(grade_{peer}[id_{sub}'][id_p] - grade_{con}[id_{sub}'][id_p])^2$
  $v_{sum} = \sum_{id_{sub}'} 1.0$
  $var_{final-p}[id_p] = d_{sum}/v_{sum}$
**end**
return $grade_{final-con}$, $var_{final-p}$

### 4.4.3. Vancouver results

We evaluate the original Vancouver algorithm and the improved Vancouver using the EECS-336 data set we collected at Northwestern University EECS department and our synthetic data set. Both the original Vancouver and the improved Vancouver run for 100 iterations on each data set. MSE (Mean squared error) are calculated based on the difference between Vancouver outputs and ground truth scores, which are TA grades for EECS-336 data and mean grades in synthetic data sets during data generation. We use the same prior mean and variance among all experiments.

Table 4.3-4.9 show the results.

| | plain weighting | homework variance weighting |
|---|---|---|
| Using peer variance prior | 263.14 | 256.26 |
| Not using peer variance prior | 288.98 | 281.71 |

Table 4.3. mean squared error comparison of 2019 Spring data

| | plain weighting | homework variance weighting |
|---|---|---|
| Using peer variance prior | 170.06 | 179.13 |
| Not using peer variance prior | 189.15 | 201.29 |

Table 4.4. mean squared error comparison of 2019 Fall data

| | plain weighting | homework variance weighting |
|---|---|---|
| Using peer variance prior | 180.25 | 175.34 |
| Not using peer variance prior | 197.82 | 192.52 |

Table 4.5. mean squared error comparison of 2017 Spring data

| | plain weighting | homework variance weighting |
|---|---|---|
| Using peer variance prior | 272.96 | 275.81 |
| Not using peer variance prior | 288.21 | 291.47 |

Table 4.6. mean squared error comparison of 2017 Fall data

| | plain weighting | homework variance weighting |
|---|---|---|
| Using peer variance prior | 101.20 | 103.16 |
| Not using peer variance prior | 102.97 | 106.36 |

Table 4.7. mean squared error comparison of syn-asymbias

The results show that MSE of using peer variance prior is much lower than not using peer variance prior for all EECS-336 data sets, syn-asymbias, and syn-symbias. For syn-unbias, using prior makes MSE worse. Note that peer grades of syn-unbias are unbiased. The results indicate that peer variance prior improves Vancouver accuracy when systematic bias exists. Also, we found that the MSE of using peer variance prior do not change much when using different prior mean and variance setting, which suggests that peer variance prior is a robust strategy to improve Vancouver accuracy.

We observe that MSEs decrease when using plain weighting strategy for most of the data set when peer variance is not used. However, when peer variance prior is activated, weighting strategy does not impact MSE much. This is also what we expected because the

| | plain weighting | homework variance weighting |
|---|---|---|
| Using peer variance prior | 43.51 | 44.99 |
| Not using peer variance prior | 44.40 | 46.13 |

Table 4.8. mean squared error comparison of syn-symbias

| | plain weighting | homework variance weighting |
|---|---|---|
| Using peer variance prior | 20.63 | 21.62 |
| Not using peer variance prior | 21.12 | 22.89 |

Table 4.9. mean squared error comparison of syn-unbias

prior mean and variance introduce new information to Vancouver, while plain weighting just removes duplicated signals in Vancouver.

Note that using peer variance prior actually decreases and smooth the peer variances. Similar effect can be achieved by taking the squared root of the original peer variances. Table 4.10 shows the result. We found that the square root approach surprisingly outperforms the peer variance prior approach. In addition, the square root approach does not need prior or other hyper-parameters. Thus, it seems to be a better way to improve Vancouver. So far, we are not sure the reason that it outperforms peer variance prior and will leave it for future study.

| | plain weighting | homework variance weighting |
|---|---|---|
| 2019 Spring | 241.18 | 244.99 |
| 2019 Fall | 136.98 | 135.07 |
| 2017 Spring | 167.08 | 156.42 |
| 2017 Fall | 248.61 | 246.71 |
| syn-asymbias | 91.94 | 92.14 |
| syn-symbias | 37.31 | 37.44 |
| syn-unbias | 19.36 | 19.39 |

Table 4.10. Using square root to smooth peer variance

Table 4.11 illustrates simple average baseline results.

| | Simple ave |
|---|---|
| 2019 Spring | 219.84 |
| 2019 Fall | 113.75 |
| 2017 Spring | 141.74 |
| 2017 Fall | 232.17 |
| syn-asymbias | 88.03 |
| syn-symbias | 38.58 |
| syn-unbias | 23.76 |

Table 4.11. Simple average baseline MSE of all data sets

We observe that the simple average baseline achieves better MSE in most data sets except syn-unbias. The reason is likely to be the biased peer scores. For EECS-336 data sets, we observe systematic peer score bias. A lot of peers overestimated the submission scores. On average, peers overestimated submission scores by 8.91 comparing to TA ground truth among the four EECS-336 data sets. [3] also argues that systematic bias causes the simple average baseline achieves better MSE than Vancouver algorithm.

## 4.5. Semi-automated Methods for Peer Grading

As is shown in the previous section Vancouver does not beat simple average baseline on our data. This section introduces our novel methods for peer grading algorithms.

### 4.5.1. Math notations

We first introduce math notations used in our algorithm. We assume that, in a class, $n$ students are given a sequence of $K$ homework $HW = [hw_1, hw_2, ...hw_K]$ in total. For homework $k$, every student makes a submission and is given a few other students' submissions to review. We use $[s_{sid_{k1}}, s_{sid_{k2}}, ..., s_{sid_{kn}}]$ to denote the submission list. $sid_{ki}$ is the id of the submission of student $i$ on homework $k$. In peer review, students (peers)

provide peer grades and textual reviews. We use the symbol $pg_{ij}$ and $r_{ij}$ to denote the peer grade and textual reviews given by peer $i$ to submission $j \in sid$. For each homework, instructor will grade a small portion of submissions and textual reviews. Grades provided by instructors are $ig_j$ and $rg_{ij}$, where $j$ is a submission and $i$ is the peer id. The goal of peer grading is to estimate the ground truth grade, $g_j$, for each submission $j$.

### 4.5.2. Peer bias estimation

Since peers are not well-trained graders, peer grades may not be accurate. For example, in the EECS-336 data we gathered for our experiments, 61% of the peer grades are higher than the instructor grades. That is, in our data the peers are biased and tend to overestimate the grades. We also observed that if a peer overestimates submissions in the past, they are likely to overestimate in the future. Figure 4.1, shows the number of over-estimated and under-estimated peer grades for all EECS-336 data. Each dot represents a peer. X-axis represents the number of over-estimated peer grades in a quarter. Y-axis is the number of under-estimated peer grades. The diagonal, $y = x$ is shown as the orange line. As we can see, peers are to either over-estimate peer grades ($x > y$) or under-estimate peer grades($y > x$). The statistical result shows that 61% of peer grades are higher than TA ground truth.

Also, peers are likely to behave similarly (either overestimate or underestimate) on submissions that are assigned in the same week. Table 4.12 to 4.15 shows the coefficients of L1 differences between peer grades and ground truth grades for two homework that are assigned in the same week for EECS-336. As we can see, most of the coefficients are

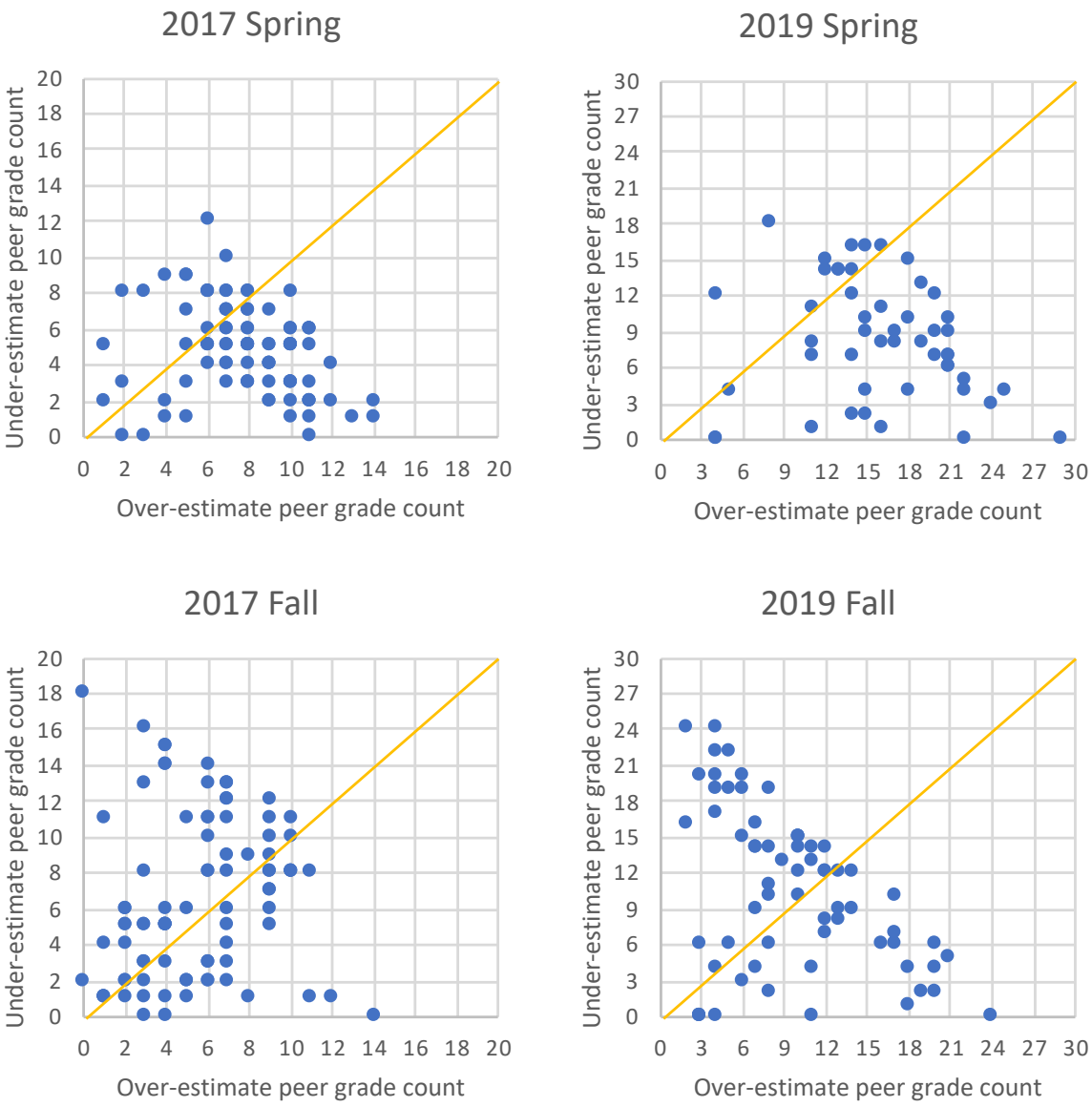


Figure 4.1. Number of over-estimated and under-estimated peer grades. Each dot represents a peer. Most of the peers are to either over-estimate peer grades ($x > y$) or under-estimate peer grades($y > x$). The statistical result shows that 61% of peer grades are higher than TA ground truth.

positive, which means if a peer over-estimate or under-estimate one of homework, he or she will likely to over or underestimate the other homework as well.

| | week 1 | week 2 | week 3 | week 6 | week 7 |
|---|---|---|---|---|---|
| coef | 0.1402 | 0.0159 | -0.0214 | 0.2827 | 0.3722 |

Table 4.12. L1 difference coefficients of different homework within a week. 2019 Spring.

| | week 1 | week 2 | week 3 | week 6 | week 9 |
|---|---|---|---|---|---|
| coef | 0.1684 | 0.3348 | 0.2258 | 0.1250 | 0.5140 |

Table 4.13. L1 difference coefficients of different homework within a week. 2019 Fall.

| | week 1 | week 2 | week 3 | week 4 | week 5 | week 6 | week 8 |
|---|---|---|---|---|---|---|---|
| coef | -0.1205 | -0.0672 | 0.2762 | 0.1287 | 0.2078 | 0.0671 | 0.3521 |

Table 4.14. L1 difference coefficients of different homework within a week. 2017 Spring.

| | week 1 | week 2 | week 3 | week 7 | week 8 | week 9 |
|---|---|---|---|---|---|---|
| coef | -0.2681 | 0.3849 | 0.2267 | 0.2918 | 0.4404 | 0.1155 |

Table 4.15. L1 difference coefficients of different homework within a week. 2017 Fall.

Based on the above observations, we model peer bias using a limited amount of historical instructor submission ground truth grades. This method first estimates bias for each peer by averaging the difference between the historical peer grades and the corresponding instructor grades. Then, it subtracts peer biases from peer grades and averages them as the estimated consensus grades. Formally:

$$b_i = \frac{\sum_{j' \in D}(pg_{ij'} - ig_{j'})}{|D|} \tag{4.1}$$

$$\hat{g}_j = \frac{\sum_{i' \in E}(pg_{i'j} - b'_i)}{|E|} \tag{4.2}$$

Equation 4.1 and 4.2, describe how consensus grades are computed for homework $k$. In equation 4.1, $D$ is the set of submission ids that are graded by both peer $i$ and the instructor for homework 1 to $k-1$. $b_i$ is the estimated bias of peer $i$ (this is the maximum likelihood estimate assuming biases are e.g. Gaussian distributed). Equation 4.2 computes the estimated consensus grade of submission $j$ from homework $k$. $E$ is the peer set that reviewed submission $j$. We name this method SAB (Semi-Automated Peer Bias).

To evaluate SAB, we compare SAB with simple average, Vancouver, and MALS algorithm (Score-Weighted Mallows) proposed in [94] on EECS-336 data and synthetic data. Note that ideally we should have done model development and selection on one data set, and then evaluated on a completely disjoint data. Because the data is very expansive to obtain, we were force to develop and test using the data set we have. Table 4.16 show the results. Note that we now use a 2-way cross-validation on 2019 Spring and Fall data, since we need to split TA ground truth into training and testing for the supervised textual review quality methods proposed in later subsections.

| | SAB | simple ave | Vancouver | MALS |
|---|---|---|---|---|
| 2019 Spring | **171.61** | 210.04 | 276.93 | 209.83 |
| 2019 Fall | **112.71** | 113.54 | 192.77 | 113.59 |
| 2017 Spring | **138.13** | 141.74 | 192.52 | 139.65 |
| 2017 Fall | **228.44** | 232.17 | 291.47 | 229.34 |
| syn-asymbias | **39.83** | 88.03 | 106.36 | 86.69 |
| syn-symbias | **27.99** | 38.58 | 46.13 | 37.80 |
| syn-unbias | 23.44 | 23.76 | **22.89** | 22.94 |

Table 4.16. MSE comparison of SAB, simple average, Vancouver and MLAS on all data sets.(Lower is better) SAB outperforms other methods in most of the data sets

As shown, SAB achieves lower MSEs than other methods for most of the data sets. This indicates that by using previous instructor ground truth, SAB can accurately model peer biases and estimates consensus grades. Vancouver performs the best on syn-unbias, showing that it is most effective when data happens to be unbiased. But Vancouver's poor performance on the other data sets suggests that the unbiased assumption is too strong for real classroom data.

### 4.5.3. Homework bias estimation

Our previous model, SAB, assumes the grading bias only comes from peers. However, homework difficulty differs. Some homework may be hard to grade, while some are easy. Thus, homework may introduce bias as well. We now assume the bias comes from two parts, homework bias and peer bias. Homework bias can be estimated using the difference between average peer scores and TA scores. We view the residual difference as peer bias. Thus, for each peer, we compute a peer bias, and each homework corresponding to a homework bias.

$$hb_k = \frac{\sum_{j' \in F} (pg_{ij'} - ig'_j)}{|F|} \tag{4.3}$$

Equation 4.3 shows how bias of homework $k$, $hb_k$, is computed. During consensus scores estimation, we use part of the current instructor ground truth $F$ to compute the homework bias and the rest ground truth scores are used for evaluation purpose. We call this method SAB(HW).

| | SAB(HW) | SAB | simple ave |
|---|---|---|---|
| 2019 Spring | **110.62** | 171.61 | 210.04 |
| 2019 Fall | 173.80 | **112.71** | 113.54 |
| 2017 Spring | 277.24 | **138.10** | 141.74 |
| 2017 Fall | 284.96 | **228.75** | 232.17 |
| syn-asymbias | 44.87 | **33.24** | 88.03 |
| syn-symbias | 44.60 | **26.38** | 38.58 |
| syn-unbias | 25.68 | **23.49** | 23.76 |

Table 4.17. MSE comparison of SAB(HW), SAB, simple average. (Lower is better). SAB(HW) does not beat SAB on most data sets except 2019 Spring.

We compare the proposed homework bias method SAB(HW) with SAB and simple average baseline in Table 4.17. MSEs of other baseline models can be found in Table 4.16. Table 4.17 shows that the MSE of homework bias model is the best only on 2019 Spring data. A further investigation shows that there is a homework, where the peer grades are much higher than ground truth grades by 25 to 40. This homework leads to very high MSEs for SAB and simple average. This homework happened when students learn a new and hard to understand the topic. Thus, the TA scores are low (about 40s to 50s). However, peers tend to provide similar scores as before (70s to 90s). Since homework bias is able to detect such an outlier by using a part of ground truth scores of the current submission, it achieves very low MSE on this homework comparing to SAB, which results in low average MSE. Other than 2019 Spring, SAB is still the best model for both real and synthetic data sets. Thus, our conclusion is that homework bias is applicable when peer scores are very different than TA scores, but in general this phenomenon is rare. Due to this, we do not include homework bias in the following experiments.

### 4.5.4. Textual review quality estimation

In this subsection, we consider using textual peer reviews to improve peer grading performance. Textual reviews reflect how much effort a peer spends on peer reviewing and how well he or she understands the submission. Intuitively, peers who provide good textual reviews are likely to provide more accurate peer grades. Our idea is to increase the weights of peer grades that corresponding to high-quality textual reviews and down-weight the peer grades of bad ones. This subsection proposes several methods to model the quality of peer reviews using textual comments and puts more weight on good peer reviews when computing consensus grades. Note that quality is determined by how much effort a peer spent and how skillful is the peer. In this thesis, we only focusing on measuring the effort part.

We now present our methods for using textual review comments to improve peer grading performance. Formally, all of our models estimate a textual review quality $r_{ij}$ for peer $i$ and submission $j$, and linearly map it to the range $[-\tau, \tau]$ as the weight $w_{ij}$. We set $tau = 0.1$ in this work. Equation 4.4 computes consensus peer grades using the weights:

$$\hat{g_{new}} = \frac{\sum_{i \in E}(pg_{ij} - b_i) * (1.0 + w_{ij})}{|E|} \tag{4.4}$$

$$w_{ij} = \frac{len(r_{ij}) - (len_{max} - len_{min})/2.0}{\frac{1.0}{\tau} * (len_{max} - len_{min})} \tag{4.5}$$

**4.5.4.1. Using review length to estimate peer review quality.** We propose a simple yet effective method, SABTXT (Semi-Automated peer Bias grading approach with TeXTual reviews), that estimates review quality using the length of textual review content. SABTXT trains a linear regression model to learn historical relation between review length and peer grading accuracy. The independent variables are $[len(r_j^i), len(r_j^i)^2, \sqrt{len(r_j^i)}]$. The dependent variable is $\frac{1.0}{(pg_j^i - ig_j)^2 + 1.0}$. To predict review thoroughness, we linearly map the range of dependent variable to $[-\tau, \tau]$. Since there is no historical data for the first homework, SABTXT linearly maps review lengths to weights as is shown in Equation 4.5, where $len_{max}$ and $len_{min}$ represent the maximum and minimum review length.

**4.5.4.2. Using BERT to estimate peer review quality.** Sometimes we have TAs' evaluation on the quality of textual peer reviews. In such cases, it is possible to use those evaluations to train a supervised model of peer review quality. Bidirectional Encoder Representations from Transformers(BERT)[24], is a widely used method of pre-training language representations which obtains state-of-the-art results on a wide array of Natural Language Processing tasks, including General Language Understanding Evaluation (GLUE)[118], Stanford Question Answering Dataset (SQuAD)[92], Situations With Adversarial Generations (SWAG)[128], etc. To predict peer review text quality (peer review quality for short), we fine tune the pre-trained BERT model (Base, Uncased) using the peer reviews as the input and peer review quality (TA grades) as the output. Specifically, after feeding peer reviews into BERT, we take the last output layer and linear map to the peer review grades provided by TA. We tried three different loss functions: softmax

cross-entropy loss, sigmoid cross-entropy loss, and mean squared error loss. Figure 4.2 illustrates the basic architecture of our model.

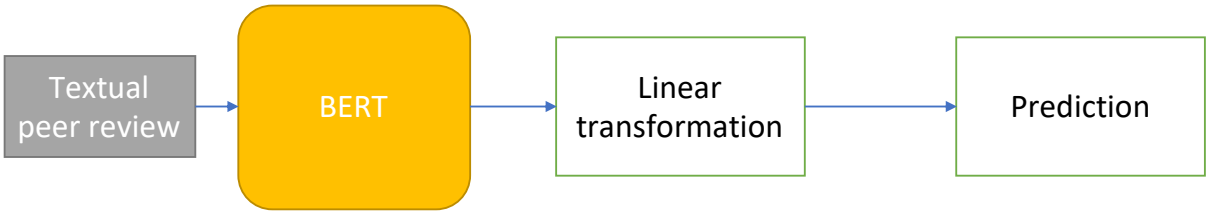


Figure 4.2. basic architecture of peer review quality prediction model using BERT

We consider the following strategies to improve BERT prediction results.

**1:** Classification loss function

Since BERT is originally designed for classification tasks, instead of MSE loss (regression), we could use cross-entropy loss, which is widely used for classification tasks. To train our model, we convert float output scores into integer types. Since the scale of review scores is from 0 to 10, there are 11 different classes. Each score corresponds to one class.

**2:** LM fine tuning

BERT is pre-trained on two tasks: masked language model and next sentence prediction[24] using BooksCorpus [136] and English Wikipedia. Since sentences in peer review are different from BooksCorpus and English Wikipedia, we could continue to pre-train BERT (base, uncased) on our peer review data set beforehand. This is called LM fine tuning (different from fine tuning mentioned before). The intuition of LM fine tuning is to warm BERT up by feeding peer review sentences, which will be used during fine tuning.

**3:** Adding peer grades

Peer grades might provide useful information to predict review quality. For example, if the peer grade is low, high quality peer review should justify the reason for the low peer grades. Thus, we propose to use both peer grades and textual reviews to predict textual review quality. Specifically, we concatenate peer grades strings with the peer review as the input to BERT. Figure 4.3 shows the architecture of the proposed model.

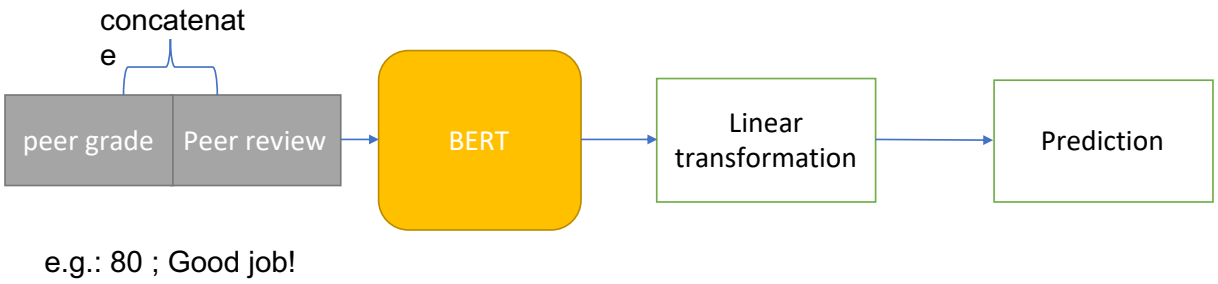


Figure 4.3. concatenate peer grades embeddings

Table 4.18 and 4.19 evaluate the effect of approaches proposed above. We evaluate the review quality score prediction performance using MSE and l1 error on 2019 Spring and Fall data. We randomly select 80% of TA graded peer reviews for training 10% for validation and 10% for testing purpose. We randomly selected three different training, validation, and testing data. Average mean squared error, l1 error and accuracy are reported. We compared our models with zero-r baseline and average review score baseline. Zero-r baseline simply outputs the most frequent integer score. Average review score baseline outputs the average review score of the training data. Table 4.18 and 4.21 show MSEs, L1 errors and accuracies of all different BERT settings on 2019 Spring and 2019 Fall data. Table 4.20 and 4.21 show the baseline performance.

| 2019 Spring | | Softmax cross-entropy loss | | | Sigmoid cross-entropy loss | | | Mean squared error loss | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | MSE | L1 | ACC | MSE | L1 | ACC | MSE | L1 | ACC |
| Original BERT | W/O peer grades | 6.274 | 1.856 | 0.199 | 5.556 | 1.728 | 0.237 | 4.510 | 1.634 | N/A |
| | Peer grades | 5.896 | 1.785 | 0.212 | 4.800 | 1.593 | 0.244 | 4.765 | 1.726 | N/A |
| LM fine-tuned BERT | W/O peer grades | 6.383 | 1.888 | 0.192 | 5.556 | 1.702 | 0.231 | 4.707 | 1.657 | N/A |
| | Peer grades | 4.902 | 1.631 | 0.218 | 6.300 | 1.830 | 0.192 | 4.796 | 1.728 | N/A |

Table 4.18. MSE, L1 error and accuracy comparison of different BERT setting on 2019 Spring data.

We observe that the models that using mean squared error loss achieves much lower MSE than softmax and sigmoid cross-entropy loss. Using sigmoid cross-entropy loss, BERT achieves better prediction accuracy than softmax in general. Also, we observe that lower MSE does not necessarily mean lower L1 error. Comparing the models using sigmoid cross-entropy and mean squared error for 2019 Fall data. Mean squared error loss models achieve lower MSE but higher L1 error, while sigmoid cross-entropy loss models achieve higher MSE but lower L1. We found that LM fine-tuned model performs worse than the original model. This observation is different from [42, 37], where fine-tuning model using domain related corpus improves classification accuracy. We believe that the reason is that the number of training examples is small, less than 500. Thus, it is hard to fine tune BERT on small data sets.

| 2019 Fall | | Softmax cross-entropy loss | | | Sigmoid cross-entropy loss | | | Mean squared error loss | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | MSE | L1 | ACC | MSE | L1 | ACC | MSE | L1 | ACC |
| Original BERT | W/O peer grades | 3.348 | 0.966 | 0.569 | 3.147 | 0.941 | 0.539 | 2.742 | 1.029 | N/A |
| | Peer grades | 3.279 | 0.926 | 0.574 | 3.216 | 0.941 | 0.549 | 2.680 | 1.023 | N/A |
| LM fine-tuned BERT | W/O peer grades | 3.471 | 1.069 | 0.475 | 3.103 | 0.956 | 0.505 | 2.721 | 1.002 | N/A |
| | Peer grades | 3.289 | 0.966 | 0.525 | 3.319 | 0.975 | 0.539 | 2.728 | 1.012 | N/A |

Table 4.19. MSE, L1 error and accuracy comparison of different BERT setting on 2019 Fall data.

| | MSE | L1 | ACC |
|---|---|---|---|
| Zero-R | 4.896 | 1.638 | 0.218 |
| Average | 4.811 | 1.739 | N/A |

Table 4.20. MSE, L1 error and accuracy of baselines on 2019 Spring data

| | MSE | L1 | ACC |
|---|---|---|---|
| Zero-R | 3.397 | 0.917 | 0.608 |
| Average | 2.597 | 1.072 | N/A |

Table 4.21. MSE, L1 error and accuracy of baselines on 2019 Fall data

The best BERT setting for 2019 Spring data in terms of L1 error and accuracy is using original BERT, sigmoid cross-entropy loss, and peer grades feature, which achieves the L1 error and accuracy of 1.593 and 0.244 separately. Original BERT, mean squared error loss without peer grades is the best setting that achieves the lowest MSE. The best setting outperforms both baselines for 2019 Spring data. For 2019 Fall data, in terms of

MSE, the original BERT, MSE loss with peer grades feature is the best. Original BERT, softmax cross-entropy loss with peer grades feature achieves the best L1 error of 1.593. Surprisingly, MSE of average baseline and L1 error of Zero-R baseline beats BERT models. We guess that due to the fact that we only have a limited amount of training examples, it is not easy to train BERT well. General speaking, LM fine-tuning, and adding peer grades does not improve prediction performance much.

Back to peer review quality prediction, we use original BERT, sigmoid cross-entropy loss without peer grade. We train BERT model using a part of ground truth review scores and use the rest data for validation and testing purpose. We call this approach SABTXT(BERT). SABTXT(BERT) computes weights using Equation 4.6.

$$w_{ij} = \frac{\widehat{rg}_{ij} - 5.0}{1.0/\tau} \tag{4.6}$$

**4.5.4.3. Using Bag-Of-Words vector to estimate peer review quality.** BERT is a powerful model but has over a hundred million parameters. Our data collected from EECS-336 may be too sparse to train it effectively. Thus, we also build a smaller model that takes Bag-Of-Words vectors of reviews as input and feeds them into a one-layer neural network (i.e. a logistic regression model) to predict instructor grades on reviews. We name this model SABTXT(BOW). Like SABTXT(BERT), SABTXT(BOW) uses Equation 4.6 to compute weights $w_{ij}$ as well.

**4.5.4.4. Textual review quality estimation results.** Table 4.22 compares different textual review quality estimation methods, including SABTXT, SABTXT(BERT), and

SABTXT(BOW). Note that since the instructor did not evaluate reviews in 2017 Spring and Fall, supervised results are not available.

Surprisingly, the length-based method in SABTXT outperforms the two supervised models. We further compare SABTXT with a model that linearly maps the ground truth instructor grades on reviews to the review weights, which can be considered as an upper bound for the supervised models. Surprisingly, SABTXT even beats the upper bound. The result suggests that review length is a better indicator of appropriate weight than are the instructor grades on reviews. Also, the length-based method does not require any instructor evaluation of textual reviews, increasing practicality.

Table 4.23 compares SABTXT with other baselines. The results show that SABTXT (1st column) achieves lower MSEs than the other methods for nearly all of the data sets. The MSEs of 2019 Spring and 2017 Fall are higher than the other data sets. This is due to a handful of assignments in those courses that cover newly introduced topics. In those cases, instructor grades are much lower than the peer grades, which causes higher average MSEs for all methods. This suggests we could potentially improve our methods by accounting for how bias may be higher for more challenging, unfamiliar assignments, and this is an item of future work.

| | SABTXT | SABTXT(BERT) | SABTXT(BOW) |
|---|---|---|---|
| 2019 Spring | 171.24 | **166.37** | 233.76 |
| 2019 Fall | **111.15** | 111.88 | 112.71 |
| 2017 Spring | **137.14** | N/A | N/A |
| 2017 Fall | **227.63** | N/A | N/A |

Table 4.22. Text review quality weighting methods comparison.

To test how the amount of historical instructor grades affects SABTXT, we randomly select different percentages of historical grades for peer bias estimation. We plot

| | SABTXT | simple ave | Vancouver | MALS |
|---|---|---|---|---|
| 2019 Spring | **171.24** | 210.04 | 276.93 | 209.83 |
| 2019 Fall | **111.15** | 113.54 | 192.77 | 113.59 |
| 2017 Spring | **137.14** | 141.74 | 192.52 | 139.65 |
| 2017 Fall | **227.63** | 232.17 | 291.47 | 229.34 |
| syn-asymbias | **39.83** | 88.03 | 106.36 | 86.69 |
| syn-symbias | **27.99** | 38.58 | 46.13 | 37.80 |
| syn-unbias | 23.44 | 23.76 | **22.89** | 22.94 |

Table 4.23. Mean Squared Error of grade estimation for SABTXT, simple average, Vancouver and MLAS on all data sets. SABTXT outperforms baseline methods on all data sets, except the unbiased synthetic data set syn-unbias.

the average MSE over the classroom and synthetic data in Figure 4.4 against the two best-performing baselines. Figure 4.4 shows that MSE improves given more historical instructor grades, although the improvement tapers off at about 60% of the data. Moreover, even without instructor data, SABTXT outperforms the baselines.

### 4.5.5. Peer review quality co-training

According to the results in the previous section, training supervised models using TA evaluation on peer review quality did not beat SABTXT, which just takes review length to re-weight peer grades. The reasons are probably there are not enough training examples to train neural models, or TA grades may be inaccurate sometimes. To overcome these two shortcomings, this section proposes an iterative method to predict the quality of textual peer reviews.

This method adopts the idea of co-training [10]. Instead of predicting TA grades, this method directly predicts weights of peer reviews given textual reviews. This method first estimates the weights of textual reviews using the differences between peer grades and

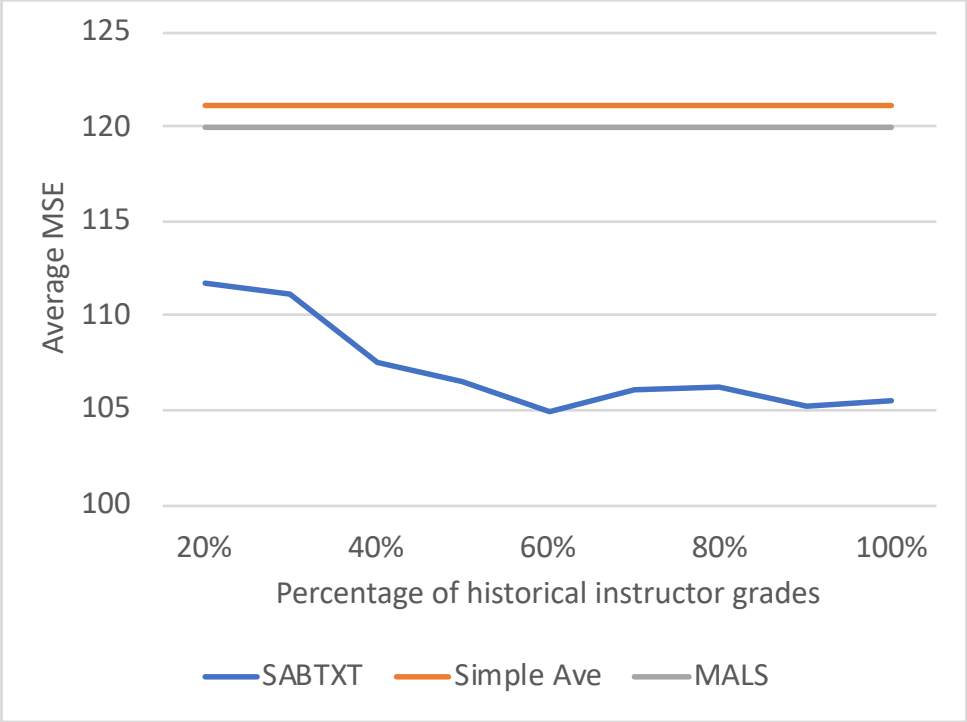


Figure 4.4. MSEs of SABTXT using different amount of historical instructor grades. MSE drops as historical instructor grades increases.

estimated consensus grades. The small differences mean high weights. We then select a part of textual reviews and the corresponding weights to train a neural model (either BERT or Bag-Of-Word model) and use the prediction of the model to update the weights of the rest. We then compute the consensus grades using new weights. And estimate the weight for textual reviews again. Algorithm 5 shows the procedure for the co-training method at a high-level.

Note that this co-training method selects a part of data to train a model and updates the weights of the rest. Intuitively, we want the training data and validation to be good quality, so that after training, we will have a good model to update the test data. One way to select good quality data is to select peers that behave constantly. We measure

**Algorithm 5:** Co-training algorithm pseudo code

```
Initialize all weights to be 1.0
for iteration in [1,2,3...] do
    compute consensus grades based on weights
    compute weights based on consensus grades
    select part of weights to train a neural network model and update the rest
end
return consensus grades
```

the quality of peers by first calculating the distance between peer grades and ground truth grade and sort peers by the variance of distances. Peers who have low variance means their behavior is relatively constant and can be considered as good quality training examples. We tried two different strategies to split the data. The first strategy is to sort peers according to the variances mentioned. Then, it selects peer reviews belong to peers who have low variances for training and validation data. The rest data is the peers have high variances. The second strategy is to random split data into training, validation, and testing parts.

To evaluate the performance of co-training, we use the data we collected in EECS-336. For each data set, we tried three different loss functions: sigmoid loss, softmax loss, and MSE loss. For each setting, we train 10 iterations and plot the best test MSE. Note that this experiment is to test the best possible performance of co-training method, which is the lower-bound of MSE. Due to very long time training and lack of data, it is not practical to do cross-validation on this experiment. Table 4.24 to 4.27 show the results. For each setting, we show the MSE of the SABTXT (with review length) and SAB (without review length), the best MSE of the co-training and the best iteration. Note

that weights are updated at the beginning of an iteration. Thus, iteration 0 is actually the MSE of SABTXT or SAB.

| | | With textual review length | | | Without textual review length | | |
|---|---|---|---|---|---|---|---|
| | | softmax | sigmoid | mse | softmax | sigmoid | mse |
| 2019 Spring | SAB(TXT) mse | 157.321 | 157.321 | 157.321 | 172.645 | 172.645 | 172.645 |
| | best mse | 157.321 | 157.321 | 157.321 | 172.645 | 172.645 | 171.689 |
| | best iter | 0 | 0 | 0 | 0 | 0 | 6 |
| 2019 Fall | SAB(TXT) mse | 121.361 | 121.361 | 121.361 | 117.971 | 117.971 | 117.971 |
| | best mse | 121.361 | 121.361 | 121.361 | 117.971 | 117.971 | 117.971 |
| | best iter | 0 | 0 | 0 | 0 | 0 | 0 |
| 2017 Spring | SAB(TXT) mse | 136.780 | 136.780 | 136.780 | 138.101 | 138.101 | 138.101 |
| | best mse | 136.780 | 136.780 | 136.780 | 138.101 | 138.101 | 137.972 |
| | best iter | 0 | 0 | 0 | 0 | 0 | 5 |
| 2017 Fall | SAB(TXT) mse | 224.579 | 224.579 | 224.579 | 228.752 | 228.752 | 228.752 |
| | best mse | 223.749 | 223.573 | 223.552 | 227.577 | 227.874 | 228.129 |
| | best iter | 7 | 2 | 3 | 8 | 5 | 4 |

Table 4.24. Co-training using Bag-Of-Word model, sort peers

| | | With textual review length | | | Without textual review length | | |
|---|---|---|---|---|---|---|---|
| | | softmax | sigmoid | mse | softmax | sigmoid | mse |
| 2019 Spring | SAB(TXT) mse | 157.321 | 157.321 | 157.321 | 172.645 | 172.645 | 172.645 |
| | best mse | 157.321 | 157.321 | 156.838 | 172.587 | 172.645 | 172.481 |
| | best iter | 0 | 0 | 3 | 3 | 0 | 7 |
| 2019 Fall | SAB(TXT) mse | 121.361 | 121.361 | 121.361 | 117.971 | 117.971 | 117.971 |
| | best mse | 121.196 | 121.230 | 120.959 | 117.383 | 117.772 | 117.608 |
| | best iter | 2 | 8 | 8 | 8 | 1 | 2 |
| 2017 Spring | SAB(TXT) mse | 136.780 | 136.780 | 136.780 | 138.101 | 138.101 | 138.101 |
| | best mse | 135.680 | 135.564 | 135.600 | 136.487 | 136.679 | 137.147 |
| | best iter | 7 | 6 | 7 | 7 | 6 | 6 |
| 2017 Fall | SAB(TXT) mse | 224.579 | 224.579 | 224.579 | 228.752 | 228.752 | 228.752 |
| | best mse | 223.608 | 223.234 | 223.077 | 227.070 | 226.850 | 227.358 |
| | best iter | 9 | 4 | 2 | 2 | 1 | 2 |

Table 4.25. Co-training using Bag-Of-Word model, random split

| | | With textual review length | | | Without textual review length | | |
|---|---|---|---|---|---|---|---|
| | | softmax | sigmoid | mse | softmax | sigmoid | mse |
| 2019 Spring | SAB(TXT) mse | 157.321 | 157.321 | 157.321 | 172.645 | 172.645 | 172.645 |
| | best mse | 157.321 | 157.321 | 157.321 | 172.645 | 172.418 | 172.188 |
| | best iter | 0 | 0 | 0 | 0 | 4 | 6 |
| 2019 Fall | SAB(TXT) mse | 121.361 | 121.361 | 121.361 | 117.971 | 117.971 | 117.971 |
| | best mse | 121.361 | 121.361 | 121.361 | 117.971 | 117.971 | 117.908 |
| | best iter | 0 | 0 | 0 | 0 | 0 | 2 |
| 2017 Spring | SAB(TXT) mse | 136.780 | 136.780 | 136.780 | 138.101 | 138.101 | 138.101 |
| | best mse | 136.780 | 136.780 | 136.780 | 137.971 | 137.049 | 136.827 |
| | best iter | 0 | 0 | 0 | 1 | 1 | 7 |
| 2017 Fall | SAB(TXT) mse | 224.579 | 224.579 | 224.579 | 228.752 | 228.752 | 228.752 |
| | best mse | 223.640 | 223.306 | 223.686 | 227.685 | 227.443 | 227.010 |
| | best iter | 3 | 4 | 3 | 6 | 5 | 4 |

Table 4.26. Co-training using BERT, sort peers

| | | With textual review length | | | Without textual review length | | |
|---|---|---|---|---|---|---|---|
| | | softmax | sigmoid | mse | softmax | sigmoid | mse |
| 2019 Spring | SAB(TXT) mse | 157.321 | 157.321 | 157.321 | 172.645 | 172.645 | 172.645 |
| | best mse | 157.097 | 156.856 | 156.652 | 172.645 | 172.369 | 172.622 |
| | best iter | 5 | 3 | 4 | 0 | 4 | 9 |
| 2019 Fall | SAB(TXT) mse | 121.361 | 121.361 | 121.361 | 117.971 | 117.971 | 117.971 |
| | best mse | 121.113 | 120.949 | 120.893 | 117.280 | 117.784 | 117.712 |
| | best iter | 1 | 8 | 9 | 8 | 1 | 1 |
| 2017 Spring | SAB(TXT) mse | 136.780 | 136.780 | 136.780 | 138.101 | 138.101 | 138.101 |
| | best mse | 135.417 | 136.083 | 135.362 | 136.763 | 137.145 | 136.328 |
| | best iter | 7 | 7 | 7 | 7 | 8 | 7 |
| 2017 Fall | SAB(TXT) mse | 224.579 | 224.579 | 224.579 | 228.752 | 228.752 | 228.752 |
| | best mse | 223.297 | 223.373 | 225.502 | 227.157 | 227.127 | 226.270 |
| | best iter | 6 | 3 | 1 | 2 | 2 | 1 |

Table 4.27. Co-training using BERT model, random split

As is shown, for some settings, co-training achieves better MSE than SABTXT or SAB, which indicates that using co-training method, it is possible to improve the performance of consensus grade estimation. Most settings that sorted peers according to variances

do not get lower MSE than SABTXT or SAB. We believe that the reason is that the weights of training data are not updated during co-training. Consequently, it is likely that at each iteration, the model selects the same or similar for training purpose. Thus, the performance of neural network is not improved through co-training. The results also show that a lot of experiments that adopting random training, validation, and test data split strategy achieve lower MSE comparing to other settings. This implies that random data split allows neural network to explore more data and results in better prediction accuracy. However, we found that the overall MSE improvement of using co-training is small. Also, neural network models are trained for every iteration, which causes a long time to get the final result. Thus, it is hard to adopt co-training in real scenarios.

## 4.6. Peer Review Process and Students' Behavior

Note that the peer review processes of the four EECS-336 data sets we collected in this chapter are different. For 2017 Spring and Fall quarter, peers provided peer grades and TA graded some submissions. Then, peers are graded using Mechanical TA [121]. The basic idea of Mechanical TA is to give higher grades to peers that provide accurate peer grades.

For the two quarters in 2019, peers are asked to grade assignments based on three different rubric elements: correctness of the algorithm, correctness of the prove and clearness of the writing. The final peer grade is the summation of them. TAs also grade these three aspects on some of the assignments. Similarly, peer grades are automatically graded using Mechanical TA. Besides peer grades, peers were asked to provide comments to justify their peer grades or provide constructive suggestions for assignments. TAs evaluated

the quality of textual peer reviews and grade textual reviews. Table 4.28 compares the peer review processes of EECS-336 data sets.

| Data Set | Text review | Evaluation |
|---|---|---|
| 2019 Spring | Rubric | Y |
| 2019 Fall | Rubric | Y |
| 2017 Spring | Overall | N |
| 2017 Fall | Overall | N |

Table 4.28. Peer review process comparison

This section investigates the students' behavior of different peer review processes. Specifically, this section answers the following questions:

**1:** What kind of peer review process encourages peers to provide better peer reviews?

**2:** For these peer review processes, is there a difference in textual review content?

Intuitively, we could estimate the time a peer spent on peer review by checking the length of the corresponding textual peer review. Longer reviews indicate that peers put more effort in peer review. Figure 4.5 plots the average review length for each homework. Note that homework IDs are in time order, homework 1 is the first homework in a quarter, homework 2 is the second one, so on and so forth. As we can see, the review lengths of 2019 Spring and Fall quarter are much longer than 2017 Spring and Fall. This indicates peers were likely to spent more time on peer review in 2019 than the two quarters in

2017. There are two possible reasons for the longer reviews of 2019. First, instead of providing a single review for each submission, peers were asked to provide feedback on three rubric elements in 2019. Peer reviewers were given more specific instructions on peer review. This may increase the length of textual reviews. Second, unlike 2017, the quality of textual reviews were human evaluated and graded by TA. This motivated peers to spend more time and provide longer reviews.

Figure 4.6 shows the percentage of short reviews for each homework. We define short reviews to be reviews that consist of 5 or fewer characters. Those reviews are usually empty reviews or reviews that are too general, such as "good", "great", etc. Counting short review is another way to measure the amount of effort peers spend on reviewing. Figure 4.6 shows the difference between the two quarters in 2017 and the two quarters in 2019. There are more empty reviews in 2017 than that of 2019. What's more, comparing to 2019 Spring and Fall, there is a clear trend that peers gave more and more empty reviews in 2017 Spring and Fall. Especially for the last few homework, about 60% to 80% reviews are short reviews. We believe that the reason for the difference is due to the fact that textual reviews were not graded in 2017. Consequently, after they realized they won't get credit for peer review text, they were reluctant to spend much time on providing textual reviews. Our conclusion is that by having TA evaluate the quality of reviews, peers are likely to provide longer reviews.

We now propose an unsupervised method to compare peer review content specificity of 2017 and 2019. Note that this unsupervised peer review quality evaluation method is not designed to evaluate the quality of a specific peer review but the overall content specificity of a homework, although content specificity partially correlates to review quality. This

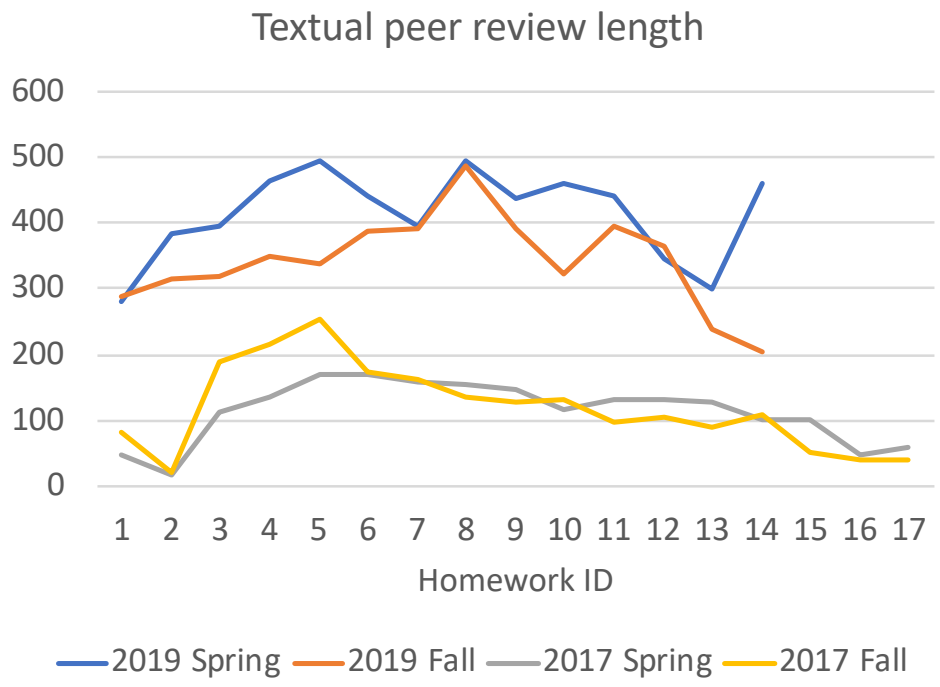


Figure 4.5. Average textual review length (number of characters) for each homework.

unsupervised method assumes that good peer reviews should be specific to the submission and point out major issues. Low quality peer reviews are usually very general, such as 'good job!'. Thus, the unsupervised task we propose here is that for each homework rubric, say there are $n$ peer reviews, and we build a $n^2$ matrix $M$. $M[i][j] == 1$ means peer review $i$ and $j$ belongs to the same submission and $M[i][j] == 0$ means peer review $i$ and $j$ belongs to different submissions. Then, a model is built to predict matrix elements. The intuition is that it's hard to predict the results for low quality peer reviews because low quality reviews are very general. If both peer reviews are high quality, the model is able to tell if they belong to the same submission or not by checking whether they talk about similar issues. Note that this unsupervised method enlarges the size of data from

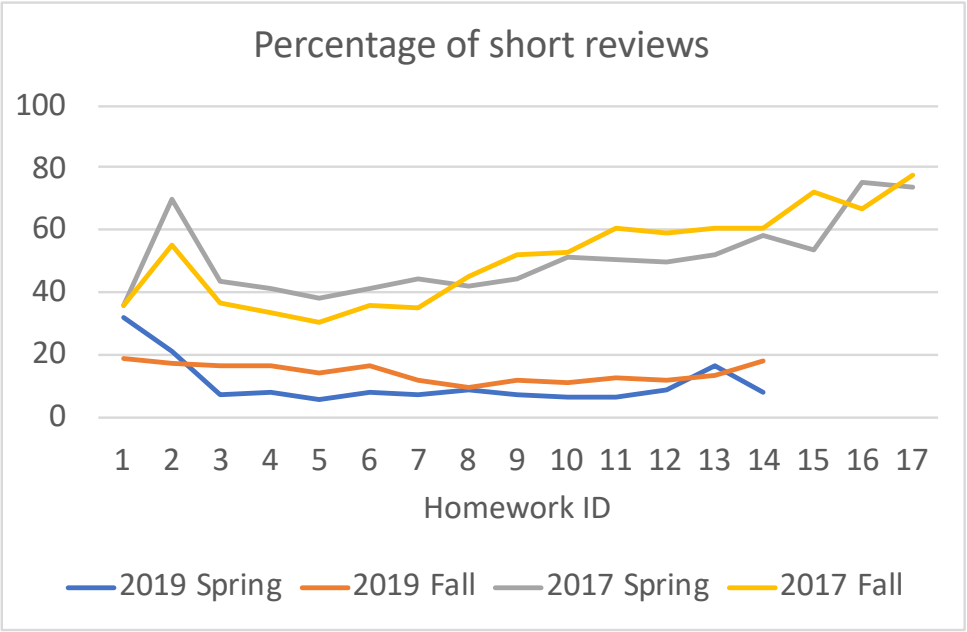


Figure 4.6. Short review percentage per homework. 2017 Spring and Fall have more short reviews than 2019 Spring and Fall. Also, peers gave more and more short reviews in 2017 Spring and Fall, while for 2019 Spring and Fall, the short review percentage roughly keeps the same level.

$n$ to $n^2$ for each homework, which enlarged the training data set by a lot. In addition, unlike the supervised models mentioned above, this unsupervised model doesn't require TA assessments on textual peer reviews.

We train BERT model to do the proposed unsupervised task. The BERT model takes two reviews as input. The binary output indicates whether they belong to the same submission or not. Note that peers are asked to provide grades and reviews on three aspects of the algorithm in 2019, while only one peer grade and a single overall review is required in 2017. Since BERT is GPU memory demanding, we have to train three different models for different rubric elements separately. Each homework has a different number of submissions, and each submission received different submissions received a different

number of reviews. To get a fair comparison between homework within a quarter, we randomly drop some submissions and reviews so that each homework has the same number of submissions, and each submission has the same number of reviews. We randomly select 80% examples for training purpose, 10% for validation purpose and report the F1 score using the rest 10% testing data.

We compare the F1 score of our method with two baselines. Random 50 random guesses a review pair is true or false 50% to 50%, while random bias random guess X% pairs to be true and the rest of pairs to be false, where X% is the percentage of true pairs in the training data. Figure 4.7 shows the F1 score for each homework. r0, r1, and r2 represent three rubric elements that are used during 2019 Spring and Fall. Besides F1 of these different rubric elements, we also plot the f1 of the combined prediction of a pair (combined) by averaging the probabilities of the corresponding rubric elements.

Figure 4.7 shows that peer review content specificity in 2019 is higher than in 2017. Especially in 2019 Fall, F1 score slightly increases. So, because of human evaluation and providing comments on different aspects, peers are likely to provide more specific and detailed peer reviews. Our result also show that the combined F1 score achieves about the same F1 scores as three rubric elements. Thus, averaging the prediction of different rubric elements does not improve F1 score much.

## 4.7. Conclusion

Due to high biases and variances of peer grades, peer grading is a challenging task. This chapter first proposed, SABTXT, a semi-automated peer grading method. SABTXT improves peer grading accuracy through two mechanisms. First, by using a limited amount

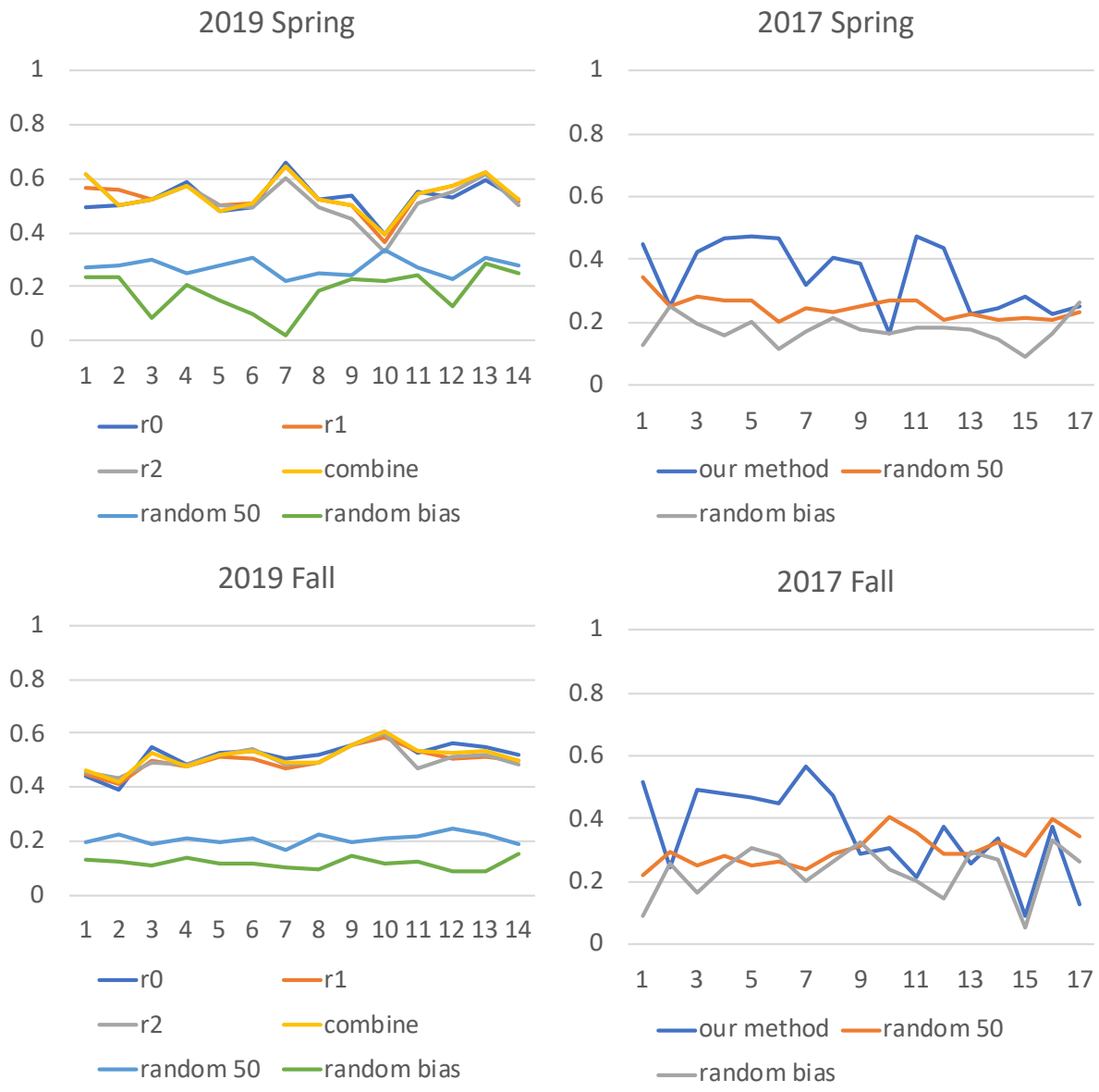


Figure 4.7. Peer review specificity evaluation using unsupervised method. A $N^2$ is built for each class or rubric element. The test F1 score(Y-axis) for each homework(X-axis) is reported. Peer reviews of 2019 achieves higher f1 scores comparing to 2017.

of historical instructor grades, SABTXT refines a model of each peer's bias throughout the course. Second, SABTXT models the quality of peer reviews based on their textual

content, and puts more weight on the more thorough peer reviews when estimating submission grades. The experimental results with over ten thousand peer reviews collected over four courses demonstrate that SABTXT outperforms three baseline models. We find that simple models of the text perform comparably to more powerful techniques. We also investigated the impact of peer review processes on students behavior. Our results showed that students tends to provide longer reviews when the review quality is evaluated by TA. Also, by providing separate reviews based on different rubric elements, textual review contents are more specific.

CHAPTER 5

# Conclusion and Future Work

This dissertation introduced NLP methods that benefit peer review. Specifically, by highlighting important text of submissions, peers could potentially save time on peer review by focusing on the important parts of submissions and skip the un-highlighted parts. We also did a preliminary study which is open domain entity typing. Peer grading algorithms proposed in this dissertation improves the accuracy of consensus grade estimation. We now conclude this dissertation and discuss the remaining challenges and future work.

**Chapter 2** tackled the problem of open domain entity typing, which is to type entities using labels that do not exist in training data. OTyper is proposed to solve this problem. OTyper relies on type embeddings in order to extend to unseen types. OTpyer learns to map the mention representation and type representation to a common vector space. The dot distance of correct mentions and types are minimized. The results showed that OTyper outperforms two baselines models and achieves similar accuracy and F1 score with state-of-the art named entity typing model. Also, similar training types provide more information for unseen type prediction than dissimilar training types do.

As we mentioned in chapter 2, our results showed that OTyper is good at ranking the unseen types of entities, but the performance of predicting whether an entity belongs to a type is not promising. OTyper achieves a F1 score of 0.26 of answering the yes or no question. This result indicates that the yes or no threshold for different types are different. A threshold prediction part needs to be added in order to do this task better.

So, given an entity and a type, how to answer the question of yes or no could be one of future work.

Also, although OTyper performs well on open domain entity typing, it cannot generate types, which is an interesting and challenging task, i.e. given an entity in a context, generates all possible types of the entity. This could be done using language generation models.

**Chapter 3** addressed the task of highlighted text prediction. We proposed models that could predict the important parts of submissions. Thus, peers can save time on reviewing. Our neural model adopts over-labeling technique to use textual reviews and review labels to help with highlighted text prediction. Our results showed that using review labels our model achieves higher F1 scores comparing to baseline models. However, using textual reviews did not show promising results. This technique can potentially reducing peer review time by highlighting important parts of submissions, although we did not do experiments on this part.

One direction to the improve textual review over-labeling model is to incorporate document or paragraph representation. A good document representation can preserve semantic information, which is helpful for highlighted text prediction. There are many neural network based techniques to generate document representation, such as CNN based document representation model [68], unsupervised weighted embedding model [137]. We could also combine neural document representations with other document representations. Latent Dirichlet allocation (LDA) is a topic model, which can generate the topic distribution of a document. Topic distribution could be a good feature which may improve

accuracy. Also, domain knowledge bases could be combined into our model to improve prediction performance.

Also, we note that a potential limitation of highlighted text prediction method proposed in this dissertation is that the proposed method needs training data. This training data is specific to a course or a submission. It may not be generalize to other courses easily. So, how to predict important section of submissions for general purpose is an interesting direction.

**Chapter** 4 mainly focused on the problem of consensus grade estimation. We first investigated and improved a state-of-the-art peer grading algorithm. We also proposed SABTXT, a novel peer grading algorithm, which estimates peer bias by using historical data and weight peer grades based on the quality of textual reviews. Various methods were proposed to estimate textual review quality. In our experiments, a simple length-based model of quality was shown to outperform much more sophisticated and cumbersome language models. A co-training method is proposed to further improve peer grading accuracy. Our analysis also showed that by human evaluating the quality of peer reviews and providing comments on different aspects of submissions, peers tend to provide better quality reviews.

One possible direction to further improve peer grading performance is to consider individual behavior on textual peer reviews. Textual review feature differs from peers to peers. Thus, by considering textual review features for each individual, models may estimate peer grades weights better. Also, SABTXT computes peer biases based on all ground truth scores. If a peer just gives average grades regardless of submission qualities, peer biases may not be bad, but the peer grades are not informative. A future direction to

deal with this problem is to identify peer bias or quality using submissions which do not receive average grades. Thus, if a peer grade is close to non-average submissions, this peer should be considered as good peers and peer bias should be estimated accordingly. Our results show that our methods exhibit very different absolute performance across different classes and homeworks, which suggests that further experiments with a variety of classes beyond the four we evaluate here are necessary. Exploring whether richer models of peers and submission content can achieve higher accuracy could also be a promising direction in the future.

I hope this dissertation could benefit peer review in real scenarios and encourage researchers to tackle more practical problems in education using artificial intelligence.

# References


[1] *A Closer Look at the Common Challenges in Peer Review*. `https://www.econtentpro.com/blog/common-challenges-in-peer-review/49`.

[2] Ahmed Abdul-Hamid and Kareem Darwish. "Simplified feature set for Arabic named entity recognition". In: *Proceedings of the 2010 Named Entities Workshop*. Association for Computational Linguistics. 2010, pp. 110–115.

[3] Luca de Alfaro and Michael Shavlovsky. "Crowdgrader: Crowdsourcing the evaluation of homework assignments". In: *arXiv preprint arXiv:1308.5273* (2013).

[4] Ivo Anastácio, Bruno Martins, Pável Calado, et al. "Supervised Learning for Linking Named Entities to Knowledge Base Entries." In: *TAC*. 2011.

[5] Yigal Attali and Jill Burstein. "AUTOMATED ESSAY SCORING WITH E-RATER® V. 2.0". In: *ETS Research Report Series* 2004.2 (2004).

[6] Gabriel Badea and Elvira Popescu. "A Web-Based Platform for Peer Assessment in Technology Enhanced Learning: Student Module Prototype". In: *2019 IEEE 19th International Conference on Advanced Learning Technologies (ICALT)*. Vol. 2161. IEEE. 2019, pp. 372–374.

[7] Gabriel Badea and Elvira Popescu. "Instructor Support Module in a Web-Based Peer Assessment Platform". In: *2019 23rd International Conference on System Theory, Control and Computing (ICSTCC)*. IEEE. 2019, pp. 691–696.

[8] Alberto Bartoli et al. "Your paper has been accepted, rejected, or whatever: Automatic generation of scientific paper reviews". In: *International Conference on Availability, Reliability, and Security*. Springer. 2016, pp. 19–28.

[9] Taylor Berg-Kirkpatrick, Dan Gillick, and Dan Klein. "Jointly learning to extract and compress". In: *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*. Association for Computational Linguistics. 2011, pp. 481–490.

[10] Avrim Blum and Tom Mitchell. "Combining labeled and unlabeled data with co-training". In: *Proceedings of the eleventh annual conference on Computational learning theory*. 1998, pp. 92–100.

[11] David Carmel et al. "Improving term weighting for community question answering search using syntactic analysis". In: *Proceedings of the 23rd acm international conference on conference on information and knowledge management*. 2014, pp. 351–360.

[12] Xavier Carreras, Lluis Marquez, and Llués Padró. "Named entity extraction using adaboost". In: *proceedings of the 6th conference on Natural language learning-Volume 20*. Association for Computational Linguistics. 2002, pp. 1–4.

[13] Kai-Wei Chang, Rajhans Samdani, and Dan Roth. "A constrained latent variable model for coreference resolution". In: *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*. 2013, pp. 601–612.

[14] Ming-Wei Chang et al. "Importance of Semantic Representation: Dataless Classification." In: *AAAI*. Vol. 2. 2008, pp. 830–835.

[15] Zheng Chen and Heng Ji. "Collaborative ranking: A case study on entity linking". In: *Proceedings of the Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics. 2011, pp. 771–781.

[16] Nancy Chinchor and Patricia Robinson. "MUC-7 named entity task definition". In: *Proceedings of the 7th Conference on Message Understanding*. Vol. 29. 1997.

[17] Nancy Van Note Chism. *Peer Review of Teaching. A Sourcebook*. ERIC, 1999.

[18] Kwangsu Cho, Christian D Schunn, and Davida Charney. "Commenting on writing: Typology and perceived helpfulness of comments from novice peer reviewers and subject matter experts". In: *Written communication* 23.3 (2006), pp. 260–294.

[19] Sumit Chopra, Michael Auli, and Alexander M Rush. "Abstractive sentence summarization with attentive recurrent neural networks". In: *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. 2016, pp. 93–98.

[20] Kevin Clark and Christopher D Manning. "Entity-centric coreference resolution with model stacking". In: *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. 2015, pp. 1405–1415.

[21] Michael Collins and Yoram Singer. "Unsupervised models for named entity classification". In: *Proceedings of the joint SIGDAT conference on empirical methods in natural language processing and very large corpora*. 1999, pp. 100–110.

[22] Dan Cristea et al. "AR-Engine-a framework for unrestricted co-reference resolution." In: *LREC*. 2002.

[23] Hong-Jie Dai et al. "Integration of gene normalization stages and co-reference resolution using a Markov logic network". In: *Bioinformatics* 27.18 (2011), pp. 2586–2594.

[24] Jacob Devlin et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *arXiv preprint arXiv:1810.04805* (2018).

[25] Li Dong et al. "Learning to generate product reviews from attributes". In: *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*. 2017, pp. 623–632.

[26] Sourav Dutta and Gerhard Weikum. "Cross-document co-reference resolution using sample-based clustering with knowledge enrichment". In: *Transactions of the Association for Computational Linguistics* 3 (2015), pp. 15–28.

[27] Harold P Edmundson. "New methods in automatic extracting". In: *Journal of the ACM (JACM)* 16.2 (1969), pp. 264–285.

[28] Micha Elsner, Eugene Charniak, and Mark Johnson. "Structured generative models for unsupervised named-entity clustering". In: *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*. Association for Computational Linguistics. 2009, pp. 164–172.

[29] Günes Erkan and Dragomir R Radev. "Lexrank: Graph-based lexical centrality as salience in text summarization". In: *Journal of artificial intelligence research* 22 (2004), pp. 457–479.

[30] Minwei Feng et al. "Applying deep learning to answer selection: A study and an open task". In: *2015 IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*. IEEE. 2015, pp. 813–820.

[31] Hugh Glaser, Afraz Jaffri, and Ian Millard. "Managing co-reference on the semantic web". In: (2009).

[32] Swapna Gottipati and Jing Jiang. "Linking entities to a knowledge base with query expansion". In: *Proceedings of the Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics. 2011, pp. 804–813.

[33] Alex Graves. "Supervised sequence labelling". In: *Supervised sequence labelling with recurrent neural networks* (2012), pp. 5–13.

[34] Ralph Grishman and Beth Sundheim. "Message understanding conference-6: A brief history". In: *COLING 1996 Volume 1: The 16th International Conference on Computational Linguistics*. Vol. 1. 1996.

[35] Prashant Gupta and Heng Ji. "Predicting unknown time arguments based on cross-event propagation". In: *Proceedings of the ACL-IJCNLP 2009 Conference Short Papers*. 2009, pp. 369–372.

[36] Vishal Gupta and Gurpreet Singh Lehal. "A survey of text summarization extractive techniques". In: *Journal of emerging technologies in web intelligence* 2.3 (2010), pp. 258–268.

[37] Suchin Gururangan et al. "Don't Stop Pretraining: Adapt Language Models to Domains and Tasks". In: *arXiv preprint arXiv:2004.10964* (2020).

[38] Xianpei Han and Le Sun. "A generative entity-mention model for linking entities with knowledge base". In: *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*. Association for Computational Linguistics. 2011, pp. 945–954.

[39] Marti A Hearst. "Automatic acquisition of hyponyms from large text corpora". In: *Proceedings of the 14th conference on Computational linguistics-Volume 2*. Association for Computational Linguistics. 1992, pp. 539–545.

[40] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory". In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[41] *How to Make Peer Review Successful*. 2018. URL: `https://www.thegraidenetwork.com/blog-all/2018/12/20/how-to-make-peer-reviews-successful-part-1-of-2` (visited on 12/20/2018).

[42] Jeremy Howard and Sebastian Ruder. "Universal language model fine-tuning for text classification". In: *arXiv preprint arXiv:1801.06146* (2018).

[43] Wei Hu, Jianfeng Chen, and Yuzhong Qu. "A self-training approach for resolving object coreference on the semantic web". In: *Proceedings of the 20th international conference on World wide web*. 2011, pp. 87–96.

[44] Lifu Huang et al. "Building a fine-grained entity typing system overnight for a new x (x= language, domain, genre)". In: *arXiv preprint arXiv:1603.03112* (2016).

[45] Lifu Huang et al. "Liberal Entity Extraction: Rapid Construction of Fine-Grained Entity Typing Systems". In: *Big Data* 5.1 (2017), pp. 19–31.

[46] *Intelligent Essay Assessor*. `http://lsa.colorado.edu/IEA2/IEA2.html`.

[47] *itextpdf*. `https://itextpdf.com/en`.

[48] Abhyuday N Jagannatha and Hong Yu. "Structured prediction models for RNN based sequence labeling in clinical text". In: *Proceedings of the conference on empirical methods in natural language processing. conference on empirical methods in natural language processing*. Vol. 2016. NIH Public Access. 2016, p. 856.

[49] Ludovic Jean-Louis et al. "Using a weakly supervised approach and lexical patterns for the KBP slot filling task." In: *TAC*. 2011.

[50] Guoliang Ji et al. "Distant supervision for relation extraction with sentence-level attention and entity descriptions". In: *Thirty-First AAAI Conference on Artificial Intelligence*. 2017.

[51] Heng Ji and Ralph Grishman. "Knowledge base population: Successful approaches and challenges". In: *Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies-volume 1*. Association for Computational Linguistics. 2011, pp. 1148–1158.

[52] Xiaotian Jiang et al. "Relation extraction with multi-instance multi-label convolutional neural networks". In: *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*. 2016, pp. 1471–1480.

[53] Antonio J Jimeno-Yepes, Bridget T McInnes, and Alan R Aronson. "Exploiting MeSH indexing in MEDLINE to generate a data set for word sense disambiguation". In: *BMC bioinformatics* 12.1 (2011), p. 223.

[54] Hongyan Jing. "Sentence reduction for automatic text summarization". In: *Sixth Applied Natural Language Processing Conference*. 2000, pp. 310–315.

[55] Hongyan Jing and Kathleen R McKeown. "Cut and paste based text summarization". In: *Proceedings of the 1st North American chapter of the Association for Computational Linguistics conference*. Association for Computational Linguistics. 2000, pp. 178–185.

[56] Sydney Johnson. *Enrollment trends*. `https://www.edsurge.com/news/2018-04-03-computer-science-degrees-and-technology-s-boom-and-bust-cycle`. 2018.

[57] Dongyeop Kang et al. "A Dataset of Peer Reviews (PeerRead): Collection, Insights and NLP Applications". In: *arXiv preprint arXiv:1804.09635* (2018).

[58] Diederik Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[59] Kevin Knight and Daniel Marcu. "Statistics-based summarization-step one: Sentence compression". In: *AAAI/IAAI* 2000 (2000), pp. 703–710.

[60] Chinmay E Kulkarni, Michael S Bernstein, and Scott R Klemmer. "PeerStudio: rapid peer feedback emphasizes revision and improves performance". In: *Proceedings of the second (2015) ACM conference on learning@ scale*. 2015, pp. 75–84.

[61] Onur Kuru, Ozan Arkan Can, and Deniz Yuret. "CharNER: Character-Level Named Entity Recognition." In: *COLING*. 2016, pp. 911–921.

[62] John Lafferty, Andrew McCallum, and Fernando CN Pereira. "Conditional random fields: Probabilistic models for segmenting and labeling sequence data". In: (2001).

[63] Philip Laird and Ronald Saul. "Discrete sequence prediction and its applications". In: *Machine learning* 15.1 (1994), pp. 43–68.

[64] Changki Lee et al. "Fine-grained named entity recognition using conditional random fields for question answering". In: *Asia Information Retrieval Symposium*. Springer. 2006, pp. 581–587.

[65] Pan Li and Alexander Tuzhilin. "Towards Controllable and Personalized Review Generation". In: *arXiv preprint arXiv:1910.03506* (2019).

[66] Yankai Lin et al. "Neural relation extraction with selective attention over instances". In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 2016, pp. 2124–2133.

[67] Xiao Ling and Daniel S Weld. "Fine-Grained Entity Recognition." In: *AAAI*. 2012.

[68] Chundi Liu, Shunan Zhao, and Maksims Volkovs. "Learning Document Embeddings With CNNs". In: *CoRR* abs/1711.04168 (2017). arXiv: 1711.04168. URL: http://arxiv.org/abs/1711.04168.

[69] ChunYang Liu et al. "Convolution neural network for relation extraction". In: *International Conference on Advanced Data Mining and Applications*. Springer. 2013, pp. 231–242.

[70] Tianyi Liu et al. "Neural relation extraction via inner-sentence noise reduction and transfer learning". In: *arXiv preprint arXiv:1808.06738* (2018).

[71] Hans Peter Luhn. "The automatic creation of literature abstracts". In: *IBM Journal of research and development* 2.2 (1958), pp. 159–165.

[72] Erin Lynch. *Annotating Text Strategies That Will Enhance Close Reading*. https://www.sadlier.com/school/ela-blog/teaching-annotation-to-students-grades-2-8-annotating-text-strategies-that-will-enhance-close-reading.

[73] Tomas Mikolov, Wen-tau Yih, and Geoffrey Zweig. "Linguistic regularities in continuous space word representations." In: *hlt-Naacl*. Vol. 13. 2013, pp. 746–751.

[74] David Milne and Ian H Witten. "Learning to link with wikipedia". In: *Proceedings of the 17th ACM conference on Information and knowledge management*. 2008, pp. 509–518.

[75] Yuji Mo, Stephen D Scott, and Doug Downey. "Learning hierarchically decomposable concepts with active over-labeling". In: *2016 IEEE 16th International Conference on Data Mining (ICDM)*. IEEE. 2016, pp. 340–349.

[76] Sean Monahan et al. "Cross-Lingual Cross-Document Coreference with Entity Linking." In: *TAC*. 2011.

[77] *Named-entity recognition*. https://en.wikipedia.org/wiki/Named-entity_recognition.

[78] Thien Huu Nguyen and Ralph Grishman. "Relation extraction: Perspective from convolutional neural networks". In: *Proceedings of the 1st Workshop on Vector Space Modeling for Natural Language Processing*. 2015, pp. 39–48.

[79] Jianmo Ni and Julian McAuley. "Personalized review generation by expanding phrases and attending on aspect-aware representations". In: *Proceedings of the*

*56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. 2018, pp. 706–711.

[80] Jianmo Ni et al. "Estimating reactions and recommending products with generative models of reviews". In: *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. 2017, pp. 783–791.

[81] Ellis Batten Page. "Project essay grade: PEG". In: *Automated essay scoring: A cross-disciplinary perspective* (2003).

[82] Mark Palatucci et al. "Zero-shot learning with semantic output codes". In: *Advances in neural information processing systems*. 2009, pp. 1410–1418.

[83] Haoruo Peng, Daniel Khashabi, and Dan Roth. "Solving hard coreference problems". In: *arXiv preprint arXiv:1907.05524* (2019).

[84] Jeffrey Pennington, Richard Socher, and Christopher D Manning. "Glove: Global vectors for word representation." In: *EMNLP*. Vol. 14. 2014, pp. 1532–1543.

[85] Chris Piech et al. "Tuned models of peer assessment in MOOCs". In: *arXiv preprint arXiv:1307.2579* (2013).

[86] Danuta Ploch. "Exploring entity relations for named entity disambiguation". In: *Proceedings of the ACL 2011 Student Session*. Association for Computational Linguistics. 2011, pp. 18–23.

[87] Carol Porter-O'Donnell. "Beyond the yellow highlighter: Teaching annotation skills to improve reading comprehension". In: *English Journal* (2004), pp. 82–89.

[88] Pengda Qin, Weiran Xu, and William Yang Wang. "Dsgan: Generative adversarial training for distant supervision relation extraction". In: *arXiv preprint arXiv:1805.09929* (2018).

[89] Xipeng Qiu and Xuanjing Huang. "Convolutional neural tensor network architecture for community-based question answering". In: *Twenty-Fourth international joint conference on artificial intelligence*. 2015.

[90] Maxim Rabinovich and Dan Klein. "Fine-Grained Entity Typing with High-Multiplicity Assignments". In: *arXiv preprint arXiv:1704.07751* (2017).

[91] Alec Radford, Rafal Jozefowicz, and Ilya Sutskever. "Learning to generate reviews and discovering sentiment". In: *arXiv preprint arXiv:1704.01444* (2017).

[92] Pranav Rajpurkar et al. "Squad: 100,000+ questions for machine comprehension of text". In: *arXiv preprint arXiv:1606.05250* (2016).

[93] Karthik Raman and Thorsten Joachims. "Bayesian ordinal peer grading". In: *Proceedings of the Second (2015) ACM Conference on Learning@ Scale*. 2015, pp. 149–156.

[94] Karthik Raman and Thorsten Joachims. "Methods for ordinal peer grading". In: *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. 2014, pp. 1037–1046.

[95] Lev Ratinov and Dan Roth. "Design challenges and misconceptions in named entity recognition". In: *Proceedings of the Thirteenth Conference on Computational Natural Language Learning*. for Computational Linguistics. 2009, pp. 147–155.

[96] Lev Ratinov and Dan Roth. "Learning-based multi-sieve co-reference resolution with knowledge". In: *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*. Association for Computational Linguistics. 2012, pp. 1234–1244.

[97] Lloyd J Rieber. "Using peer review to improve student writing in business courses". In: *Journal of Education for Business* 81.6 (2006), pp. 322–326.

[98] Alan Ritter, Sam Clark, Oren Etzioni, et al. "Named entity recognition in tweets: an experimental study". In: *Proceedings of the Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics. 2011, pp. 1524–1534.

[99] Alan Ritter, Stephen Soderland, and Oren Etzioni. "What Is This, Anyway: Automatic Hypernym Discovery." In: *AAAI Spring Symposium: Learning by Reading and Learning to Read*. 2009, pp. 88–93.

[100] Erik Tjong Kim Sang. "Extracting hypernym pairs from the web". In: *Proceedings of the 45th annual meeting of the ACL on interactive poster and demonstration sessions*. Association for Computational Linguistics. 2007, pp. 165–168.

[101] Abigail See, Peter J Liu, and Christopher D Manning. "Get to the point: Summarization with pointer-generator networks". In: *arXiv preprint arXiv:1704.04368* (2017).

[102] Julian Seitner et al. "A Large DataBase of Hypernymy Relations Extracted from the Web." In: *LREC*. 2016.

[103] *Sequence labeling*. https://en.wikipedia.org/wiki/Sequence_labeling.

[104] M.D. Shermis and J.C. Burstein. *Automated Essay Scoring: A Cross-Disciplinary Perspective*. Routledge, 2016. ISBN: 9781138964211. URL: https://books.google.com/books?id=ccdbjgEACAAJ.

[105] Sonse Shimaoka et al. "An attentive neural architecture for fine-grained entity type classification". In: *arXiv preprint arXiv:1604.05525* (2016).

[106] Sonse Shimaoka et al. "Neural Architectures for Fine-grained Entity Type Classification". In: *arXiv preprint arXiv:1606.01341* (2016).

[107] Rion Snow, Daniel Jurafsky, and Andrew Y Ng. "Learning syntactic patterns for automatic hypernym discovery". In: *Advances in neural information processing systems*. 2005, pp. 1297–1304.

[108] Richard Socher et al. "Zero-shot learning through cross-modal transfer". In: *Advances in neural information processing systems*. 2013, pp. 935–943.

[109] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. "Sequence to Sequence Learning with Neural Networks". In: *CoRR* abs/1409.3215 (2014). arXiv: 1409.3215. URL: http://arxiv.org/abs/1409.3215.

[110] Ming Tan et al. "Lstm-based deep learning models for non-factoid answer selection". In: *arXiv preprint arXiv:1511.04108* (2015).

[111] Duyu Tang et al. "Question answering and question generation as dual tasks". In: *arXiv preprint arXiv:1706.02027* (2017).

[112] Yi Tay, Luu Anh Tuan, and Siu Cheung Hui. "Hyperbolic representation learning for fast and efficient neural question answering". In: *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*. 2018, pp. 583–591.

[113] *Text annotation*. https://en.wikipedia.org/wiki/Text_annotation.

[114] *The Conjugate Prior for the Normal Distribution*. https://people.eecs.berkeley.edu/~jordan/courses/260-spring10/lectures/lecture5.pdf. Accessed: 2019-11-14.

[115] Erik F Tjong Kim Sang and Fien De Meulder. "Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition". In: *Proceedings of*

*the seventh conference on Natural language learning at HLT-NAACL 2003-Volume 4*. Association for Computational Linguistics. 2003, pp. 142–147.

[116] Quoc-Tuan Truong and Hady Lauw. "Multimodal Review Generation for Recommender Systems". In: *The World Wide Web Conference*. 2019, pp. 1864–1874.

[117] *Use a text annotation tool to find information in text quicker*. `https://medium.com/@tagtog/use-a-text-annotation-tool-to-find-information-in-text-quickly-c404fc226ae5`.

[118] Alex Wang et al. "Glue: A multi-task benchmark and analysis platform for natural language understanding". In: *arXiv preprint arXiv:1804.07461* (2018).

[119] 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*. 2009, pp. 187–194.

[120] Sam Wiseman, Alexander M Rush, and Stuart M Shieber. "Learning global features for coreference resolution". In: *arXiv preprint arXiv:1604.03035* (2016).

[121] James R Wright, Chris Thornton, and Kevin Leyton-Brown. "Mechanical TA: Partially automated high-stakes peer grading". In: *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*. 2015, pp. 96–101.

[122] Xiaobing Xue, Jiwoon Jeon, and W Bruce Croft. "Retrieval models for question and answer archives". In: *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*. 2008, pp. 475–482.

[123] Limin Yao, Sebastian Riedel, and Andrew McCallum. "Universal schema for entity type prediction". In: *Proceedings of the 2013 workshop on Automated knowledge base construction*. ACM. 2013, pp. 79–84.

[124] Dani Yogatama, Daniel Gillick, and Nevena Lazic. "Embedding Methods for Fine Grained Entity Type Classification." In: *ACL (2)*. 2015, pp. 291–296.

[125] Lei Yu et al. "Deep learning for answer sentence selection". In: *arXiv preprint arXiv:1412.1632* (2014).

[126] Zheng Yuan and Doug Downey. "Otyper: A neural architecture for open named entity typing". In: *Thirty-Second AAAI Conference on Artificial Intelligence*. 2018.

[127] Hongyu Zang and Xiaojun Wan. "Towards automatic generation of product reviews from aspect-sentiment scores". In: *Proceedings of the 10th International Conference on Natural Language Generation*. 2017, pp. 168–177.

[128] Rowan Zellers et al. "Swag: A large-scale adversarial dataset for grounded commonsense inference". In: *arXiv preprint arXiv:1808.05326* (2018).

[129] Daojian Zeng et al. "Relation classification via convolutional deep neural network". In: (2014).

[130] Wenhuan Zeng et al. "Automatic generation of personalized comment based on user profile". In: *arXiv preprint arXiv:1907.10371* (2019).

[131] Wenyuan Zeng et al. "Incorporating relation paths in neural relation extraction". In: *arXiv preprint arXiv:1609.07479* (2016).

[132] Xiaodong Zhang et al. "Attentive interactive neural networks for answer selection in community question answering". In: *Thirty-First AAAI Conference on Artificial Intelligence*. 2017.

[133] Xinsong Zhang et al. "Multi-labeled relation extraction with attentive capsule network". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 33. 2019, pp. 7484–7491.

[134] Lujun Zhao et al. "Review Response Generation in E-Commerce Platforms with External Product Information". In: *The World Wide Web Conference*. 2019, pp. 2425–2435.

[135] Yanpeng Zhao et al. "Sequence Prediction Using Neural Network Classiers". In: *International conference on grammatical inference*. 2017, pp. 164–169.

[136] Yukun Zhu et al. "Aligning books and movies: Towards story-like visual explanations by watching movies and reading books". In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 19–27.

[137] Zhaocheng Zhu and Junfeng Hu. "Context Aware Document Embedding". In: *arXiv preprint arXiv:1707.01521* (2017).