

# **DEEP LEARNING FOR UNSTRUCTURED DATA BY LEVERAGING DOMAIN KNOWLEDGE**

---

A Dissertation  
Submitted to  
the Temple University Graduate Board

---

in Partial Fulfillment  
of the Requirements for the Degree of  
DOCTOR OF PHILOSOPHY

---

by  
Shanshan Zhang  
August 2019

---

Examining Committee Members:

Dr Slobodan Vucetic, Advisory Chair, Dept. of Computer and Information Sciences  
Dr Eduard Dragut, Dept. of Computer and Information Sciences  
Dr Krishna Kant, Dept. of Computer and Information Sciences  
Dr Qimin Yan, External Member, Dept. of Physics

©  
Shanshan Zhang  
2019

by

---

All Rights Reserved

# ABSTRACT

Deep Learning for Unstructured Data by Leveraging Domain Knowledge

by

Shanshan Zhang

Unstructured data such as texts, strings, images, audios, videos are everywhere due to the social interaction on the Internet and the high-throughput technology in sciences, e.g., chemistry and biology. However, for traditional machine learning algorithms, classifying a text document is far more difficult than classifying a data entry in a spreadsheet. We have to convert the unstructured data into some numeric vectors which can then be understood by machine learning algorithms. For example, a sentence is first converted to a vector of word counts, and then fed into a classification algorithm such as logistic regression and support vector machine. The creation of such numerical vectors is very challenging and difficult. Recent progress in deep learning provides us a new way to jointly learn features and train classifiers for unstructured data. For example, recurrent neural networks proved successful at learning from a sequence of word indices; convolutional neural networks are effective to learn from videos, which are sequences of pixel matrices. Our research focuses on developing novel deep learning approaches for text and graph data.

Breakthroughs using deep learning have been made during the last few years for many core tasks in natural language processing, such as machine translation, POS tagging, named entity recognition, etc. However, when it comes to informal and noisy text data, such as tweets, HTMLs, OCR, there are two major issues with modern deep learning technologies. First, deep learning requires large amount of labeled data to train an effective model; second, neural network architectures that work with natural language are not proper with informal text. In this thesis, we address the two important issues and develop new deep learning approaches in four supervised and unsupervised tasks with noisy text. We first present a deep feature engineering approach for informative tweets discovery during the

emerging disasters. We propose to use unlabeled microblogs to cluster words into a limited number of clusters and use the word clusters as features for tweets discovery. Our results indicate that when the number of labeled tweets is 100 or less, the proposed approach is superior to the standard classification based on the bag of words feature representation. We then introduce a human-in-the-loop (HIL) framework for entity identification from noisy web text. Our work explores ways to combine the expressive power of REs, ability of deep learning to learn from large data into a new integrated framework for entity identification from web data. The evaluation on several entity identification problems shows that the proposed framework achieves very high accuracy while requiring only a modest human involvement. We further extend the framework of entity identification to an iterative HIL framework that addresses the entity recognition problem. We particularly investigate how human invest their time when a user is allowed to choose between regex construction and manual labeling. Finally, we address a fundamental problem in the text mining domain, i.e, embedding of rare and out-of-vocabulary (OOV) words, by refining word embedding models and character embedding models in an iterative way. We illustrate the simplicity but effectiveness of our method when applying it to online professional profiles allowing noisy user input.

Graph neural networks have been shown great success in the domain of drug design and material sciences, where organic molecules and crystal structures of materials are represented as attributed graphs. A deep learning architecture that is capable of learning from graph nodes and graph edges is crucial for property estimation of molecules. In this dissertation, We propose a simple graph representation for molecules and three neural network architectures that is able to directly learn predictive functions from graphs. We discover that, it is true graph networks are superior than feature-driven algorithms for formation energy prediction. However, the superiority can not be reproduced on band gap prediction. We also discovered that our proposed simple shallow neural networks perform comparably with the state-of-the-art deep neural networks.

*To my mother and mother-in-law Hongmei Peng, Limin Wang*

*To my father and father-in-law Zhaohua Qin, Dongmin Han*

*To my husband Chao Han*

*To my aunts and uncles in my big warm family*

*Your endless and doubtless love sends me here. Thanks! I love you back.*

In loving memory of my dad and grandma

Jun Zhang, Shanying Xiao

## ACKNOWLEDGEMENTS

This thesis could not have been written without the support from Dr. Slobodan Vucetic. During my graduate study, he is always supportive, helpful and patient with me. He taught me the tactics to do research but more importantly, the art of doing good research, which I can not learn from anywhere and anybody else. I want to thank him for guiding me to be practical and insightful with a good sense of data.

Special thanks to Dr. Eduard Dragut, who brings research questions and opportunities to me. I enjoyed the productive discussions and debates with him very much. I would also thank Dr. Krishna Kant, from whom I learn how to collaborate with researchers from other areas. I sincerely thank Dr. Qimin Yan, who introduces me to new adventures in physics.

I thank the staffs and faculties from the Department of Computer and Information Science at Temple University, for creating a world-class environment for me to work in. I want to thank Dr. Justin Shi, who is always there as a friend. Special thanks to Julie Krystopa Skrocki, Andrea Mcgady, Christopher Bryant who help me with tons of paper work.

I thank my collaborators, my labmates, and my friends, Dr. Ana Gamero, Huta Banjade, Amitangshu Pal, Tian Bai, Aniruddha Maiti, Ashis Chanda, Ziyu Yang, Saman Enayati, Sandro Hauri, Shikai Fang, Hoang Ho, Phong The Ha, Jie Yang, Xue Wang, Lu Li, Rong Su, Min Xiao, Ning Xia, Tao Ma, Jiang Long, and so on. The encouragement and good memories from them keep me going when there are difficulties.

I thank Chao Han, with whom I have shared the whole journey with and will share the rest of all journeys.

Finally, this work was supported by NSF grant CNS-1461932, NSF BigData 1546480 and NIH R21CA202130..

# TABLE OF CONTENTS

<b>ABSTRACT</b> . . . . .	ii
<b>DEDICATION</b> . . . . .	iv
<b>ACKNOWLEDGEMENTS</b> . . . . .	v
<b>LIST OF FIGURES</b> . . . . .	ix
<b>LIST OF TABLES</b> . . . . .	x
<b>CHAPTER</b>	
<b>1. INTRODUCTION</b> . . . . .	1
1.1 Tweet Message Classification . . . . .	1
1.2 Entity Identification and Recognition . . . . .	2
1.3 Word Embedding for OOV words . . . . .	3
1.4 Property Estimation for Crystal Materials . . . . .	4
<b>2. DEEP FEATURE ENGINEERING FOR MESSAGE FILTERING AND SUMMARIZATION</b> . . . . .	6
2.1 Introduction . . . . .	6
2.2 Message Filtering Methodology . . . . .	8
2.2.1 Problem Setup . . . . .	8
2.2.2 Feature Selection Filters . . . . .	9
2.2.3 Word Clusters as Features . . . . .	9
2.3 Evaluate Message Filtering . . . . .	10
2.3.1 Data Set . . . . .	11
2.3.2 Experiment Design . . . . .	12
2.3.3 Results . . . . .	14
2.4 Message Summarization Methodology . . . . .	17
2.4.1 Informativeness score . . . . .	18
2.4.2 Semantic completeness . . . . .	18
2.5 Evaluate Message Summarization . . . . .	20
2.5.1 Data Sets . . . . .	20
2.5.2 Simulated Settings . . . . .	20
2.5.3 Results . . . . .	22
2.6 Conclusions . . . . .	25

<b>3. REGULAR EXPRESSION GUIDED ENTITY MENTION MINING FROM NOISY WEB DATA</b>	28
3.1 Introduction	28
3.2 Related Work	31
3.3 Methodology	32
3.4 Experimental Setup	34
3.5 Experimental Results	36
3.5.1 Entity Mention Detection with Limited Human Annotations	36
3.5.2 Effect of the Number of Human Annotations	37
3.5.3 Case Studies	38
3.5.4 Impact of the Initial REs	39
3.6 Conclusions	40
<b>4. HOW TO INVEST MY TIME: LESSONS FROM HUMAN-IN-THE-LOOP ENTITY EXTRACTION</b>	41
4.1 Introduction	41
4.2 Related Work	43
4.3 Proposed Framework	45
4.3.1 Selection of Candidate Substrings	46
4.3.2 Weak Labeling	47
4.3.3 Entity Recognizer	48
4.3.4 Fine-Tuning with Human Labels	49
4.3.5 Summary of the Framework	50
4.4 Experimental Design	51
4.4.1 Entity Recognition Tasks	51
4.4.2 Labeled Data for Evaluation	52
4.4.3 Accuracy Measures	52
4.5 Framework Characterization	53
4.5.1 $RE_0$ for Candidate Substring Extraction	54
4.5.2 $RE_{wl}$ for Weak Labeling	54
4.5.3 Hyperparameter Tuning and Settings	55
4.5.4 Impact of Weak Supervision	56
4.5.5 Impact of Active Sampling	57
4.6 User Studies	58
4.6.1 Experimental Design	58
4.6.2 Experimental Results	60
4.7 Conclusions	63
<b>5. ITERATIVE REFINEMENT OF WORD AND CHARACTER EMBEDDING MODELS</b>	64

5.1	Introduction . . . . .	64
5.2	Related Work . . . . .	66
5.3	Methodology . . . . .	67
5.3.1	Mimick and GWR . . . . .	67
5.3.2	Iterative Mimicking . . . . .	68
5.4	Word Similarity . . . . .	70
5.4.1	Experimental Setup . . . . .	70
5.4.2	Results . . . . .	71
5.5	Sentiment Analysis . . . . .	72
5.5.1	Experimental Setup . . . . .	72
5.5.2	Results . . . . .	73
5.6	Job Title Normalization . . . . .	73
5.6.1	Data Set . . . . .	74
5.6.2	Customized IM Embedding Models . . . . .	75
5.6.3	Baselines . . . . .	77
5.6.4	Hyperparameters . . . . .	77
5.6.5	Evaluation Metrics . . . . .	78
5.6.6	Results . . . . .	79
5.7	Case Studies . . . . .	80
5.7.1	Nearest Neighbors Analysis . . . . .	80
5.7.2	Job Title Normalization . . . . .	80
5.8	Conclusions . . . . .	81
<b>6. GRAPH NEURAL NETWORKS FOR PROPERTY ESTIMATION . .</b>		<b>83</b>
6.1	Introduction . . . . .	83
6.2	Related Work . . . . .	84
6.2.1	Feature-driven algorithms . . . . .	84
6.2.2	Neural networks . . . . .	86
6.3	Methodology . . . . .	87
6.4	Experiments . . . . .	88
6.4.1	Dataset . . . . .	88
6.4.2	Experimental setup . . . . .	89
6.5	Results . . . . .	90
6.5.1	Qualitative results . . . . .	90
6.5.2	Error analysis . . . . .	91
6.6	Conclusions . . . . .	92
<b>BIBLIOGRAPHY . . . . .</b>		<b>93</b>

# LIST OF FIGURES

**Figure**

2.1	Detailed performance comparison for 6 disasters. . . . .	17
2.2	The Manhattan area is partitioned into cells with size $D_{fix} = 500m$ . The tweet densities are shown in different color scale. . . . .	21
2.3	The population density plot for different $D_{fix}$ . X-axis: $\log_2(\text{rank})$ of cell by density; Left Y-axis: number of unique users; Right Y-axis: fraction of unique users. . . . .	21
3.1	Overview of the solution (left) and deep learning architecture (right) used to train model $M_{RE}$ . . . . .	30
3.2	Trend of F1 when varying the number of human annotations. . . . .	36
3.3	Performance of $M_{RE+human}$ for different initial sets of REs and sizes of $D_{human}$ . . . . .	38
4.1	Overview of the solution framework(left) and deep learning architecture (right) used to train entity recognizer. . . . .	46
4.2	An example of course number recognition on a sample HTML document. Only 5 of the 9 $S_0$ candidate substrings are shown . . . . .	46
4.3	4 query algorithms for <b>DATE</b> TIME recognition. . . . .	58
4.4	Time efficiency of volunteer M on <b>DATE</b> TIME task. . . . .	60
5.1	Mimick model. $G$ is a character BiLSTM. . . . .	68
5.2	Illustration of iterative mimicking . . . . .	69
5.3	Distribution of job title frequencies. X-axis is the rounded logarithm of job title frequencies; Y-axis is the number of job titles with a corresponding frequency in log scale. . . . .	75
5.4	Architecture of CNN-LSTM attention model. Job title <i>jr java developer</i> is broken into a set of tokens $\{jr, java, developer, jr\_java\_developer\}$ . . . . .	76
5.5	Example of job title normalization results from 5 competing models . . . . .	77
5.6	Example of job title normalization . . . . .	82
6.1	Architectures of the proposed neural networks. . . . .	85
6.2	Distributions of formation energy and band gap. . . . .	87
6.3	Subfigures from left to right are: ground true property distributions; predicted property distributions; scatter plot of predictions vs. true values; MAE by true measurement range. . . . .	92

# LIST OF TABLES

## Table

2.1	Basic data statistics of each disaster. . . . .	11
2.2	Summary results to compare Semi-supervised ( <b>SS</b> ) and <b>BOW</b> algorithms. . . . .	15
2.3	Comparing two clustering algorithms <b>BC</b> and <b>W2V</b> . . . . .	15
2.4	Comparing 4 types of unlabeled corpus. . . . .	16
2.5	The number of clusters ( <b>K</b> ) to achieve best performance for <b>W2V</b> <sub>400M</sub> . . . . .	17
2.6	Notation used in the summarization stage. . . . .	18
2.7	Number (percentage %) of messages after filtering. . . . .	22
2.8	Number (percentage %) of messages kept after summarization. . . . .	22
2.9	Crisis Related categories <i>Imran et al. (2015,0)</i> . . . . .	23
2.10	Distribution of message categories after filtering ( $\mathcal{S}_1$ ) & after summarization ( $\mathcal{S}_2$ ). . . . .	25
2.11	The 5 selected messages from each city. . . . .	26
3.1	Dataset Summary. . . . .	32
3.2	The comparisons in the presence of very few human annotations: $ D_{human}  = 20$ . . . . .	35
3.3	Examples of misclassified strings by $M_{RE}$ and $RE$ for Date Time task. The first columns shows number of misclassified strings, number of positives, and number of negatives. The last column represent human label and classifications by $RE$ , and $M_{RE}$ . Datetime is in bold for positive human annotation. . . . .	37
3.4	Comparison between $RE$ and $M_{RE}$ model for 5 different sets of REs. . . . .	38
4.1	Summary of documents in the 5 entity recognition tasks. . . . .	50
4.2	Summary of the $RE_0$ . . . . .	54
4.3	Impact of weak supervision and active learning on the 5 tasks. . . . .	56
4.4	The performance of the 6 strategies under different time budgets of volunteers C, J, and A for <b>DATETIME</b> and <b>COUSENUMBER</b> recognition. × means not applicable. . . . .	60
4.5	Regexes created by C, J and A after the first 5 minutes for the <b>DATETIME</b> task. . . . .	62
5.1	Pearson correlation for different embedding models on the 5 data sets . . . . .	70
5.2	Test accuracy on sentiment analysis data sets . . . . .	73
5.3	Job title and its description . . . . .	75
5.4	Testing Scores of each model by different human judges . . . . .	78
5.5	Average testing Scores of each model in different frequency ranges . . . . .	79
5.6	Nearest Neighbors of different embedding models . . . . .	81
6.1	Regression results of 10 models on formation energy prediction. . . . .	91
6.2	Regression results of 10 models on band gap prediction. . . . .	91

# CHAPTER 1

## INTRODUCTION

The last decade witnesses the biggest-ever evolution of artificial intelligence led by the emergence of deep learning. Ever since AlexNet exceeds human performance on image classification in 2012, deep learning paves its way in other domains too. In the natural language processing domain, deep learning models achieve state-of-the-art performance in almost all major tasks, including machine translation, named entity recognition, question answering, named entity recognition. However, challenges exist when applying deep learning on noisy text data. First, deep learning requires huge amount of human labels; second, models that work for natural language do not directly apply to noisy web texts. In this thesis, we address the two issues in both unsupervised and supervised tasks by leveraging huge amount of unlabeled data or domain knowledge such as regular expressions.

### 1.1 Tweet Message Classification

During the past decade, there has been plentiful evidence that social media can be very useful in improving the situational awareness during emergencies and disasters *Imran and Castillo (2015)*, and several research groups even proposed specific systems to extract, process, and summarize social media data for such purpose *Abel et al. (2012)*; *Ashktorab et al. (2014)*; *Imran et al. (2014)*. The prototypical situational awareness system relying on social media, such as the popular microblogging service Twitter, consists of a data collection, a data processing, and a visual interface component. The first critical challenge in making those systems work is selection of the disaster-related posts from a massive stream of posts pouring from services such as Twitter. A supervised learning approach is

based on training a classifier that recognizes informative posts *Abel et al. (2012)*; *Ashktorab et al. (2014)*; *Imran et al. (2014)*. To train the classifier, there is a need to have access to labeled posts. If a sufficient number of labeled posts are available, the supervised approach is superior to the keyword-based approach. However, during the first few minutes or hours from the emergence of a disaster, the number of labeled posts might be too small to result in accurate classification.

In this thesis, we present an effective deep feature engineering approach which leverages a huge amount of unlabeled data *Zhang and Vucetic (2016)*. We then continue our work with a message summarization step to enable faster situational awareness during disasters *Zhang et al. (2018b)*.

## 1.2 Entity Identification and Recognition

Entity Recognition (ER) is the task of automatically locating, extracting, and classifying contiguous pieces of strings, which represent entities of interest, in text. Named entitie extraction assign pre-defined categories (e.g., person, organization, location, expressions of time, monetary values, and emails) to each extracted piece of text. There is a large class of entities that are not “named,” such as expressions of time, emails, and course identifiers. Their main characteristic is that they often follow an underlying syntactical pattern, which can be fully described or well approximated by Regular Expressions (REs). Web documents are an important domain for data extraction. Importantly, the Web is not a place where data follows “underlying syntactical pattern” at scale. A datetime RE for New York Times news articles may not work for the articles at Le Figaro or Al Jazeera. Small typos throw off REs and produce nothing. This is a concrete Web example `data-timestamp="Thu Oct 05 2017 10:33:05 -0500">` that contains an unexpected space before “-0500”. This small typo can easily deem a complex and painstakingly constructed datetime RE obsolete. At such scale, any attempt to fully understand an RE and debug it in case it fails is futile.

In this thesis, we target the problem of detecting presence of entity mentions that follow or closely resemble patterns that can be described by REs. Unlike much of the previous body of work on this topic, we do not focus on learning/inferring highly accurate REs for entity identification *Prasse et al. (2012)*; *Bui and Zeng-Treitler (2014)*; *Li et al. (2008)*; *Banko et al. (2007)*; *Brauer et al. (2011)*; *Bartoli et al. (2016)*. We aim instead to show that deep learning can leverage imperfect REs and achieve very high accuracy while requiring only a modest human involvement. We proposed a human-in-the-loop (HIL) framework *Zhang et al. (2018a)* that uses human effort to both write a regex and to manually label the documents. As will be elaborated in the methodology, a regex is used to scan a document corpus and produce weak labels to pretrain an NN. Then, manually labeled substrings are used to fine-tune the network. The results showed that fine tuning a pretrained NN is superior to training it from scratch. Thus, the results indicate that writing a regex before manual labeling is highly desirable.

We then expand the proposed HIL framework to entity extraction task, which extract the piece of text denoting an entity *Zhang et al. (2019)*. However, the framework does not take into account the time needed to create a regex and regex writing expertise. We consider the problem from a practical perspective, where a human is given a fixed amount of time to interact with the ML system for entity recognition. We are not aware of published results that may inform Amy how to efficiently invest her time. We set to gain insight into this problem in this thesis.

### **1.3 Word Embedding for OOV words**

Representing words as real-valued vectors is crucial for modern natural language processing. Standard word embedding models such as Word2Vec and GloVe *Mikolov et al. (2013)*; *Pennington et al. (2014)*; *Le and Mikolov (2014)* learn to map a finite set of frequent words to vectors given a large unlabeled corpus by following the distributional hypothesis which states that similar words occur in similar contexts. As a result, semantically (*like* vs

*love*) and syntactically (*take vs took*) similar words are likely to have similar embeddings.

Inability to represent rare and unseen words, which is referred to as the out-of-vocabulary (OOV) problem, limits applicability of the standard embedding approaches. In many downstream tasks, the OOV problem is prominent and inevitable. For example, in the e-recruitment domain there is an extremely large number of ways to write a job title for the same type of a job. LinkedIn users who are software engineers can describe themselves by phrases such as *software developer*, *sw engi.*, *sw developppppper* or *sw and web application developer*, which are caused by different variants, abbreviations, misspellings, or compounding. In this case, standard embedding for a finite set of job titles is not helpful when recommending jobs to users with rare or unseen job titles. Similar issues exist in many other domains such as microblogs analysis or query rewriting *Grbovic et al. (2015,0)*.

One popular solution is to train a character-level neural network to reproduce the embeddings from a standard word embedding model. The trained neural network is then used to assign vectors to any input string, including OOV and rare words. However, these models tend to be able to capture the morphological characteristics of the words, but are weak on the semantic side. We enhance this approach and introduce an algorithm that iteratively refines and improves both the word embedding and the character-level neural network. We show that our method outperforms the existing algorithms on 5 word similarity data sets and 3 sentiment analysis data sets. We also demonstrate that our method can be successfully applied to *job title normalization*, which is an important problem in the e-recruitment domain.

## **1.4 Property Estimation for Crystal Materials**

Accurate estimation of chemical and physical properties is critical for new discoveries of molecules and materials. Numerical approximation methods such as ab initio DFT calculation have been studied for centuries by physicians. On one hand, it is costly in computation for large molecules. On the other hand, it can not make use of the large amount

of experimental data accumulated through high throughput experiments. Therefore, In the last decade, we have witnessed machine learning being applied to the field, since machine learning algorithms are efficient and effective given enormous training data.

Traditional machine learning requires domain experts involvement in designing a feature vector for each material. Recently, graph neural networks have been shown great success in the domain of drug design and material sciences, where organic molecules and crystal structures of materials are represented as attributed graph. We propose a simple graph representation for molecules and three neural network architectures in this thesis that is able to directly learn predictive functions from graphs.

The thesis is organized as followed. We give an introduction of the topics covered in this thesis in Chapter 1. We demonstrate a deep feature engineering approach for text classification in Chapter 2 to address the lacking of labels issue of deep learning. We propose a HIL framework for entity identification in Chapter 3, which leverages imperfect regular expressions in deep learning and achieves very high accuracy while requiring only a modest human involvement. In Chapter 4, we extend the HIL framework to entity recognition. We emphasize the trade-offs of distributing time in the proposed system with a small scale user study. In Chapter 5, we propose an iterative mimicking embedding model to deal with out-of-vocabulary words. In Chapter 6, we demonstrate the advantages of graph neural networks in property estimation for crystal materials. We conclude and discuss the future work in each chapter separately.

## CHAPTER 2

# DEEP FEATURE ENGINEERING FOR MESSAGE FILTERING AND SUMMARIZATION

### 2.1 Introduction

During the past decade, there has been plentiful evidence that social media can be very useful in improving the situational awareness during emergencies and disasters *Imran and Castillo (2015)*, and several research groups even proposed specific systems to extract, process, and summarize social media data for such purpose *Abel et al. (2012)*; *Ashktorab et al. (2014)*; *Imran et al. (2014)*. The prototypical situational awareness system relying on social media, such as the popular microblogging service Twitter, consists of a data collection, a data processing, and a visual interface component. The first critical challenge in making those systems work is selection of the disaster-related posts from a massive stream of posts pouring from services such as Twitter. The objective of this work is to propose a new approach for selection of informative posts, with a particular focus on the beginning stages of a disaster, when very little is known about the emergency and the ways in which people report about and react to it.

There are two main approaches for selection of the informative posts. One is the information retrieval approach, which is based on developing a lexicon of disaster-related unigrams or bigrams based on a study of posts from previous disasters and emergencies *Olteanu et al. (2014)*; *Temnikova et al. (2015)*. While there are terms that are commonly used over different disaster types (e.g. “victims”, “prayers”, “help”), each disaster is different and might require its own specialized lexicon. As such, pre-built lexicons cannot be

expected to have very large sensitivity and specificity during the emerging disasters. An alternative supervised learning approach is based on training a classifier that recognizes informative posts *Abel et al. (2012)*; *Ashktorab et al. (2014)*; *Imran et al. (2014)*. To train the classifier, there is a need to have access to labeled posts. If a sufficient number of labeled posts are available, the supervised approach is superior to the keyword-based approach. However, during the first few minutes or hours from the emergence of a disaster, the number of labeled posts might be too small to result in accurate classification.

The main issue when training from small labeled data is avoiding to build overly complex models that cannot generalize well. A typical remedy is to keep classification models simple. That is why logistic regression is a very popular model choice in the low-sample learning scenarios. A closely related issue is that the dimensionality of data is often too large and it complicates the issue of learning from small labeled data. For example, in case of classifying text data such as microblogs, a popular data representation approach is the bag of words representation, in which each word from a dictionary is treated as a feature. A standard remedy is to reduce dimensionality by feature selection and use some form of model regularization. However, when the number of labels is very small (less than a hundred), selection of the most informative words to be used as features can become unreliable. In particular, for all but the most frequent words, there might be too little evidence to estimate how useful they are with any degree of statistical certainty.

In this work, we propose to cluster similar words in groups and treat each cluster as a feature. By treating word clusters as features, the features would become less sparse and the estimate of their usefulness for classification would be more confident. In addition, word clusters would allow utilizing words that occur rarely or do not even occur in the labeled corpus. To create word clusters, we rely on unlabeled historical posts. One of the open questions in this study was what kind of unlabeled corpus is the most appropriate for this application. In the following text, we describe the details of our approach and provide an experimental evaluation of the approach on Twitter data from several disasters.

After we discover the informative messages from Twitter, the disaster related messages are likely to be redundant. For example, it is common that multiple people report on important events such as flooding or food shortage. Thus there is an opportunity to further reduce the number of messages by summarization. Though it is not related to learning, we show the method and results we use for summarization in this chapter for the restiveness.

## 2.2 Message Filtering Methodology

### 2.2.1 Problem Setup

Let us suppose we are given a corpus of  $n$  labeled documents  $\mathbf{D}_{train} = \{(d_1, y_1), (d_2, y_2), \dots, (d_n, y_n)\}$  and each document  $d_i$  is converted into a feature vector  $\mathbf{x}_i \in \mathbb{R}^m$ . For example, we could use the bag of words representation, where given a vocabulary  $\mathbf{V}$  the feature vector  $\mathbf{x}_i$  is a binary vector of length  $|\mathbf{V}|$ , whose  $j$ -th element indicates whether the  $j$ -th word from the vocabulary is present in the document  $d_i$ . By a slight extension,  $n$ -grams, part-of-speech tags, named entities, or word clusters could also be used for feature representation of documents. The label  $y_i$  of document  $d_i$  is a binary variable indicating whether it is disaster-related or not, where  $y_i = 1$  indicates that the document is disaster related and  $y_i = 0$  that it is not. Let us also assume that we are given a corpus of unlabeled documents  $\mathbf{D}_{unl} = \{d_1, d_2, \dots, d_N\}$  containing  $N$  documents that are available at the training time. The objective is to train a model  $f(x)$  whose output can be used to classify a document. We will assume that the output is real-valued and that larger values indicate a stronger likelihood of being disaster-related. For classification, we need to select a particular threshold  $\theta$  and classify all documents with outputs above the threshold as disaster-related and those below the threshold as unrelated.

### 2.2.2 Feature Selection Filters

The objective of feature selection is to reduce dimensionality of the feature vector. Feature selection filters are the simplest class of such algorithms that attempt to select only a subset of the most informative features and remove the rest. They do it so by calculating the score of each feature that measures how strong its correlation is with the classification label. Only the  $K$  features with the largest score are retained. There are many known scoring functions *Radivojac et al. (2004)*, among which the  $\chi^2$  statistics and Pointwise Mutual Information (PMI) are very popular for text classification. In this paper, we will use the PMI score, which is defined in the following way:

$$score(t) = PMI(t, pos) - PMI(t, neg) = \log_2 \frac{p(t|pos)}{p(t|neg)} \quad (2.2-1)$$

where  $p(t|pos)$  is the probability of feature  $t$  appearing in positive class, and  $p(t|neg)$  is the probability of feature  $t$  appearing in negative class.

### 2.2.3 Word Clusters as Features

The idea of document representation with word clusters is to group similar words and treat each cluster as a feature. In particular, given a vocabulary  $\mathbf{V}$  the objective is to assign them to one of  $K$  clusters. Given those clusters, each document  $d_i$  is converted into a binary feature vector  $\mathbf{x}_i$  of length  $K$  whose  $j$ -th element  $x_{ij}$  equals one if any of the words from the  $j$ -th cluster are present in the document and zero if not.

To create the clusters, we rely on the corpus of unlabeled documents  $\mathbf{D}_{unl} = \{d_1, d_2, \dots, d_N\}$ . Given the corpus, we will evaluate two ways of generating word clusters, as explained next.

**Brown clustering (BC)** *Brown et al. (1992)*. This is a traditional algorithm for word clustering. It is a hierarchical clustering algorithm that assigns a single cluster to each word and proceeds by merging the two clusters that result in the smallest reduction of global mutual information. The output of Brown clustering is a dendrogram of words,

which means we can cut at any level in the hierarchy to create word clusters.

**Clustering based on the skip-gram representation of words.** The idea of neural language models, of which the skip-gram model *Mikolov et al. (2013)* is a representative, is to learn low-dimensional vector representation of words. The skip-gram model defines the probability of word  $w_N$  being in the neighbourhood of a target word  $w_T$  in text by assuming that the words in the neighborhood are conditionally independent given the target word. It models probability of neighbour  $w_N$  given the target word  $w_T$  as a softmax function

$$p(w_N|w_T) = \frac{\exp(v'_{w_N} v_{w_T})}{\sum_{w=1}^{|\mathcal{V}|} \exp(v'_w v_{w_T})} \quad (2.2-2)$$

where  $v_w \in \mathbb{R}^p$  and  $v'_w \in \mathbb{R}^p$  are the “input” and “output”  $p$ -dimensional vector representations of word  $w$ . The input and output representations of every word are obtained by maximizing the log-likelihood of the model over a corpus of documents. In this paper, we use word2vec *Mikolov et al. (2013)*, which is a stochastic gradient algorithm that is commonly used to maximize this objective function. The resulting word representations place words occurring in similar contexts near each other.

Given the output word representations from the word2vec algorithm, we use the traditional k-means clustering algorithm to group all words from the vocabulary into  $K$  clusters.

### 2.3 Evaluate Message Filtering

The objective of our experimental evaluation is to obtain an insight into the performance of the proposed semi-supervised approach, which uses an unlabeled document corpus to create word clusters and trains a classifier on a small labeled document corpus where the word clusters are used as features.

### 2.3.1 Data Set

For our experiments, we used CrisisLexT6 data described in *Olteanu et al. (2014)*. It is a collection of tweets related to 6 natural disasters that occurred between October 2012 and July 2013: Sandy Hurricane (**SH**), Boston Bombings (**BB**), Oklahoma Tornado (**OT**), West Texas Explosion (**WE**), Alberta Floods (**AF**), Queensland Floods (**QF**).

For each disaster, the same number of tweets was selected using two different methods. One method relied on a predefined set of keywords appropriate for a particular disaster. Another method relies on selecting geocoded tweets originating from the affected areas during the onset of a disaster. The collected tweets were labeled as disaster or not disaster related by 100 workers from Crowdfunder. Out of the 60,000 Twitter IDs of the labeled tweets provided by the authors of *Olteanu et al. (2014)*, we were only able to download 70% of them and we summarize their basic statistics in Table 2.1.

Table 2.1: Basic data statistics of each disaster.

Disaster Type	# Location-based sample	# Positive	# Keyword-based sample	# Positive
Sandy Hurricane	3505	1000	3267	3058
Boston Bombings	3604	610	3651	3451
Oklahoma Tornado	4088	423	3897	3362
West Texas Explosion	4535	305	4245	4168
Alberta Floods	3712	367	4384	4044
Queensland Floods	3851	361	4302	4051

From Table 2.1, we observe a highly skewed distribution of classes depending on the selection criteria. In particular, the keyword-based sample contains a high fraction of positive (disaster-related) labels, while the location-based sample has a high fraction of negative labels. Unlike the previous work that merged tweets from those two samples and split them randomly into training and test sets for evaluation *Imran et al. (2014,0)*; *Olteanu et al. (2014)*, in this work we used only the tweets obtained by the location search. We discarded the portion of tweets obtained by the keyword-based selection. Our reasoning was that as long as the keywords are well defined by authorities, virtually all tweets sampled by the

keyword search can be assumed as positives. Identifying disaster-related Tweets within the location-based sample is a more challenging classification problem, which can be particularly important during the initial minutes or hours of an emerging disaster.

### 2.3.2 Experiment Design

In our experiments we wanted to compare the proposed approach with the standard supervised learning approach that uses feature selection on the bag of words and trains a logistic regression classifier. We also wanted to explore impact of a range of choices on the accuracy, including the choice of the word clustering algorithm, the choice of an unlabeled document corpus, the size of the labeled corpus, the number of features, and the choice of hyperparameters.

*Preprocessing.* All tweets were tokenized using the tool from `twitter_nlp`<sup>1</sup>. Stop-words, URLs, and user mentions (`@username`), tokens too short (2 characters or less), tokens too long (16 characters or more) were removed from the tweets. Given the tweets preprocessed in this way, we divided the location-based tweets of each disaster randomly into two subsets: 70% as training set and 30% as the test set. The test set was not used in any stage of classifier building and was used solely to calculate the accuracy.

*Classification Algorithm.* In all experiments we relied on Logistic Regression (**LR**) with  $L_2$  regularization. The regularization hyperparameter was obtained using leave-one-out cross-validation on the training data.

*Choice of training size.* We varied the training size in the range  $t = [20, 50, 100, 200, 500, 1000]$ . This allowed to achieve an insight into the impact of the training size on accuracy and its relative impact on different algorithms. The smallest training sizes of 20 and 50 are representative of scenarios with extremely limited labeled data sets that might be expected shortly after the outset of a disaster, while the larger training sizes are representatives of labeled data sets that can become available with a more significant delay.

---

<sup>1</sup>[https://github.com/aritter/twitter\\_nlp](https://github.com/aritter/twitter_nlp)

*Choice of features.* We compared two clustering algorithms described in the Methodology, the Brown Clustering (**BC**) and the clustering based on word2vec (**W2V**). We denote feature selection based on the traditional bag-of-words as **BOW**. We varied the number of clusters in the range of  $K = [50, 100, 200, 500, 1000, 2000]$ . To be comparable, for **BOW** we used feature selection based on PMI score to train Logistic Regression based on the top  $K$  scoring words.

*Choice of the unlabeled corpus for word clustering.* We explored several types of unlabeled Twitter sets for word clustering using **BC** and **W2V** approaches:

- $\mathbf{D}_{50K}$ . For any disaster, the unlabeled corpus contains all available CrisisLexT6 tweets from the other 5 disasters.
- $\mathbf{D}_{50Kp}$ . For any disaster, the unlabeled corpus contains all positively labeled CrisisLexT6 tweets from the other 5 disasters.
- $\mathbf{D}_{3K}$ . For any disaster the corpus contains all available keyword-based tweets from that disaster.

In addition, we downloaded a pre-trained *word2vec* matrix  $\mathbf{W}$  with word vector dimension  $p = 400$  and vocabulary size  $|\mathbf{V}| \sim 3,000,000$  that was trained on 400 million Tweets.

By combining different choices for feature construction, we obtained 8 different algorithms that we evaluated experimentally: bag of words (**BOW**), bag of Brown clusters ( $\mathbf{BC}_{50K}$ ,  $\mathbf{BC}_{50Kp}$ ,  $\mathbf{BC}_{3K}$ ), and bag of word2vec clusters ( $\mathbf{W2V}_{50K}$ ,  $\mathbf{W2V}_{50Kp}$ ,  $\mathbf{W2V}_{3K}$ ,  $\mathbf{W2V}_{400M}$ )

*Other experimental settings.* We run *word2vec* with negative sampling rate  $10^{-3}$ , context window size 100, and word vector dimension 20 (due to a relatively small unlabeled corpus). For each choice of the feature construction algorithms, each training set size, and each number of features  $K$ , we run Logistic Regression 10 times on randomly sampled

subsets from the training set. The average Area Under the ROC Curve (AUC) of the 10 random sampling experiments is reported.

### 2.3.3 Results

Our results provide insights into several interesting questions related to recognizing disaster-related tweets during the emerging disasters. Figure 2.1 is the most comprehensive view into our results. For each disaster and each training size, we are showing 8 curves, representing 8 different feature construction algorithms, as a function of the number of features. However, to answer our posed questions, we summarized those results with 4 tables, as will be described in the following.

**Are word clusters better than BOW?** In Table 2.2, we compare the accuracies of the best word clustering and the best **BOW** classifier for each of the 6 disasters and for each training data size. Notice that for **BOW** approach at a specific training size, we report the best accuracy among different choices of  $K$ , which means we are treating the  $K$  as a hyperparameter during training. For the 7 semi-supervised approaches, we report the best accuracy from that group. When the training data size is 20, 50, and 100, the word clustering approach is superior to **BOW** on all but the Oklahoma Tornado (**OT**) data set. This result strongly indicates that an unlabeled Twitter corpus could provide a very useful information when there is a severe deficit of labeled tweets for an emerging disaster. As the number of labeled tweets is exceeding 100, the benefit from the unlabeled corpus fades, to the extent that **BOW** is more accurate than the word clusters on all 6 disasters when the training data size reaches 1,000. This is an expected result, since such a large training data set is sufficient to identify the most informative disaster-related words.

**Does the choice of a word clustering algorithm make a difference?** In Table 2.3 we compare the best accuracy obtained by **BC** clustering to the best accuracy by **W2V** clustering. For both **BC** and **W2V**, we only considered  $\mathbf{D}_{50K}$ ,  $\mathbf{D}_{50Kp}$ , and  $\mathbf{D}_{3K}$  corpuses. None of the algorithms is superior over all 6 disasters. **W2V** is superior on **OT**, **AF**, and

Table 2.2: Summary results to compare Semi-supervised (SS) and BOW algorithms.

Train	SH		BB		OT	
	SS*	BOW*	SS*	BOW*	SS*	BOW*
<b>20</b>	<b>0.734</b>	0.69	<b>0.714</b>	0.7	0.688	0.693
<b>50</b>	<b>0.76</b>	<b>0.76</b>	<b>0.761</b>	0.707	0.746	0.746
<b>100</b>	<b>0.8</b>	<b>0.8</b>	<b>0.794</b>	0.774	0.795	0.803
<b>200</b>	0.838	0.844	<b>0.816</b>	0.811	0.846	0.857
<b>500</b>	0.865	0.878	0.844	0.874	0.887	0.897
<b>1000</b>	0.883	0.892	0.868	0.892	0.909	0.918
Train	WE		AF		QF	
	SS*	BOW*	SS*	BOW*	SS*	BOW*
<b>20</b>	<b>0.754</b>	0.709	<b>0.746</b>	0.686	<b>0.716</b>	0.645
<b>50</b>	<b>0.77</b>	0.774	<b>0.798</b>	0.753	<b>0.764</b>	0.702
<b>100</b>	<b>0.83</b>	0.807	<b>0.851</b>	0.848	<b>0.812</b>	0.783
<b>200</b>	0.863	0.865	<b>0.907</b>	0.876	<b>0.856</b>	0.85
<b>500</b>	0.93	0.921	<b>0.936</b>	0.928	<b>0.9</b>	0.9
<b>1000</b>	0.935	0.947	0.942	0.949	0.929	0.934

QF, BC is superior on SH, and they are similar on the remaining two disasters. This is a slightly surprising result, considering the age of Brown clustering and the recent hype surrounding the word2vec algorithm. Studying this result more in depth would certainly be worthwhile. For the purposes of this paper, our conclusion is that both algorithms have their strengths, but that if we have to choose one, it would be W2V.

Table 2.3: Comparing two clustering algorithms BC and W2V.

Train	SH		BB		OT	
	BC	W2V	BC	W2V	BC	W2V
<b>20</b>	<b>0.734</b>	0.65	0.707	<b>0.714</b>	0.639	<b>0.682</b>
<b>50</b>	<b>0.758</b>	0.736	<b>0.761</b>	0.755	0.706	<b>0.741</b>
<b>100</b>	<b>0.792</b>	0.764	<b>0.794</b>	0.784	0.762	<b>0.789</b>
Train	WE		AF		QF	
	BC	W2V	BC	W2V	BC	W2V
<b>20</b>	0.73	<b>0.754</b>	0.575	<b>0.613</b>	0.591	<b>0.651</b>
<b>50</b>	<b>0.759</b>	0.748	0.617	<b>0.678</b>	0.668	<b>0.702</b>
<b>100</b>	0.801	<b>0.82</b>	0.681	<b>0.689</b>	0.724	<b>0.761</b>

**What is the impact of the unlabeled corpus choice?** In Table 2.4 we are showing the accuracies of the best W2V algorithm for four different choices of an unlabeled corpus:

$\mathbf{D}_{50K}$ ,  $\mathbf{D}_{50Kp}$ ,  $\mathbf{D}_{3K}$ , and  $\mathbf{D}_{400M}$ . On four disasters (**SH**, **OT**, **AF**, **QF**), the 400 million Twitter corpus results in the best accuracy, it is competitive on **WE**, and only on **BB** it is less accurate than  $\mathbf{D}_{50K}$ . This result is a strong indicator that mantra “the more data the better” works in the emerging disaster scenario. Confirming this, it is interesting to see that using  $\mathbf{D}_{50K}$  is in general better than using its positively labeled subset  $\mathbf{D}_{50Kp}$ . Finally, using the location-based tweets from the emerging disaster itself does not seem to be particularly effective, and it could probably be attributed to the small size of this corpus.

Table 2.4: Comparing 4 types of unlabeled corpus.

Train	SH				BB			
	3K	50Kp	50K	400M	3K	50Kp	50K	400M
<b>20</b>	0.646	0.693	0.665	<b>0.696</b>	0.694	0.658	<b>0.714</b>	0.662
<b>50</b>	0.681	0.726	0.736	<b>0.746</b>	0.722	0.683	<b>0.755</b>	0.737
<b>100</b>	0.661	0.776	0.764	<b>0.795</b>	0.737	0.733	<b>0.784</b>	0.768
Train	OT				WE			
	3K	50Kp	50K	400M	3K	50Kp	50K	400M
<b>20</b>	0.604	0.651	0.682	<b>0.688</b>	0.661	0.702	<b>0.754</b>	0.71
<b>50</b>	0.692	0.708	0.741	<b>0.746</b>	0.719	0.702	0.748	<b>0.752</b>
<b>100</b>	0.695	0.732	0.789	<b>0.795</b>	0.774	0.777	<b>0.82</b>	<b>0.82</b>
Train	AF				QF			
	3K	50Kp	50K	400M	3K	50Kp	50K	400M
<b>20</b>	0.725	0.608	0.613	<b>0.746</b>	0.608	0.589	0.651	<b>0.716</b>
<b>50</b>	0.781	0.636	0.678	<b>0.798</b>	0.636	0.653	0.702	<b>0.764</b>
<b>100</b>	0.811	0.669	0.689	<b>0.851</b>	0.703	0.698	0.761	<b>0.812</b>

**How many word clusters?** In Table 2.5 we show the best choice of the number of clusters for **W2V** obtained from the 400 million corpus. When the training data size is small, the smaller number of clusters is preferable. This result illustrates that the benefit of merging similar words into a smaller number of clusters to create dense features outweighs a potential loss of semantic cohesion in those clusters. As expected, for larger training sets, cluster cohesion becomes an important driving factor for high accuracy.

**Impact of training size.** Not to be lost in the previous discussion, it is evident that the number of labeled tweets has a strong influence on accuracy, and we can observe a robust increase in accuracy with the training data size from Figure 2.1.

Table 2.5: The number of clusters (**K**) to achieve best performance for  $W2V_{400M}$

Train	SH	BB	OT	WE	AF	QF
20	500	2000	500	100	500	200
50	1000	1000	100	1000	1000	100
100	1000	2000	500	500	2000	1000
200	2000	2000	500	2000	2000	2000
500	2000	2000	2000	2000	1000	1000
1000	2000	2000	2000	1000	2000	1000

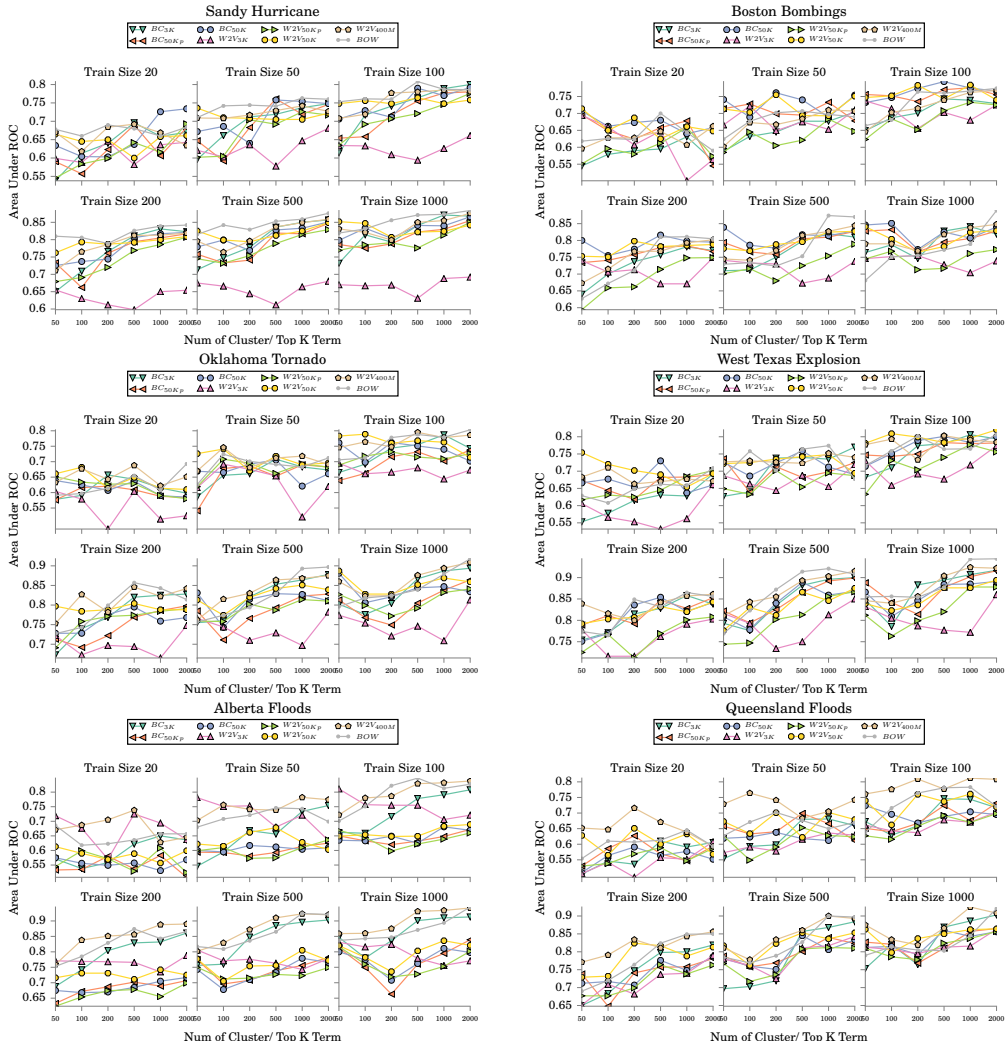


Figure 2.1: Detailed performance comparison for 6 disasters.

## 2.4 Message Summarization Methodology

Let us suppose we are given a set of  $N$  messages. We formulate the problem as a subset selection problem following work in *Rudra et al. (2016a,0)*, where the objective is to select

a subset of  $n$  messages that maximize the *informativeness*. In this work, we also maximize *semantic completeness*, to be defined later in the text.

### 2.4.1 Informativeness score

Content words can be provided by domain experts, describing disaster situations. In our experiments, we simply use *nouns*, *proper nouns*, *verbs* and *numbers* as content words. Let us denote the set of  $V$  unique content words as  $CW = \{cw_1, cw_2, \dots, cw_V\}$ . Emergency responders at CBE can define a weight for each content word  $cw_j$  as  $W(cw_j)$ . Then, informativeness of message  $i$  can be calculated as the sum of the weights of its content words:  $I(m_i) = \sum_j^{v_i} W(cw_j)$ .

Table 2.6: Notation used in the summarization stage.

Notation	Description
$x_i, y_j$	Binary values with 1 if the message $i$ or the content word $j$ is selected and 0 otherwise
$I(m_i)$	Informativeness score of message $i$
$W(cw_j)$	Weight of content word $j$
$L$	Maximum number of messages can be selected
$\mathcal{T}(cw_j)$	Indexes of messages of which $cw_j$ is a member
$\mathcal{W}(m_i)$	Indexes of words in message $m_i$
$\mathcal{C}_k$	Indexes of messages in semantic cluster $k$
$K$	Total number of semantic clusters
$N$	Total number of messages in the pool

### 2.4.2 Semantic completeness

While informativeness score is a useful objective criterion for summarization, it is not sufficient to reduce redundancy in messages. For example, suppose the following three messages are given the same informativeness score:

- (i) Water has entered my house.
- (ii) Streets are flooded and not one rain drop came down so far.
- (iii) Attention everyone I lost power.

Instead of picking (i) and (ii) out of them, we should select (ii) and (iii), since they refer to two semantic groups and cover two aspects of disaster.

Thus, we propose a preprocessing step that clusters messages. We first represent the  $N$  messages as a matrix  $R^{N \times V}$ , with value at cell  $(i, j)$  being the count of word  $j$  in message  $i$ . Since the count matrix is sparse, we use the *SVD* algorithm to reduce the number of columns to  $d$ ,  $d \ll V$ . Then, we apply k-means algorithm to group all messages into  $K$  clusters.

The resulting message selection optimization algorithm is described in equation(2.4-3). The notation is explained in Table 2.6.

$$\begin{aligned}
 & \underset{x_i, y_j}{\text{Maximize}} && \sum_{i=1}^N I(m_i)x_i + \sum_{j=1}^V W(cw_j)y_j \\
 & \text{subject to} && \sum_{i=1}^N x_i < L \\
 & && \sum_{i \in \mathcal{T}(cw_j)} x_i \geq y_j \\
 & && \sum_{j \in \mathcal{W}(m_i)} y_j \geq |\mathcal{W}(m_i)|x_i \\
 & && \sum_{i \in \mathcal{C}_k} x_i \leq \tau \sum_{i=1}^N x_i, \text{ for } k \text{ in } 1, 2, \dots, K.
 \end{aligned} \tag{2.4-3}$$

The above optimization maximizes the informativeness and the semantic coverage of the selected messages. The first constraint specifies the budget, which is the number of messages the summarizer wants to extract. The second and third constraints ensure validity of the selected messages, i.e., all of the content words in a selected message are selected and a selected content word must reside in a selected message. The final set of  $K$  constraints de-

finds a maximum number of messages that could be selected from any given cluster, which is controlled by parameter  $\tau \in (0, 1]$ . The formulated problem is solved using GUROBI Optimizer *Inc* (2014).

## 2.5 Evaluate Message Summarization

### 2.5.1 Data Sets

We evaluated performance of filtering and summarization using real world data. We collected 4.7 million tweets posted during 2012 Sandy Hurricane, which affected the U.S. Northeast from 10/28/2012 to 11/03/2012. We focused on a subset of tweets generated from 4 large metro areas in the region: New York, NY (NYC); Philadelphia, PA (Phila.); Boston, MA (Boston) and Washington, DC (DC).

### 2.5.2 Simulated Settings

Typically, a CBE will divide the targeted area into small *cells* (e.g., school districts). Since we did not have this information handy, we divided each of the 4 cities into a grid of equally sized cells with size  $D_{fix} \times D_{fix}$ .

Note that in this setting the traffic load of each cell is highly imbalanced, because of varying population densities in typical big cities. Figure 2.2 shows the approximate population density of New York City metro area by considering the number of unique users who tweeted. We assume  $D_{fix} = 500$  for Figure 2.2. The dense regions are denoted by dark color. The figure shows that the population is highly concentrated on the island of Manhattan.

Dividing a targeted area into small regions results in a large number of cells. For example, since the New York region is of size  $48km \times 48km$ , we get around 10,000 cells of size  $500m \times 500m$ . We discovered that the densest 10% of cells cover  $\sim 70\%$  of the population. The long tail effect is illustrated in Figure 2.3. Since for the less dense cells the network

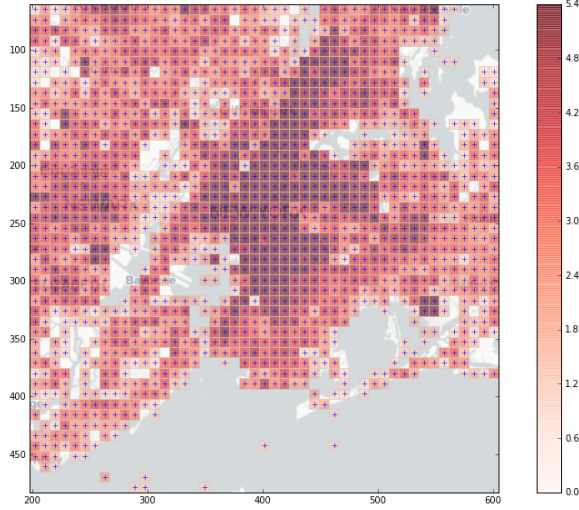


Figure 2.2: The Manhattan area is partitioned into cells with size  $D_{fix} = 500m$ . The tweet densities are shown in different color scale.

congestion is not a significant issue, we only evaluated our filtering and summarization algorithms on the densest 10% of the cells.

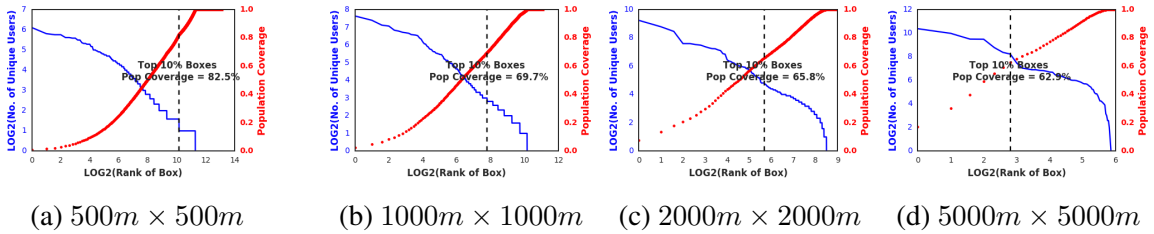


Figure 2.3: The population density plot for different  $D_{fix}$ . X-axis:  $\log_2(\text{rank})$  of cell by density; Left Y-axis: number of unique users; Right Y-axis: fraction of unique users.

**Filter:** To train the filter we used a labeled dataset  $M_{train}$ , obtained from *Olteanu et al. (2014)*. It contains 3,505 tweets labeled as 1 if they are related to Sandy Hurricane and 0 otherwise. To create the word clusters we downloaded word vectors pre-trained by *wort2vec* algorithm on 400 million unlabeled tweets from the Internet. Then, we created 2,000 clusters using k-means algorithm. We then trained the filter using logistic regression.

**Summarizer:** The key to the summarizer is the definition of the content words and their weights. In reality, the content words and their weights can be passed from authorities or domain experts to better guide the summarization. For simplicity, we extract content

Table 2.7: Number (percentage %) of messages after filtering.

	<b>NYC</b>	<b>Phila.</b>	<b>Boston</b>	<b>DC</b>
$ \mathcal{M} $	444, 016	121, 122	86, 731	100, 508
$ \mathcal{S}_1 (p(\mathcal{S}_1))$	16, 201(3.65)	2, 304 (1.9)	1, 294(1.5)	1, 563(1.6)

Table 2.8: Number (percentage %) of messages kept after summarization.

	$D_{fix} = 500m$				$D_{fix} = 1000m$			
	$L = 10$	$L = 20$	$L = 50$	$L = 100$	$L = 10$	$L = 20$	$L = 50$	$L = 100$
<b>NYC</b>	2990(0.67)	5277(1.19)	7811(1.76)	8656(1.95)	1250(0.28)	2500(0.56)	5244(1.18)	7149(1.61)
<b>Philla.</b>	689(0.57)	753(0.62)	753(0.62)	753(0.62)	454(0.37)	32(0.03)	664(0.55)	664(0.55)
<b>Boston</b>	392(0.45)	441(0.51)	441(0.51)	441(0.51)	190(0.22)	278(0.32)	278(0.32)	278(0.32)
<b>DC</b>	466(0.46)	510(0.51)	510(0.51)	510(0.51)	224(0.22)	359(0.36)	359(0.36)	359(0.36)
	$D_{fix} = 2000m$				$D_{fix} = 5000m$			
	$L = 10$	$L = 20$	$L = 50$	$L = 100$	$L = 10$	$L = 20$	$L = 50$	$L = 100$
<b>NYC</b>	370(0.08)	740(0.17)	1850(0.42)	3664(0.83)	100(0.02)	200(0.05)	500(0.11)	1000(0.23)
<b>Philla.</b>	128(0.11)	232(0.19)	304(0.25)	304(0.25)	80(0.07)	136(0.11)	208(0.17)	208(0.17)
<b>Boston</b>	80(0.09)	152(0.18)	208(0.24)	208(0.24)	75(0.09)	131(0.15)	211(0.24)	259(0.30)
<b>DC</b>	80(0.08)	152(0.15)	208(0.21)	208(0.21)	48(0.05)	72(0.07)	104(0.10)	112(0.11)

words automatically from all tweets in an area as follows:

1. **Content words:** We first use the tweet POS (part-of-speech) tagger from CMU *Gimpel et al.* (2011) to tag all of the tweets from a city. Then *nouns*, *proper nouns*, *numbers* and *verbs* with more than 5 appearances are selected as content words.
2. **Weights of content words:** For our experiments, we scored content word  $cw$  as:  $W(cw) = \log_2(f_{cw} + 1) * \log_2 \frac{N}{|\mathcal{T}(cw)|}$ , where  $f_{cw}$  is the total count of  $cw$  and  $\mathcal{T}$  is the number of tweets containing  $cw$ .

Another important decision for the summarizer is the message clustering procedure, which helps us achieve a better variety of selected messages. In our experiment we clustered messages in a cell into 10 clusters and removed the smallest 2 clusters.

### 2.5.3 Results

We evaluated the performance of our proposed message extraction framework both quantitatively and qualitatively.

Table 2.9: Crisis Related categories *Imran et al. (2015,0)*.

Categories	Topics
$C_1$	Injured or dead people
$C_2$	Missing, trapped, or found people
$C_3$	Displaced people and evacuations
$C_4$	Infrastructure and utilities damage
$C_5$	Donation needs/volunteering services
$C_6$	Caution and advice
$C_7$	Sympathy and emotional support
$C_8$	Other useful information

**Quantitative Analysis** We would like to check the percentage of messages kept after filtering and after summarization. Suppose  $\mathcal{M}$  is the set of all tweets for a specific city and  $\mathcal{S}_1, \mathcal{S}_2$  are the remaining tweets after filtering and after summarization respectively, the percentage of messages kept in  $\mathcal{S}_i$  is then defined as  $p(\mathcal{S}_i) = \frac{|\mathcal{S}_i|}{|\mathcal{M}|} \times 100\%, i \in \{1, 2\}$ .

The numbers (percentages) of messages kept after filter are reported in Table 2.7. The fraction of crisis-related tweets is twice larger in NYC compared to other cities, which is expected because NYC was more heavily impacted by Sandy Hurricane.

Then, we checked whether our summarizer works as expected when we vary the cell size  $D_{fix}$  and the message budget  $L$ . We explored the following limits on the number of messages per cell,  $L = [10, 20, 50, 100]$ . The cell size  $D_{fix}$  ranged from  $[500m, 1000m, 2000m, 5000m]$ . We fixed  $\tau = 0.125$ , which encourages about the same number of messages to be selected from each cluster. The numbers and percentages are reported in Table 2.8. From Table 2.8 we can observe that:

1. As we increase the cell size, we will keep a smaller fraction of messages.
2. As we increase the size of budget, we will keep more messages.

**Qualitative Analysis** Besides the number of extracted messages, we also seek a good coverage of crisis-related topics.

One standard and widely accepted crisis-related ontology proposed in *Imran et al.*

(2015,0) contains 8 categories ( $C_1-C_8$ ) as shown in Table 2.9. We measured the distribution of message categories of the filtered tweets and the final summarizations. To get the number of tweets in each category, we took a labeled data set provided by *Imran et al.* (2014), which contains 2,013 tweets from Odile Hurricane in 2014, and trained a multi-class classifier using the described method for the filter component. Then the classifier was used to predict the category of tweets from Sandy Hurricane.

We first get the set of tweets after filtering  $\mathcal{S}_1$  for each city from Table 2.7. We also get a set of tweets after summarization  $\mathcal{S}_2$  for each city using  $L = 50$ ,  $D_{fix} = 2000m$  and  $\tau = 0.125$  from Table 2.8. Therefore, the percentage of category  $k$  in  $\mathcal{S}_i$  is defined as:  $p_{\mathcal{S}_i}(k) = \frac{n_k}{|\mathcal{S}_i|} \times 100\%$ ,  $i \in \{1, 2\}$ , where  $n_k$  is the number of tweets in  $\mathcal{S}_i$  falling in category  $k$ . Table 2.10 shows how the distribution of message categories changes from  $\mathcal{S}_1$  to  $\mathcal{S}_2$ .

One observation is that the eight categories are not evenly distributed. A small proportion of tweets is classified into categories 1, 2, 3 and 6. It might be because the death or injury of people were rare events. Fair amount of tweets was related to infrastructure damage, donations, and words of sympathy. Many tweets were classified as other useful information.

Comparing the distribution of message categories in  $\mathcal{S}_1$  and that in  $\mathcal{S}_2$ , we do not see dramatic differences. However, the percentage of tweets of category 8 is reduced by 5% – 10%, while the percentages of other categories are lifted slightly. The results illustrate that the summarizer reduced the imbalance slightly.

Finally, we set  $L = 5$ ,  $D_{fix} = 1000m$ ,  $\tau = 1$  (by which we do not force the clustering constraints) for all cities and apply our message extraction framework on the most dense cell in each city. The most important 5 messages from the 4 cities are shown in Table 2.11. The two numbers in the parentheses denote the number of selected tweets and the total number of tweets, respectively. During Sandy Hurricane, the Manhattan area was considerably affected. Therefore, damage and disruptions, such as flight delay, power outage, were commonly reported. Users from the other three cities were worried about power out-

Table 2.10: Distribution of message categories after filtering ( $\mathcal{S}_1$ ) & after summarization ( $\mathcal{S}_2$ ).

		<b>NYC</b>	<b>Phila.</b>	<b>Boston</b>	<b>DC</b>
$\mathcal{S}_1$	$C_1$	0.00	0.00	0.00	0.00
	$C_2$	0.09	0.04	0.00	0.00
	$C_3$	0.01	0.00	0.00	0.00
	$C_4$	4.44	3.04	3.86	3.58
	$C_5$	3.11	2.95	3.01	3.26
	$C_6$	0.01	0.00	0.00	0.00
	$C_7$	4.30	7.29	7.57	5.37
	$C_8$	84.67	83.29	81.07	84.52
$\mathcal{S}_2$	$C_1$	0.00	0.00	0.00	0.00
	$C_2$	0.15	0.00	0.00	0.00
	$C_3$	0.00	0.00	0.00	0.00
	$C_4$	4.46	2.96	7.21	5.29
	$C_5$	4.61	4.61	4.33	6.25
	$C_6$	0.00	0.00	0.00	0.00
	$C_7$	4.76	12.83	8.17	7.21
	$C_8$	80.95	73.03	75.48	79.33

age and were sending prayers to the people along the eastern shore, but there is much less reporting of the damage and disturbances.

## 2.6 Conclusions

In this paper we addressed the problem of retrieving disaster-related tweets shortly after the onset of a disaster. In such a scenario, we can expect to have access to a very limited number of labeled tweets. The accuracy of classifiers trained on such small data would be limited. To remedy this problem, we proposed a semi-supervised approach that can utilize a large unlabeled corpus of tweets to create word clusters and use them as features for classification. Our experiments on Twitter data from 6 disasters strongly indicate that the proposed semi-supervised approach could most often result in the improvements in accuracy as compared to the traditional supervised learning approach that uses feature selection on the bag of words features. Our study also provides useful insights into different mod-

Table 2.11: The 5 selected messages from each city.

<p><b>NYC (5/365)</b></p> <p>Flight out on Monday is cancelled, looks like flying today is not an option. Got a hotel reservation here in NY until Thur. // New Jersey will be under water for weeks after this storm and Chris Christie is "going to watch the jets for a couple of hours" // 57th Street closed off. Times Square like a ghost town. Sandy has sent "The City That Never Sleeps" into La La Land. // Wow! RT BREAKING - Reuters witness: 19 Con Edison workers trapped by rising floodwaters at Manhattan power station.? // Gov Cuomo- subway system damaged heavily. A 30 year vet said hes never seen such damage. East river flooded into ground zero #sandy // lower manhattan power outage Midtown Manhattan</p>
<p><b>Phila. (5/56)</b></p> <p>Happy Sunday! I'm in Philadelphia for a trade show, Hurricane Sandy is approaching.I wonder how bad this storm of the century will be? ????? // Ummm pool party at Joani's??? Let's play games by candle light whether we have power or not. // It's been a crazy day on the east coast... Some of the footage of the shore is insane!! I hope you're on high ground! // Turned off gps and data, turned on wifi to conserve battery in case the power goes out tonight. The first real precaution I've taken. #Sandy // 2 minutes of #sandy from CC Philly Monday night #6abcSandy <a href="http://t.co/HE2q37jy">http://t.co/HE2q37jy</a></p>
<p><b>Boston (5/49)</b></p> <p>@SpeedoUSA: Sending our thoughts to those on the East Coast affected by #sandy #frankenstorm" // yea my dad says he hasn't seen wind this bad since hurricane bob in 91' plus it got worse at high tide // The only thing worse than Hurricane Sandy is watching Piers Morgan talk about. #sandy // Local news decided to stop weather broadcast in favor of Dancing w/the Stars, you know that signals the end of the storm here in Beantown // So no power outage in the back bay == karma from last year's senseless 3 day outage. I guess we're even now.</p>
<p><b>DC (5/42)</b></p> <p>Braving the storm while the supermarket is still open. If I don't tweet for a while, send help! #SandyDC // @bethanyshondark The hype was right! Hoping I don't lose power in DC... // @ckbarrett Still raining and freezing, and most everything is shut down. Alex made it home safely this morning from his night shift. #sandy // Offices are fine, no water damage, electricity and Internet... What else could we possibly need (@ Clearly Innovative) <a href="http://t.co/GJqp21cl">http://t.co/GJqp21cl</a> // There are very few times I wish I was allowed to donate blood. Right now is one of those times. #sandy</p>

eling choices when using the proposed approach. While “the bigger the better” mantra in data science mostly holds true in this application, by a careful look at the results, it is also evident that “one size does not fit all,” and that many modelling choices have differing effect on different types of disasters. More effort is needed to gain a better insight into those differences and design approaches with more consistent behavior across a wide range of situational awareness scenarios.

## CHAPTER 3

# REGULAR EXPRESSION GUIDED ENTITY MENTION MINING FROM NOISY WEB DATA

### 3.1 Introduction

Named Entity Recognition (NER) is the task of automatically locating, extracting, and classifying contiguous pieces of strings, which represent entities of interest, in text. Classification (or typing) seeks to assign pre-defined categories (e.g., person, organization, location, expressions of time, monetary values, and emails) to each extracted piece of text. NER is a subtask of the broader problem of Information Extraction (IE) from text *Chang et al. (2006); Etzioni et al. (2005); Finkel and Manning (2009); Nadeau and Sekine (2007); Shen et al. (2015)*. Named entities usually refer to entity names that describe unique identifiers of people, locations, movies, events, and organizations. There is a large class of entities that are not “named,” such as expressions of time, emails, and course identifiers. Their main characteristic is that they often follow an underlying syntactical pattern, which can be fully described or well approximated by Regular Expressions (REs).

Despite being the workhorse of many entity recognition tasks, REs have a number of drawbacks. The construction of highly accurate REs is difficult and requires specific technical skills. For a simple task such as recognizing emails, there are 361 REs proposed in [RegExLib.com](http://RegExLib.com). Moreover, REs are brittle and difficult to maintain. These obstacles have motivated the work on automatic inference of REs *Banko et al. (2007); Li et al. (2008); Bartoli et al. (2018)* where the objective is to develop approaches that are fast and deployable in real time. However, the existing approaches tend to require large number of examples to

cover both the alphabet and the possible syntactic patterns. Moreover, they often produce overly complicated or long REs or combinations of REs *Bartoli et al.* (2016). One of the most complete REs for emails has nearly 6,500 characters *Millner* (2008)!

Web documents are an important domain for data extraction. Importantly, the Web is not a place where data follows "underlying syntactical pattern" at scale. A datetime RE for New York Times news articles may not work for the articles at Le Figaro or Al Jazeera. Small typos throw off REs and produce nothing. This is a concrete Web example `data-timestamp="Thu Oct 05 2017 10:33:05 -0500">` that contains an unexpected space before "-0500". This small typo can easily deem a complex and painstakingly constructed datetime RE obsolete. At such scale, any attempt to fully understand an RE and debug it in case it fails is futile. New automated or semi-automated tools are needed to either supersede REs or to work in tandem with REs. In this work, we focus on the later.

In this paper, we target the problem of detecting presence of entity mentions that follow or closely resemble patterns that can be described by REs. Unlike much of the previous body of work on this topic, we do not focus on learning/inferring highly accurate REs for entity identification *Prasse et al.* (2012); *Bui and Zeng-Treitler* (2014); *Li et al.* (2008); *Banko et al.* (2007); *Brauer et al.* (2011); *Bartoli et al.* (2016). We aim instead to show that deep learning can leverage imperfect REs and achieve very high accuracy while requiring only a modest human involvement.

Suppose the goal is to recognize *datetime* string expressions. We use some reasonable REs  $R$  for datetime to generate a weakly labeled training dataset from a large corpus of Web documents, e.g., news articles. We train a deep neural network on this data. Denote this model  $M_{RE}$ . To our surprise,  $M_{RE}$  is already capable of recognizing the presence of datetime expressions beyond those recognized by  $R$ . Furthermore, with the addition of a very small number of training samples (between 20 - 50 instances) from a human labeler, we obtain a model  $M_{RE+human}$  that is superior to  $M_{RE}$  by a significant margin. In general, complex systems do not generalize easily with the addition of new data, because

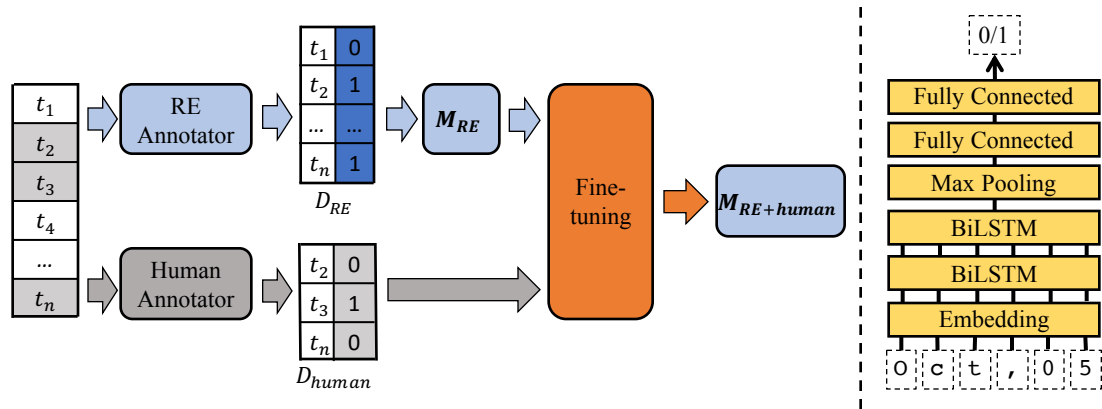


Figure 3.1: Overview of the solution (left) and deep learning architecture (right) used to train model  $M_{RE}$ .

the amount of labeled data required to provide a good coverage grows exponentially with the complexity of the problem *Chiu and Nichols (2015); Lample et al. (2016); Huang et al. (2015); Mahajan et al. (2018)*. We show that there is an opportunity for faster convergence to a generalized recognizer for this class of entities.

The main contributions in this paper are:

- We show how starting from REs  $R$  that recognizes a fraction of entities of a given type  $\mathcal{E}$  (say, email) we can pretrain a deep neural network (NN) model which can be a richer recognizer of entities of type  $\mathcal{E}$  than  $R$ .
- We show that we can fine tune the pretrained model to recognize an even larger set of entities of type  $\mathcal{E}$  with the addition of a small number of labeled instances, as small as 20.

The paper is organized as follows. Section 4.2 gives a brief overview of the related work. Section 6.3 describes our method. Section 4.4 presents our methodology for parameter learning and the overall experimental setup. Section 3.5 describes the experimental results. Finally, Section 4.7 concludes the paper.

## 3.2 Related Work

This section will mention several lines of research we deem the most related to our work.

The problem of inducing regular expressions has been an active area of work for more than two decades. One line of work focuses on improving the initial REs by identifying the true or false matches *Li et al. (2008)*; *Murthy et al. (2012)*; *Cetinkaya (2007)*; *Cochran et al. (2015)*. Another line of work attempts to directly induce REs from positive and negative sample strings *Fernau (2009)*; *Denis (2001)*. The common approaches include generation of prefix and suffix automata that represent overlapping syntactical features of the entities on character and token level *Brauer et al. (2011)* and the automatic creation of REs based on genetic programming *Bartoli et al. (2012,0,0,0)*.

Constraining NN training to comply with known rules has also been an active research topic. *Hu et al. (2016)* proposes integration of constraints coming in the form of first order logic rules during training of NNs. *Alashkar et al. (2017)* trains an NN by minimizing a joint loss based on prediction of labels and adhering to the predefined rules. *Locascio et al. (2016)* proposed training LSTM NN to generate REs from sample pieces of text. *Luo et al. (2018)* incorporates knowledge of REs into training of NNs at three different levels: as the input features to NNs, as regularizations of the outputs of NN layers, or as a reward/penalty in the loss functions in NNs.

Unlike the aforementioned work, we do not attempt to learn explicit REs and do not force the outputs of NN layers match predetermined rules. Instead, we leverage REs as a means of generating a large quantity of weak labels from unlabeled data and using such data to pre-train an NN to recognize the provided REs. We fine tune such an NN on human-labeled data to exceed accuracy of the REs. A similar approach is effective in other domains. For example, *Felbo et al. (2017)* proposed pre-training an NN on millions of tweets labeled by emojis before fine tuning it for sentiment analysis. *Mahajan et al. (2018)* pre-trained an NN on billions of images labeled by hashtags before fine tuning it to

Table 3.1: Dataset Summary.

Task	Data Size	Example	Labor
Date Time	761,002	data-timestamp="Thu <b>Oct 05 2017 10:33:05 -0500</b> "> <script src='/js/next-stories.20170925144113.js'>	2 days
Course Number	44,651	<body><h1> <b>CS 556</b> Interactive Software Systems /home.html>Dan R. Olsen Jr.</a><li>Office: 3360 TM	1 week
Bill Date	49,002	(a) shall terminate on <b>30-09-2012</b> . ealth Service Act (42 U.S.C. 254c-15(c) (8)	-
Email Address	29,035	06:13:00 -0700 From: <b>phillip.allen@enron.com</b> To: Date:Mon, 23 Oct 2000 06:13:00 -0700 From: tom	1 week

various computer vision tasks.

### 3.3 Methodology

We describe the proposed framework in this section.

**Problem Definition:** Given a text string  $t$  and an entity type  $\mathcal{E}$ , the task is to predict whether  $t$  contains an entity mention of type  $\mathcal{E}$ . We treat this task as binary classification. To build the classifier, we assume that we are given a large corpus of unlabeled text strings  $T = \{t_1, t_2, \dots, t_n\}$ . The challenge is to train the classifier with the minimal human effort. We allow a human expert to help in two ways: (1) construct a new RE or find an RE created by others, and (2) label an unlabeled string. For the purposes of this paper, we assume that one or more REs suitable for entity identification are already available and that human effort refers only to string labeling. The available REs might have an arbitrary precision and recall. We will analyze the impact of the RE quality on classification accuracy in the experimental section.

**Solution Framework:** An overview of the proposed framework is illustrated in Figure 4.1.

STEP 1. All of the unlabeled strings from  $T$  are fed into an RE annotator. If any of the provided REs  $t_i$  recognizes  $t_i$ , it is weakly labeled as  $y_i = 1$ , otherwise it is weakly labeled

as  $y_i = 0$ .

STEP 2. An NN model  $M_{RE}$  is trained based on the weakly labeled data  $D_{RE} = \{(t_i, y_i) | i = 1, 2, \dots, n\}$ . Given a large number of sample strings, it is expected that we can train an NN with high accuracy on  $D_{RE}$ .

STEP 3. A subset of  $m$  strings from  $T$  are sampled randomly and a human annotator labels each of the sampled strings. String  $t_i$  is labeled as  $y_i = 1$  if the annotator recognizes an entity type  $\mathcal{E}$  in  $t_i$  and as  $y_i = 0$ , if not. We denote the resulting strongly labeled data set as  $D_{human} = \{(t_i, y_i) | i = 1, 2, \dots, m\}$ , where  $m \ll n$ .

STEP 4. The pre-trained NN  $M_{RE}$  is fine tuned with  $D_{human}$  data. We call the resulting NN  $M_{RE+human}$ . For comparison, we also train a randomly initialized NN directly on  $D_{human}$ . We call this NN  $M_{human}$ . The expectation is that the pre-trained NN captures very useful information about the entity type  $\mathcal{E}$  and that fine tuning is more effective than training a new NN from scratch.

**RE Annotator:** Let us denote the set of REs available for a specific entity type  $\mathcal{E}$  as  $R$ .  $R$  may be either created by human experts or generated automatically by tools like *Li et al. (2008)*; *Bartoli et al. (2018)*, which require a human-labeled subset of  $T$ . Both approaches are human-intensive.

**Deep Learning Architectures:** We do not have a preference over any deep learning architecture to train  $M_{RE}$ , as long as it can handle character-level inputs and produce binary outputs. We have no strict assumption about the strings in  $T$ . They may contain sentences from a formal news article, pieces of HTML code, or a mixture of formal texts and informal texts. For this reason, we treat  $t_i$  as a sequence of characters by default. NN architectures that meet our condition include but are not limited to: CNNs *Kim (2014)*, BiLSTMs *Lai et al. (2015)*, and BiLSTM with self-attentions *Lin et al. (2017)*. For this paper we implemented a BiLSTM architecture. However, we also tested a CNN architecture, reaching similar conclusions. Our architecture contains an embedding layer to project each charac-

ter into a vector, 2 BiLSTM layers to encode a sequence of character embeddings into a sequence of hidden vectors, and a max pooling layer followed by 2 fully-connected layers to project the hidden vectors into binary labels (Figure 4.1, on the right).

**Fine-tuning:** Once we have a model  $M_{RE}$  trained on weak labels, there are multiple ways to improve the weak model with human annotations to get  $M_{RE+human}$ . One common way is to freeze the parameters of all other layers of  $M_{RE}$  and fine-tune the last fully-connected layer *Donahue et al. (2014)*. *Felbo et al. (2017)* propose a 'chain-thaw' strategy, which freezes all layers, then sequentially unfreezes and fine-tunes a single layer at a time. We exploit a less costly strategy as proposed in *Erhan et al. (2010)*, which uses the weights learned in  $M_{RE}$  to initialize  $M_{RE+human}$ , and start training  $M_{RE+human}$  immediately with human annotations.

### 3.4 Experimental Setup

**Models in Comparison** We compare 5 models on the 4 data sets: (1) *Naive*, which always predicts 0, because 0 is the majority class on all 4 tasks. (2) *RE*, which uses the set of REs  $R$  designed by experts or tools to weakly label the strings. Although we had multiple REs for each task,  $R$  can be any subset of the available REs. (3)  $M_{RE}$ , which is the pretrained model of weak labels generated by  $R$  in model (2). (4)  $M_{human}$ , which is the model trained with human annotations. (5)  $M_{RE+human}$ , which is the fine tuned model  $M_{RE}$ .

**Evaluation Metrics** For each data set, we divide the 6,000 strings with human annotations into 5 folds. We leave one fold as our test data. The training data is selected from the other 4 folds. We report four scores for each model: Accuracy (ACC), F1, Precision, and Recall. We report the average results over three random repetitions in Section 3.5.

Table 3.2: The comparisons in the presence of very few human annotations:  $|D_{human}| = 20$ .

Model Name	Date Time (%)				Course Number (%)			
	ACC	F1	Precision	Recall	ACC	F1	Precision	Recall
<i>Naive</i>	77.34	0.00	0.00	0.00	68.47	0.00	0.00	0.00
<i>RE</i>	90.65	76.77	88.00	68.33	71.64	59.91	54.01	67.28
<i>M<sub>RE</sub></i>	91.45	79.09	<b>89.01</b>	71.39	72.64	61.52	55.21	69.47
<i>M<sub>human</sub></i>	74.78	40.75	37.25	46.38	62.81	30.74	37.63	28.45
<i>M<sub>RE+human</sub></i>	<b>93.50</b>	<b>85.44</b>	87.03	<b>84.95</b>	<b>82.86</b>	<b>74.31</b>	<b>72.45</b>	<b>77.01</b>
	Bill Date (%)				Email Address (%)			
<i>Naive</i>	92.83	0.00	0.00	0.00	88.75	0.00	0.00	0.00
<i>RE</i>	<b>94.56</b>	38.23	<b>97.92</b>	24.01	<b>98.72</b>	<b>94.61</b>	<b>89.79</b>	<b>100.00</b>
<i>M<sub>RE</sub></i>	94.53	38.10	96.30	23.96	98.64	94.28	89.19	<b>100.00</b>
<i>M<sub>human</sub></i>	92.56	2.15	6.25	1.30	83.44	42.30	35.99	53.03
<i>M<sub>RE+human</sub></i>	94.36	<b>39.95</b>	87.88	<b>27.59</b>	97.75	90.95	83.60	<b>100.00</b>

**Hyperparameters** We use 100 dimensions in the embedding layer. We set the activation function in the first fully connected layers as tanh. The batch size is set to 300. We also add dropout layers after the embedding layer, the max pooling layer, and the first fully-connected layer to avoid overfitting, with drop out rate at 0.5. Our implementation is in PyTorch. We tune the learning rate ( $\text{lr}$ ), the hidden units size ( $\text{nhidden}$ ) in BiLSTM layers and the output size ( $\text{nfc}$ ) of the first fully-connected layer by 5-fold cross validation using a random 6,000 sample from  $D_{RE}$ , for the sake of expediency. The ranges of selection are:  $\text{lr} \in [0.0002, 0.0005, 0.001, 0.002, 0.004, 0.008, 0.015]$ ,  $\text{nhidden} \in [50, 75, 100, 125, 150, 200]$  and  $\text{nfc} \in [20, 50, 100, 200, 500]$ . We use the *random search* algorithm proposed in *Bergstra and Bengio (2012)* that has been proved more effective than *grid search*. The hyperparameters used to train  $M_{human}$  and  $M_{RE+human}$  are identical to those used to train  $M_{RE}$ .

We train 2 epochs for  $M_{RE}$  on weakly labeled data.  $M_{human}$  and  $M_{RE+human}$  are trained for 50 epochs on strongly labeled data.

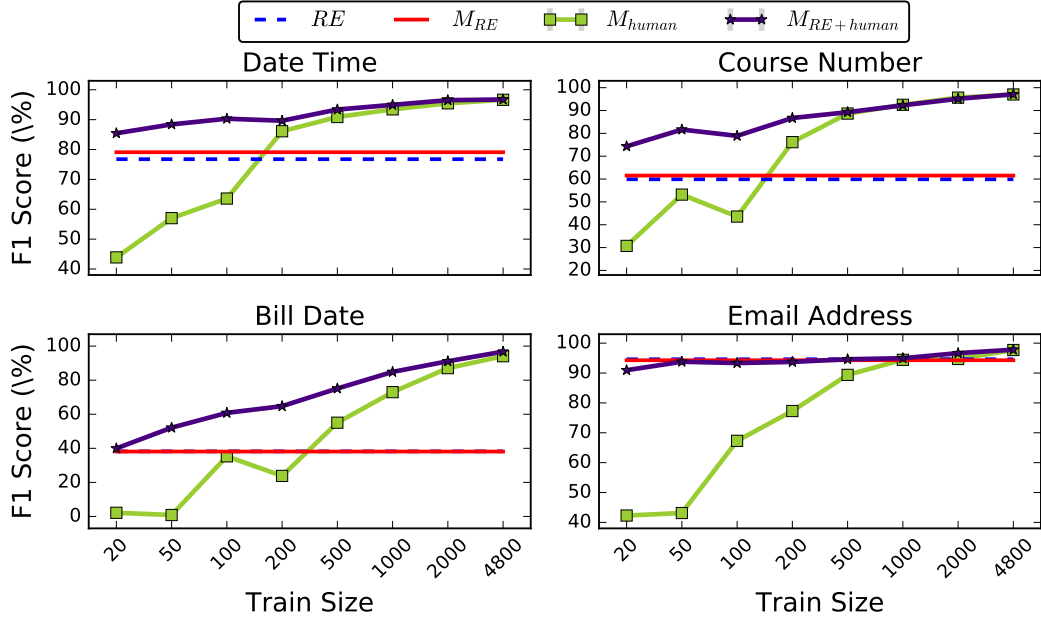


Figure 3.2: Trend of F1 when varying the number of human annotations.

### 3.5 Experimental Results

In this section, we evaluate the proposed framework with extensive experiments on the 4 entity recognition tasks. We use the empirical results to understand how the quality of initial REs impacts our conclusions.

#### 3.5.1 Entity Mention Detection with Limited Human Annotations

We report the comparisons of the 5 models in Table 4.3 when we only have 20 human annotations. We use all the available REs in each task in this experiment. Comparing the last two rows of each task,  $M_{RE+human}$  always outperforms  $M_{human}$  by a large margin according to all 4 evaluation metrics on all 4 data sets. The F1, Precision and Recall scores are more than twice larger for all of tasks. The pretraining strategy is quite effective despite the very limited human annotation.

In addition,  $M_{RE+human}$  is much better than  $RE$  in the datetime and Course Number tasks. Its Recall scores increase by 19.1% and 14.4%, respectively, in the two tasks. This means the human annotations greatly increase the coverage of the initial REs for entity

Table 3.3: Examples of misclassified strings by  $M_{RE}$  and  $RE$  for Date Time task. The first columns shows number of misclassified strings, number of positives, and number of negatives. The last column represent human label and classifications by  $RE$ , and  $M_{RE}$ . Datetime is in bold for positive human annotation.

		Date Time	
		Example	Labels
Group 1 19 / 11 / 8	"20170825" /><meta name="utime" content="20170828042824"/>	1 / 1 / 0	
	{"origin": "mw1273", "timestamp": "20171005170835", "ttl": 1900800}	1 / 1 / 0	
	<span class="timestamp-published">08/29/2017 12:13 pm ET</span>	1 / 1 / 0	
	2017/08/29/1/1700000000AEN20170829007751315F.html	0 / 0 / 1	
	11-hristo-dimitrov-2017-10-05-06-57-492-c37d5a31-4ade	0 / 0 / 1	
Group 2 48 / 36 / 12	0", "dateModified": "Thu, 05 Oct 2017 17:26:30 +0000"	1 / 0 / 1	
	<meta property="published", content="2017-08-28T14:4316-0400" />	1 / 0 / 1	
	<div>published <b>August 24, 2017 at 6:00 am</b> </time></div>	1 / 0 / 1	
	http://www.businesswire.com/news/home/20170829005822/en/</p>	0 / 1 / 0	
	resources/MWimages/MW-FV607-lava-2-MC-20171004095909.jpg">	0 / 1 / 0	

mentions. The Bill Date task is hard, since the initial REs already achieve Precision = 98% and entity mentions are really rare (ACC = 93% in *Naive* model). We still are able to improve the Recall by 15%, but at the expense of reduced Precision. This only gives a slight improvement in F1 (4.5%) and unchanged Accuracy. The Email Address task is even harder, with 90% Precision and 100% Recall from the initial REs. We fail to improve the accuracy with only 20 human annotations in this task.

### 3.5.2 Effect of the Number of Human Annotations

In Figure 3.2 we show how F1 varies with the size of the strongly labeled dataset,  $|D_{human}| = [20, 50, 100, 200, 500, 1000, 2000, 4800]$ . The observed trend is consistent over all 4 tasks: the larger the set of human annotations the better the performance of both  $M_{human}$  and  $M_{RE+human}$ . For the first 2 tasks,  $M_{RE+human}$  surpasses all other competitors at 20; it takes 50 human annotations in Bill Date task, and 2,000 in Email Address task.

Initial Set	$RE$				$M_{RE}$			
	ACC	F1	Precision	Recall	ACC	F1	Precision	Recall
RE1	75.88	4.83	24.89	2.68	75.71	4.20	23.68	2.32
RE2	79.97	20.72	100.00	11.58	80.00	20.92	100.00	11.70
RE3	79.78	19.34	100.00	10.75	79.83	19.75	100.00	10.99
RE4	83.40	41.86	100.00	26.71	83.73	43.77	100.00	28.19
All	90.65	76.77	88.00	68.33	91.45	79.09	89.01	71.39

Table 3.4: Comparison between  $RE$  and  $M_{RE}$  model for 5 different sets of REs.

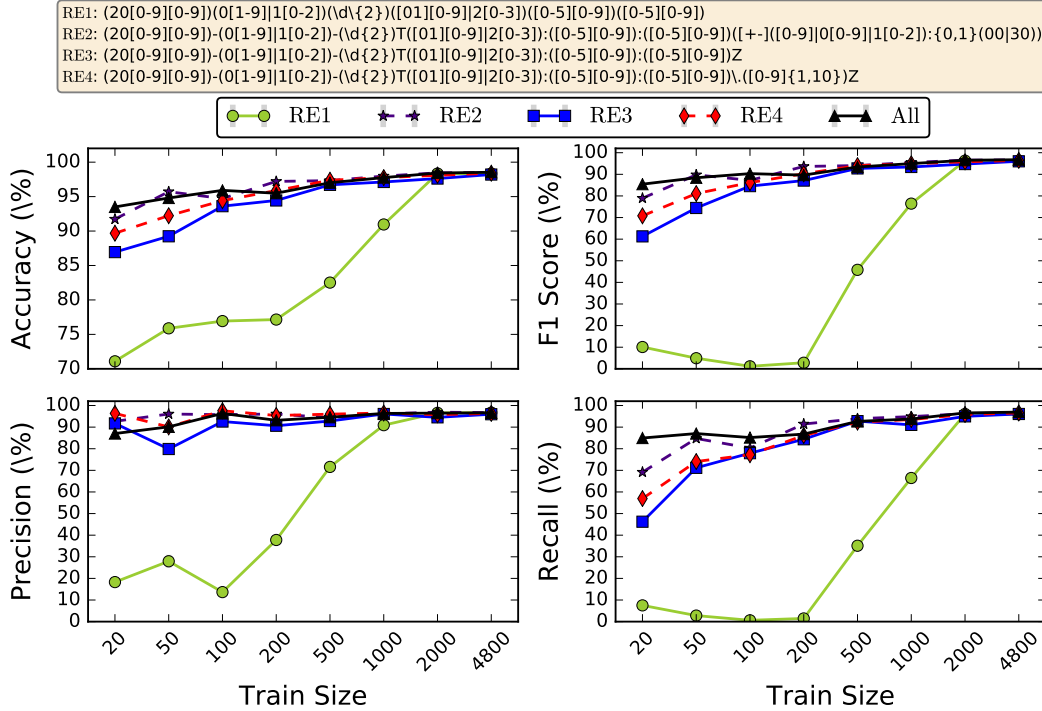


Figure 3.3: Performance of  $M_{RE+human}$  for different initial sets of REs and sizes of  $D_{human}$ .

### 3.5.3 Case Studies

From Table 4.3, we observe that  $M_{RE}$  achieves higher F1 scores than  $RE$  by about 2.3% on Date Time and 1.6% on Course Number tasks. This is a seemingly surprising outcome; NN trained on weak labels does better on human labeled test data than the REs used to generate the weak labels. To provide an insight, in Table 3.3 we illustrate example strings on which  $RE$  and  $M_{RE}$  disagree. *Group 1* consists of 19 strings where  $RE$  is correct and  $M_{RE}$  is not. *Group 2* consists of 48 strings where  $RE$  is incorrect and  $M_{RE}$  is correct. Looking at the examples in the first and last 2 rows of the table, we find strings

that match RE with `YYYYMMDDhhmms` format. This is an unusual form and it is expected that it might not always correspond to datetime entity. We hypothesize that the neural network encountered many negative strings in the weakly labeled data that have a similar form and learned that this form is not a reliable predictor of datetime.

Rows 6 - 8 in the table illustrate resilience of  $M_{RE}$  to small variations in the original REs. For example, there is *an extra space* in row 6, *a missing colon* in row 7, and *a mixture of spoken language* in row 8. We give cases where  $M_{RE}$  makes mistakes in rows 3 to 5, showing that the NN is not able to learn the underlying REs with 100% accuracy.

### 3.5.4 Impact of the Initial REs

In this subsection, we investigate the impact of the choice of REs  $R$  on the accuracy of NN models. Since *Naive* and  $M_{human}$  models are not affected by  $R$ , we compare only three models in this subsection:  $RE$ ,  $M_{RE}$  and  $M_{RE+human}$ . We select 4 out of the 25 REs in the datetime task for this study. The 4 REs are of different quality. We list the 4 REs in Figure 3.3. An example pattern that RE1 matches is `20180503101212`, for RE2 it is `2018-05-03 10:12:12`, for RE3 it is `2018-05-03T10:12:12Z` in UTC time zone and for RE4 it is `2018-05-03 10:12:12+00:00` or `2018-05-03 10:12:12-00:00`. We also consider the quality of NNs trained on weak labels from the whole set of 25 REs, denoted as All.

In Table 3.4 we compare the performance of  $RE$  and  $M_{RE}$  for the 5 different selections of REs for weak labeling. It can be observed that RE1 is the weakest individual RE in the group with  $F1 = 4.83$ , while RE4 is the strongest with  $F1 = 41.86$ . Using all 25 REs gives the highest accuracy of  $F1 = 76.77$ . We can see that  $M_{RE}$  closely follows the performance of  $RE$  and it is interesting to observe that  $M_{RE}$  becomes visibly superior only with good REs.

In Figure 3.3, we plot the 4 accuracy metrics for model  $M_{RE+human}$ , which was pretrained on weakly labeled data generated by 5 different choices of REs, with varying

initial RE sets and sizes of human annotations. We observe a significant influence of REs on accuracy. We can also observe that as the number of strong labels grows, the impact of RE choice decreases. When the number of strong labels exceeds 1,000, the impact of the RE choice becomes negligible.

There seems to be a trade-off between creating more REs and creating more strong labels : (1) If designing RE takes a lot of time, a good strategy might be to take time to construct one moderately good RE and spend time on data labeling. (2) If the pattern seems to be easy to describe by an RE, it might be a good strategy to spend time on creating a better set of REs and spend less time in labeling.

### **3.6 Conclusions**

The main premise of this work is that it is practically impossible to create REs capable of identifying entities with perfect precision and recall at web scale. This paper explores ways to combine the expressive power of REs, ability of deep learning, and human-in-the-loop into a novel integrated framework for entity recognition in web data. The framework starts by creating or collecting the existing REs for a particular type of an entity type (e.g., emails). Those REs are then used over a large document corpus to collect weak labels for the entity mentions and an NN is trained to predict those RE-generated weak labels. Finally, a human expert is asked to label a small set of documents and the neural network is fine tuned on those documents. The experimental evaluation on several entity identification problems shows that the proposed framework achieves impressive accuracy, while requiring very modest human effort.

Web sources often change in ways that prevent the induced REs from extracting data correctly. At the web scale, we require automated tools to maintain them. One direction of future work is to use our framework to diagnose when a RE is broken over a text stream.

## CHAPTER 4

# HOW TO INVEST MY TIME: LESSONS FROM HUMAN-IN-THE-LOOP ENTITY EXTRACTION

### 4.1 Introduction

Entity extraction occupies a prominent place in information retrieval. Named entity recognition, the most recognized entity extraction subtask, seeks to automatically identify substrings that represent specific people, locations, events, or organizations. Beside named entities, there is a large class of entities that are not “named,” such as expressions of *dates, times, email addresses, phone numbers, currencies, credit card numbers, social security numbers, measurements, and object properties*. These types of entities can often be expressed or approximated by a regular expression (regex) and are the focus of this work. They have drawn interest from several communities, including NLP *Li et al. (2008); Murthy et al. (2012); Cochran et al. (2015)*, databases *Li et al. (2008)*, data mining *Bartoli et al. (2012,0,0,0)*, and life sciences *Murtaugh et al. (2015); Zeng et al. (2006)*.

Two common approaches to recognize regex-like entities are to (1) manually create a regex and (2) train a machine learning model, both of which have their advantages and disadvantages. Most programmers are familiar with regex and can write reasonably accurate entity recognizers with relatively little effort, without the need to use machine learning software. However, once a regex goes beyond a level of complexity, writing it requires a lot of time and experience and results in brittle recognizers, leading to a saying "Now you have two problems" *IQAndreas (2014)*. Even a seemingly simple task of recognizing an email address apparently requires 6,500 characters *Millner (2008)*!

Machine learning (ML) approaches attempt to either infer a regex or create a regex-oblivious model. Regex learning approaches *Murthy et al. (2012)*; *Simoes et al. (2018)*; *Li et al. (2008)*; *Bartoli et al. (2016,0)* require a set of substrings and focus on constructing a short regex recognizing the substrings. Similarly to manual construction of a regex, the existing approaches quickly end up in very long and brittle formulas and are not commonly used in practice *Bartoli et al. (2016)*. In regex-oblivious approaches, the objective is to train a model such as a neural network (NN) from labeled substrings *Zhang et al. (2018a)*. An advantage is that labeling does not need programming expertise. A disadvantage is that this approach requires a large set of labeled examples. In the rest of the paper, when we refer to ML methods we refer to the regex-oblivious approach.

In our recent work we proposed a human-in-the-loop (HIL) framework *Zhang et al. (2018a)* that uses human effort to both write a regex and to manually label the documents. As will be elaborated in the methodology, a regex is used to scan a document corpus and produce weak labels to pretrain an NN. Then, manually labeled substrings are used to fine-tune the network. The results showed that fine tuning a pretrained NN is superior to training it from scratch. Thus, the results indicate that writing a regex before manual labeling is highly desirable. However, this conclusion does not take into account the time needed to create a regex and regex writing expertise. In this study, we consider the problem from a practical perspective, where a human is given a fixed amount of time to interact with the ML system for entity recognition.

Time and expertise are critical factors in a HIL ML system such as the one we consider. Let us look at a potential real-life scenario. Let us imagine a data scientist Amy, faced with a challenge of extracting publication dates from a corpus of hundreds of thousands of news articles crawled from the Web given a one hour deadline. Just finding a single mention of date would include a lengthy scanning of articles and would make the task infeasible. Alternatively, Amy may remember that all articles are published in 2019 and write simple regex 2019 to identify date mentions with a high recall but low precision. Then, she glances

over the extracted substrings surrounding mentions of 2019 and does one of the two things: (1) starts labeling the dates or (2) realizes that all dates seem to follow a particular pattern and proceeds to write a regex. Even if Amy starts writing a regex and proceeds with labeling, another question is whether she should spend a lot of time trying to improve the regex or stop and start with manual labeling. We are not aware of published results that may inform Amy how to efficiently invest her time. We set to gain insight into this problem in this work.

We make the following contributions in this paper:

- We propose a framework that recognizes an entity through character-wise classification. This is in contrast to our previous work *Zhang et al. (2018a)* where we developed a HIL framework that detects if a text passage contains an entity through sequence-wise classification.
- We propose an algorithm for active selection of substrings.
- We perform a thorough characterization of the proposed framework on 5 entity recognition tasks.
- We perform a small-scale user study to obtain insight into the trade-offs between spending time to write regular expressions and spending time to manually label text fragments.

## 4.2 Related Work

This section briefly reviews several lines of research we deem to be the most relevant to our work.

**Regex Refinement and Inference.** This line of research aims to (partially) automate regex construction. One research direction focuses on improving the precision and recall of initial regexes by identifying true or false matches *Li et al. (2008)*; *Murthy et al. (2012)*;

*Simoes et al. (2018)*. They start from a user defined regex with either high recall, but low precision, *Li et al. (2008)* or high precision, but low recall *Murthy et al. (2012)*, and search for an improved regex. In either case users need to create true positive and negative instances in the matching set of the initial regex. Some works seek to reduce human labeling efforts in this process, e.g., new matches of candidate regexes are automatically grouped into positives and negatives by comparing their context similarities to those of the generalized regex *Simoes et al. (2018)*. A different line of work attempts to induce regexes from positive and negative sample strings *Brauer et al. (2011)*; *Fernau (2009)*; *Denis (2001)*. They do not require an input regex. The most recent approach uses genetic programming *Bartoli et al. (2016,0)*. All these efforts require human input, such as a set of examples, or an initial high precision regex, or manually labeled negative and positive matches of regexes. Human effort had not been explicitly quantified in this line of work.

**Human Annotation Effort:** An important area of research in ML seeks to reduce the human annotation effort, both in scale (i.e., amount of labeled instances) and form (e.g., weak labels). This is a broad area of research and we limit our coverage to the entity mining literature. One line of work uses solely weak labels to train NER models. Distant-LSTM-CRF *Giannakopoulos et al. (2017)*, AutoNER *Shang et al. (2018)*, and SwellShark *Fries et al. (2017)*; *Ratner et al. (2016)* are examples of approaches in this category. String matching and (expert) rules are common means to generate weak labels.


Active learning aims to smartly involve human judgment in the training of a model. In NER, this follows a 2-iteration process. In the first iteration, the system samples sentences according to some heuristics, asks users to annotate them, and trains an initial NER model. In the second iteration, the system iteratively recruits unlabeled sentences by a scoring function for human annotations. The work in *Chen et al. (2015)* uses the longest sentence selection heuristic in the first iteration and 12 scoring functions in the second iteration. In *Shen et al. (2017)*, a bag of initial models is trained with a hand-labeled seed data set. They use the disagreement among the bag of models measured by KL-divergence and f-

complement for scoring.

We are not aware of any work in this space that couples weak supervision with active learning, as we propose in this work. In addition, although weak labeling is cheaper than standard labeling it still incurs human cost. This is largely ignored in the literature. We consider these factors in our framework.

### 4.3 Proposed Framework

**Problem definition** Given a corpus of documents  $\mathcal{D}$  and an entity type  $\mathcal{E}$ , the objective is to create an accurate entity extractor for  $\mathcal{E}$  while limiting the total human effort within  $T$  minutes.

We give an overview of our solution in Figure 4.1 (left). The input of our framework is a document corpus  $\mathcal{D}$  and an entity type  $\mathcal{E}$  (e.g. phone number). It outputs an entity extractor. The proposed framework has several modules. The first module selects candidate substrings that are likely to contain an entity mention. Given the set of candidate substrings, the second module is responsible for weak labeling them. This is accomplished with a regex. The third module trains an NN for entity extraction using a set of labeled substrings. The fourth module selects a subset of substrings for human labeling. After the selected substrings are manually labeled by a human, they are fed back to module 3 for fine-tuning of the NN. We highlight the places where human effort is needed with a clock . A human expert invests time in three ways in the system. She (1) creates a regex for module 1 that selects candidate substrings with high recall and allowing for low precision, (2) creates a regex for module 2 that weakly labels the candidate substrings with relatively high precision and recall, and (3) manually labels an unlabeled substring in module 4 by highlighting substrings corresponding to an entity mention. We provide technical details in the remainder of this section.

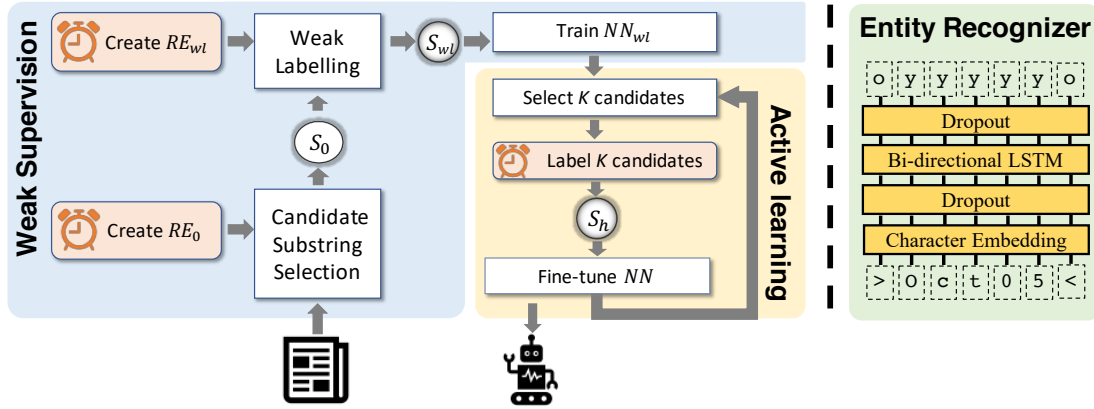


Figure 4.1: Overview of the solution framework(left) and deep learning architecture (right) used to train entity recognizer.

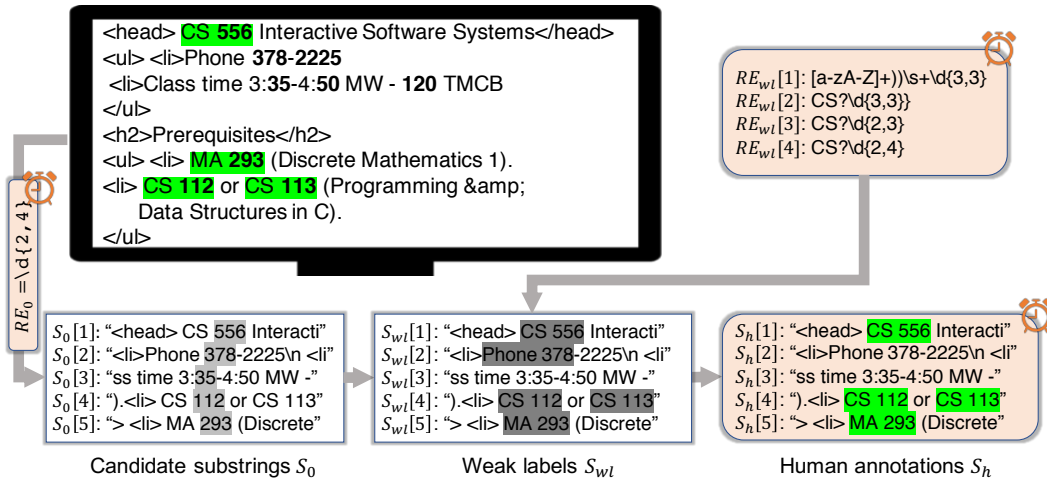


Figure 4.2: An example of course number recognition on a sample HTML document. Only 5 of the 9  $S_0$  candidate substrings are shown

### 4.3.1 Selection of Candidate Substrings

The core objective of our framework is to minimize human effort needed to build an entity extractor. As will be specified later in the paper, the user is expected to scan the corpus and recognize entity mentions. In a typical entity extraction scenario, the number of entity mentions in a corpus is relatively small compared to the total size of the corpus. Thus, it is very helpful to users if they automatically exclude portions of the corpus that do not contain entity mentions. Our main observation is that it is often possible to specify a simple regex that recognizes entities with a high recall (i.e., including most of the entities) and

potentially a low precision (i.e., allowing a high fraction of false positives). For example, if our objective is to extract course numbers, a regex that recognizes two or more digits (e.g.,  $\backslash d\{2, 4\}$ ) is likely to capture most of the course mentions in the U.S universities. It is evident that such regex also recognizes many strings that are not related to course numbers, thus resulting in low precision. We denote a regex used for recognition of candidate entities as  $RE_0$  and call it the *candidate regex*.

For each match of the candidate regex in the text, we generate the *candidate substring* by expanding it with  $\frac{L}{2}$  characters before and after it.  $L$  is selected to guarantee that the string fragment includes the entity mention and has sufficient context surrounding the mention. This helps determine if a substring contains the entity. We denote the set of candidate substrings obtained in this way from corpus  $\mathcal{D}$  by  $S_0 = \{s_1, s_2, \dots, s_n\}$ , where  $n$  is the number of strings  $RE_0$  matches.

We need to emphasize that, in our framework, the role of  $RE_0$  is to remove portions of the corpus that clearly do not contain entity mentions. It is not critical that the precision of the candidate substrings is high. Instead, the emphasis is on encouraging the user to quickly come up with a simple  $RE_0$  that has a good recall.

### 4.3.2 Weak Labeling

Given the set of candidate substrings  $S_0$ , our objective is to label and use them to train an NN model for entity extraction. The NN has to predict each character in a candidate substring as either positive (belonging to an entity) or negative (not belonging to an entity). Given those character-wise predictions, the entity is identified as a string of consecutive positively labeled characters. Thus, it is possible that we may extract multiple entities from a single candidate substring.

A straightforward approach for labeling  $S_0$  strings is to ask a user to manually label some or all of its substrings. As will be shown in the experimental results, this approach is inefficient when  $S_0$  is very large. Instead, we propose a *weak labeling* approach aided by

regex. In particular, we allow the user to provide a regex with a moderately high precision on  $S_0$ . If an entity is well studied (e.g., date, email address), it is possible that one may find a good regex quickly by searching the web. Otherwise, it is assumed that the user is experienced enough with regexes and able to come up with a good regex in a reasonable amount of time. To aid the user, we can load a random subset of  $S_0$  substrings into a freely available regex testing and debugging tool such as <https://regex101.com> *Li et al.* (2008). We denote the resulting regex by  $RE_{wl}$ , where the subscript  $wl$  is an abbreviation for weak labeling.

Given  $RE_{wl}$ , we can automatically weakly label all the substrings in  $S_0$ .  $S_{wl}$  denotes the resulting weakly labeled data set. We use  $S_{wl}$  to train an NN  $NN_{wl}$ . We expect that the recall and precision of  $NN_{wl}$  is comparable to  $RE_{wl}$ . The benefit of training  $NN_{wl}$  compared to directly using  $RE_{wl}$ , which is the traditional approach for entity extraction, is that  $NN_{wl}$  can be further fine-tuned and improved once manually labeled substrings become available.

### 4.3.3 Entity Recognizer

Given the set of labeled candidate substrings in  $S_{wl}$ , the next objective is to train an NN that classifies each character within a candidate substring. Suitable NN architectures include but are not limited to BiLSTM + CNNs *Chiu and Nichols* (2015), BiLSTM + CRF *Lample et al.* (2016), and BiLSTM + CNNs + CRF *Ma and Hovy* (2016). In this study we use a BiLSTM + CNNs architecture illustrated in Figure 4.1 (on the right).

For a candidate substring with  $L$  characters, an embedding layer is used to map the  $l^{\text{th}}$  character ( $l = 1, 2, \dots, L$ ) into a real valued vector  $e^l$ , where  $e^l \in \mathbb{R}^p$  and  $p$  is the size of character embedding. Two or more Bidirectional LSTM (BiLSTM) layers are used to generate hidden vectors at each position  $l$ . One layer of BiLSTM contains two stacks of regular LSTM cells. The forward LSTM cells process the input string from the beginning to the end, while the backward LSTM cells go from the end to the beginning.

The loss function for the  $i^{\text{th}}$  string at the  $l^{\text{th}}$  position is defined as the cross entropy function. To train the deep learning model, we average the losses from all characters in the training set. We add a dropout layer after the embedding layer and each of the BiLSTM layers to avoid overfitting. To predict labels in a string during testing we assign each character to the class with the larger probability. We denote the vector of prediction of  $s_i$  as  $\hat{y}_i$ .

#### 4.3.4 Fine-Tuning with Human Labels

The benefit of training an NN on weakly labeled data is that it can be fine-tuned and improved using manually labeled data. Assume we already trained  $NN_{wl}$ . The next questions are how many candidate substrings to manually label and how to select them from  $S_0$ . In our framework, we ask the user to label  $K$  candidate substrings and then fine-tune the NN. We repeat the manual labeling and fine-tuning process until reaching the desired accuracy or the time limit.

The baseline approach to selection of substrings to be labeled is to select  $K$  substrings at random from  $S_0$ . We refer to this selection algorithm as the **Random Querying (RQ)**. A more sophisticated approach is to use active learning, which attempts to select  $K$  substrings that result in the fastest increase in accuracy. Among the many active learning algorithms proposed in the ML literature *Krishnakumar (2007)*, the ones based on the uncertainty principle have been the most successful over a large range of application. In particular, the examples on which the current predictor is more uncertain are more likely to be selected. If a sigmoid neuron is used as output of an NN, we can interpret its output for the  $l^{\text{th}}$  character of string  $s$  as class probability,  $p(y_l|s_l)$ . The uncertainty of the prediction of the  $l^{\text{th}}$  character is defined as entropy  $E(s_l) = -\sum_{k=0,1} p(y_l = k|s_l) \log p(y_l = k|s_l)$ . Higher entropy indicates high uncertainty.

To select the most uncertain subsequences, we need to aggregate the entropy over each subsequence. We denote uncertainty of a subsequence as  $E(s)$ . We consider several op-

tions for aggregation, e.g., averaged entropy, maximum entropy, and maximum entropy over a window (it is reported superior in *Chen et al. (2015)*). While the differences are not large, we experimentally observed that maximum entropy is slightly better than the alternatives. We refer to the selection algorithm that picks the  $K$  most uncertain subsequences as the **Max Entropy (ME)**. When using the ME, it is possible that the most uncertain  $K$  substrings are highly redundant. Inspired by the idea of pre-clustering before active learning *Nguyen and Smeulders (2004)*, we consider an alternative that first selects  $M > K$  substrings from  $S_0$  at random and then uses ME to select the most uncertain  $K$  substrings. We refer to this selection algorithm as the **Random then Max Entropy (RME)**.

Table 4.1: Summary of documents in the 5 entity recognition tasks.

Domain	$ \mathcal{D} $	Doc avg length (chars)	Number of entities in $\mathcal{D}$	Annot time (hrs)	Rate (ms/ch)	Entity avg length (chars)
<b>DATE</b>	6,000	137,379	1,399 *	22	127	21.63
<b>BILLDATE</b>	600	27,518	3,085	-	100**	12.63
<b>EMAILADDRESS</b>	602	1,284	2,206	16	74.5	21.92
<b>COURSENUMBER</b>	600	4,586	4,588	60	78.5	6.46
<b>PHONENUMBER</b>	3149	2674	2,018	150	65.9	13.64

\* The number of entities in 6,000 strings in  $S_0$ , one string per document.

\*\* A reasonable guess of the human annotation rate.

### 4.3.5 Summary of the Framework

In Figure 4.2, we take course number as a running example and illustrate the intermediate data generated along the steps of our framework. We give the details in the following subsections. One notices that the user is involved in 3 steps of the algorithm: (1) creating candidate regex  $RE_0$ , (2) creating weak labeling regex  $RE_{wl}$ , and (3) labeling candidate substrings. The total human effort is a sum of the efforts on those 3 steps. The effort for creation of  $RE_0$  is assumed to be significantly smaller than for the other 2 steps. If we are given the time budget for human effort and assuming that the time to create  $RE_0$  is negligible, an open question is how should the time be split between steps (2) and (3). We design our experiments to gain insight into the trade-offs between spending time to create

a good regex and to manually label the candidate substrings.

The proposed framework also allows skipping one or more of the 3 steps. For example, instead of step (1), we can create the candidate substrings from all or from randomly selected substrings of length  $L$ . We can also decide to skip step (2) and train the first NN on the first  $K$  manually labeled candidate substrings selected from  $S_0$  at random. We can refer to such an approach as the *cold start*. Finally, we can decide to skip step (3). In this case, we can decide to directly use  $RE_{wl}$  for entity extraction. We will evaluate all those scenarios in the experimental studies.

## 4.4 Experimental Design

In this section we describe the entity extraction tasks we created to evaluate our framework and perform user studies.

### 4.4.1 Entity Recognition Tasks

We consider 5 entity recognition tasks in our experimental study:

- **DATE** recognition: we downloaded HTMLs of 6,000 English news articles published from August 24 - 30, 2017. They are randomly selected from 0.6 million articles in the Kaggle dataset<sup>1</sup>. The task is to extract datetime in 2017 from the source HTML.
- **BILLDATE** recognition: we downloaded 600 OCR scanned U.S. Congress bills<sup>2</sup>. The task is to extract dates from the bills.
- **EMAILADDRESS** recognition: the task is to extract email addresses from 602 emails in the publicly available Enron email data set<sup>3</sup>.
- **COURSENUMBER** recognition: we downloaded 600 HTMLs from the 4 Universities Data Set at CMU (Web->KB) project<sup>4</sup>. The task is to extract course numbers from faculty and

---

<sup>1</sup><https://www.kaggle.com/therohk/global-news-week>

<sup>2</sup><http://machinelearning.inginf.units.it/data-and-tools>

<sup>3</sup><https://www.cs.cmu.edu/~./enron/>

<sup>4</sup><http://www.cs.cmu.edu/afs/cs/project/theo-20/www/data/>

department web pages.

- **PHONENUMBER** recognition: we downloaded 3,149 documents, 2,000 of them from the 20 newsgroup dataset<sup>5</sup> and the remaining 1,149 from the 4 universities data set. The task is to extract phone numbers from newsgroup messages and personal web pages.

We gathered various types of documents, ranging from HTML pages to OCR scanned documents. In Table 4.1, we list the basic statistics about documents in each of the 5 recognition tasks. The average length of a document varies a lot across the tasks, ranging from 1,284 in **EMAILADDRESS** to 137k in **DATETIME**.

#### 4.4.2 Labeled Data for Evaluation

In order to allow comprehensive evaluation, we used student volunteers to manually label all documents in **EMAIL**, **COURSENUMBER** and **PHONENUMBER** corpuses. For documents in **BILLDATE** task, we knew the ground truth based on *Bartoli et al.* (2016), so we did not use volunteers. For **DATETIME** task, we deemed too time consuming to label all the 6,000 documents; instead, we manually labeled one randomly selected substring from each document that contained 2017 in the center.

We list the number of entities in each corpus in Table 4.1. We also list the total time our volunteers took to annotate the corpus in each task and report the labeling rate as (milliseconds /character), which is calculated as the total time divided by the number of characters in the corpus. We assert that the labeling rate for **BILLDATE** task, is similar to that of **DATETIME** task. We observe that **COURSENUMBER**, **EMAILADDRESS**, **PHONENUMBER** are easier to label than **DATETIME**, and that the average entity length ranges from 6 to 20.

#### 4.4.3 Accuracy Measures

We conduct all of our experiments using document-level 5-fold cross validation. We first divide documents into 5 subsets at random. We train the models on candidate sub-

---

<sup>5</sup>[www.cs.cmu.edu/afs/cs.cmu.edu/project/theo-20/www/data/news20.html](http://www.cs.cmu.edu/afs/cs.cmu.edu/project/theo-20/www/data/news20.html)

strings generated from documents in 4 out of the 5 subsets and test them on the candidate strings in the remaining subset. The reported accuracies are averaged over the 5 repetitions within the cross-validation. We report position-level and entity-level accuracy.

Suppose there are  $N$  characters in the test set and denote the true labels for the  $i$ -th character as  $y_i$  and its prediction as  $\hat{y}_i$ . We define the positional precision (PosPrec), the positional recall (PosRecall) and the positional F1 (PosF1) as:

$$\begin{aligned}
 \text{PosPrec} &= \sum_{i=1}^N \mathbb{1}(y_i == 1 \wedge \hat{y}_i == 1) / \sum_{i=1}^N \mathbb{1}(\hat{y}_i == 1) \\
 \text{PosRecall} &= \sum_{i=1}^N \mathbb{1}(y_i == 1 \wedge \hat{y}_i == 1) / \sum_{i=1}^N \mathbb{1}(y_i == 1) \\
 \text{PosF1} &= 2 \cdot \text{PosPrec} \cdot \text{PosRecall} / (\text{PosPrec} + \text{PosRecall})
 \end{aligned} \tag{4.4-1}$$

Entity level accuracies are calculated considering entities in the test strings. Suppose there are  $P$  ground truth entities in the test strings, denoted as  $E_{true}$ . After we get positional predictions for test strings, we extract the predicted entities as substrings of consecutively predicted positive characters. Assuming there are  $Q$  predicted entities in  $E_{pred}$ , we can calculate the size of their intersection  $|E_{true} \cap E_{pred}|$ . We define the entity precision (EntPrec), the entity recall (EntRecall) and the entity F1 (EntF1) as

$$\begin{aligned}
 \text{EntPrec} &= |E_{true} \cap E_{pred}| / |E_{pred}| \\
 \text{EntRecall} &= |E_{true} \cap E_{pred}| / |E_{true}| \\
 \text{EntF1} &= 2 \cdot \text{EntPrec} \cdot \text{EntRecall} / (\text{EntPrec} + \text{EntRecall})
 \end{aligned} \tag{4.4-2}$$

## 4.5 Framework Characterization

In this section, we study the proposed framework and its components, without focusing on user time.

Table 4.2: Summary of the  $RE_0$ .

Domain	$RE_0$	Prec	Recall	$ S_0 $	%Cov
<b>DATETIME</b>	2017	0.194*	1.0*	761,002	9.23
<b>BILLDATE</b>	$\backslash d\{2, 4\}$	0.105	0.980	72,258	43.8
<b>EMAILADDRESS</b>	@	0.392	1.0	5,488	71.0
<b>COURSENUMBER</b>	$\backslash d\{2, 4\}$	0.112	0.969	43,623	79.2
<b>PHONENUMBER</b>	$\backslash d\{3, 4\}$	0.198	0.990	25,087	29.8

\* Assumes all datetimes have "2017".

#### 4.5.1 $RE_0$ for Candidate Substring Extraction

The first step in the framework is creating the candidate regex  $RE_0$  with high recall. We instructed one of the coauthors to come up with  $RE_0$  for each of the 5 tasks in less than 5 minutes per task. Table 4.2 lists the resulting candidate regexes. For the **DATETIME** task, since the 6,000 documents are from August 24 - 30, 2017, the regex assumes that string 2017 occurs in all datetime entities listing year 2017.

Using the labels we collected for all the tasks (Table 4.1), we are able to evaluate the precision and recall of  $RE_0$  defined as  $\text{Prec} = \frac{\# \text{True entities hit by } RE_0}{\# \text{Total hit of } RE_0 \text{ in } \mathcal{D}}$ ,  $\text{Recall} = \frac{\# \text{True entities hit by } RE_0}{\# \text{Total true entities in } \mathcal{D}}$ , respectively. As can be seen in Table 4.2, the recall is very high on all 5 tasks, while the precision is quite low, as expected.

Each substring recognized by  $RE_0$  becomes an anchor for a candidate substring. Each candidate substring is formed by taking  $\frac{L}{2}$  characters preceding the start of the match and  $\frac{L}{2}$  characters succeeding it. We set  $L = 100$  in all experiments. As seen from Table 4.2, for **BILLDATE**, the total length of  $S_0$  ( $= 72,258 * 100$ ) is 43% of the original corpus ( $= 6,000 * 27,518$ ) while it still contains 98% of all the true entity mentions. Thus,  $|R_0|$  more than doubles efficiency of manual labeling.

#### 4.5.2 $RE_{wl}$ for Weak Labeling

For experiments in this section, we assume that  $RE_{wl}$  is already available and ask if pretraining an NN on weak labels obtained from  $RE_{wl}$  is beneficial. We collect regexes to

instantiate  $RE_{wl}$  from the web and published papers. For **BILLDATE**, the regex is from *Murthy et al.* (2012). We use the top five regexes from the Rege Library<sup>6</sup> website for **EMAILADDRESS**. For **PHONENUMBER**, seven out of eight regexes are from *Murthy et al.* (2012), and the remaining one is from *Li et al.* (2008). We use four regexes, one of which is borrowed from the results learned by ReLIE *Li et al.* (2008), and the remaining three are from *Murthy et al.* (2012) for **COURSENUMBER**. We use a disjunction of the collected regexes in each task as  $RE_{wl}$ . Those  $RE_{wl}$  are used on the candidate substrings to generate weak labels.

### 4.5.3 Hyperparameter Tuning and Settings

Deep learning is very sensitive to hyperparameters. A common approach to tune the hyperparameters is to explore several combinations of hyperparameters on validation data. However, since our framework relies on an iterative process that repeatedly fine-tunes an NN on an increasingly large set of labeled substrings, this standard approach is not feasible. Instead, we tune the hyperparameters on a subset of the weakly labeled data. We use the *random search* algorithm proposed in *Bergstra and Bengio* (2012) that was shown to be more effective than the *grid search*. The best hyperparameters obtained in this way are used in all the experiments.

We set the activation function in the first fully connected layer to tanh and the batch to 512. We also add dropout layers after the embedding layer, the max pooling layer, and the first fully-connected layer to avoid overfitting, with the dropout rate set at 0.5. Our implementation is in PyTorch. With pre-defined  $RE_{wl}$ , we can tune the learning rate (`lr`), the dimension in the character embedding layer (`emsize`), the hidden unit size (`nhidden`) in BiLSTM layers and the number of BiLSTM layers (`nlayers`) by 5-fold cross validation using a random sample of weakly labeled data. We explore the following ranges for the hyperparameters:  $lr = 2^{-k}$ ,  $k \in [6, 7, \dots, 12]$ , `emsize`  $\in [20, 30, 40, 50, 60, 70, 80]$

---

<sup>6</sup><http://www.regexlib.com/>

$n_{\text{hidden}} \in [50, 75, 100, 125, 150, 200]$  and  $n_{\text{layers}} \in [1, 2, 5]$ . We select a set of the best hyperparameters for each task.

We use 5 epochs to train an NN on weakly labeled data  $S_{wl}$ . For each round of fine-tuning, we use all previously collected manually labeled substrings  $S_h$  and train for 10 epochs over  $S_h$ . For selection of candidate substrings for labeling, we select them from a pool of 10,000 randomly selected candidate strings. We set the number of strings to be labeled in each iteration to  $K = 20$ . For **RME** selection algorithm, we set  $M$  to 500.

#### 4.5.4 Impact of Weak Supervision

We evaluate 5 different approaches: (1) **Random**, which randomly predicts 0 or 1 for any character. (2)  $RE_{wl}$ , which uses regexes from Section 4.5.2 to recognize entity mentions. (3)  $NN_{RE_{wl}}$ , which is the NN pretrained on weak labels generated by  $RE_{wl}$ . (4) **RQ w/o (100)**, which is the NN trained directly with 100 randomly selected manually labeled candidate substrings. (5) **RQ w (100)**, which is an NN pretrained on weak labels and fine-tuned with 100 randomly selected labeled candidate substrings.

Table 4.3: Impact of weak supervision and active learning on the 5 tasks.

Model Name	Human Annots	DATETIME		BILLDATE		EMAILADDRESS		COURSENUMBER		PHONENUMBER	
		EntF1	PosF1	EntF1	PosF1	EntF1	PosF1	EntF1	PosF1	EntF1	PosF1
<b>Random</b>	0	0	0.1004	0	0.0331	0.0002	0.2543	0.0002	0.0446	0	0.1335
$RE_{wl}$	0	0.4340	0.7104	0.2827	0.3519	0.8811	0.9695	0.3926	0.4299	0.3182	0.5196
$NN_{RE_{wl}}$	0	0.4411	0.7103	0.2825	0.3529	0.8819	0.9733	0.4078	0.4354	0.3138	0.5179
<b>RQ w/o (100)</b>	100	0.0449	0.8218	0.2853	0.6164	0.6914	0.9716	0.5311	0.6838	0.1418	0.5921
<b>RQ w (100)</b>	100	<b>0.5061</b>	<b>0.8675</b>	<b>0.508</b>	<b>0.6196</b>	<b>0.962</b>	<b>0.9907</b>	<b>0.6876</b>	<b>0.7899</b>	<b>0.6007</b>	<b>0.7787</b>
<b>RQ w/o (1000)</b>	1000	0.8376	0.9343	0.8226	0.9169	0.9903	0.9973	0.8408	0.9129	0.7972	0.8903
<b>RQ w (1000)</b>	1000	0.8644	0.9420	0.8452	0.9177	0.9898	0.9973	0.8451	0.9175	0.8253	0.8964
<b>ME (1000)</b>	1000	0.8696	0.9537	0.9534	<b>0.9838</b>	0.9940	0.9959	<b>0.9234</b>	<b>0.9514</b>	0.8943	0.9359
<b>RME (1000)</b>	1000	<b>0.8879</b>	<b>0.9565</b>	<b>0.9537</b>	0.9829	<b>0.9951</b>	<b>0.9983</b>	0.8765	0.9373	<b>0.8964</b>	<b>0.9401</b>

In the top half of Table 4.3, we see that weak supervision helps in two aspects. First, if we compare  $NN_{RE_{wl}}$  and  $RE_{wl}$ , we notice that the weakly supervised model  $NN_{RE_{wl}}$  preserves the precision and recall of the original regexes. Second, comparing **RQ w (100)** and **RQ w/o (100)**, we notice that pretraining on weak labels is superior. It is worth pointing out that, as expected, the positional accuracy is consistently higher than the entity-level

accuracy.

#### 4.5.5 Impact of Active Sampling

In this section, we evaluate the performance of 4 different sampling strategies described in Section 4.3.4. The first two are random sampling baselines, one being a cold start version without pretraining and another with pretraining on weak labels generated by regexes from Section 4.5.2. The last two are uncertainty-based, both using the pretraining. Table 4.3 reports EntF1 and PosF1 scores achieved by the 4 approaches after 1,000 labeled candidate substrings. First, we observe that pretraining an NN on weakly labeled data (**RQ w/ (1000)**) is superior to the cold start training (**RQ w/o (1000)**). Second, we observe that uncertainty-based sampling is superior to random sampling on all 5 tasks. **RME** is slightly better than **ME**.

In Figure 4.3, we illustrate how the accuracy changes with fine-tuning on the **DATETIME** task until one labels 1,000 candidate substrings. We zoom in the tails of the learning curves in the small embedded figures. The more time a user spends on annotation, the better the performance of the NNs. The two uncertainty-based approaches are superior to random sampling. The behavior is consistent on other 4 tasks (not shown due to space constraints).

From Table 4.3, we observe that we can achieve around 0.95 PosF1 in all 5 recognition tasks, however, the EntF1 is always worse than PosF1. It indicates that the NN entity recognizer predicts partially correct entities quite often. As an example, 75% of the partial matches in **EMAILADREESS** task differ from the true entities by only 1-2 characters. This percentage is 35.3%, 30.9%, 36.1%, 30% on the other 4 tasks, respectively.

From these results, we conclude that, given a regex, it is beneficial to pre-train an initial model with weak labels. We also conclude that uncertainty-based sampling of candidate substrings for labeling is superior to random sampling.

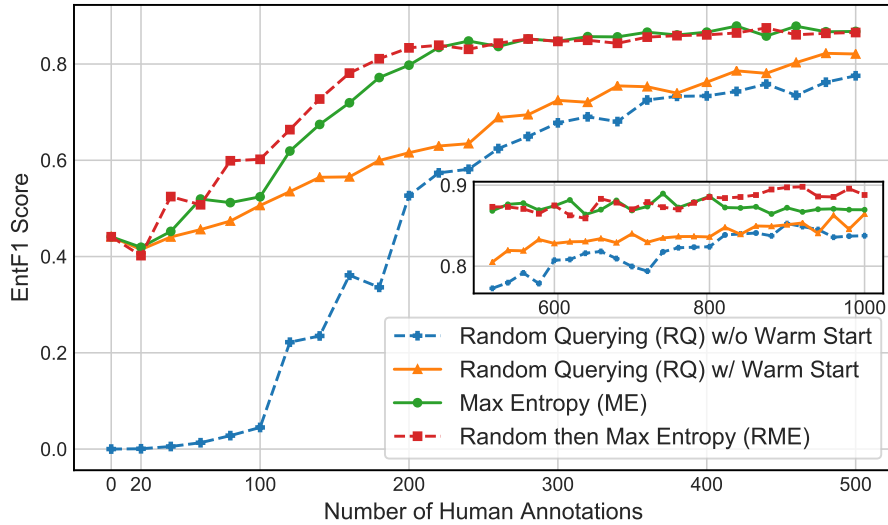


Figure 4.3: 4 query algorithms for **DATE**TIME recognition.

## 4.6 User Studies

In our framework, a user can invest her effort in coming up with a regex and/or in manual labeling of the candidate substrings. In this section, we explore the tradeoffs between the two, assuming a user is given a limited time budget of  $T$  minutes. We ignore the time to come up with the candidate regex  $RE_0$ , because it is much easier to come up with it than to come with a regex  $RE_{wl}$  that has both high recall and high precision.

### 4.6.1 Experimental Design

To experimentally explore the trade-off between creating a regex versus manual labeling of the candidate substrings, we collected data from 4 computer science students with different expertise in writing regexes. We use capital letters to represent the volunteers as M, C, J, and A. We asked the volunteers to create a regex for the **DATE**TIME and **COURSE**NUMBER tasks. For each task, we gave them 1,000 candidate substrings randomly selected from  $S_0$ . We instructed the volunteers to use {<https://regex101.com>} environment to create and debug regexes. We gave the volunteers 40 minutes to create a regex for each task. We also asked them to submit their intermediate regex after 5, 10, and 20 minutes of

work. Eventually, we obtained regexes from volunteers C, J, A for both tasks and regexes from volunteer M only for the **DATE** task.

Unlike regex writing, labeling candidate substrings does not require much, if any expertise. We assume all of the volunteers are average people and can create string annotations at the same speed as listed in Table 4.1. Using those numbers, we are able to simulate a range of strategies a volunteer may take to help with the **DATE** and **COURSE** tasks within our framework. The first two strategies are two extremes, while the next 4 strategies are the trade-offs:

- **RegAll**. User spends all the time on constructing a regex. The final entity recognizer is the regex created after 40 minutes.
- **Label**. User immediately starts to annotate candidate strings selected from  $S_0$ . NN is trained and fine-tuned using the labeled candidate substrings selected with Random Querying, as described in Section 4.3.4.
- **Reg5**. User spends the first 5 minutes on constructing  $RE_{wl}$ , which is used to pretrain an NN on weakly labeled data. Then, the user spends the remaining 55 minutes for annotating the candidate substrings selected using Random Querying.
- **Reg10**. The same as **Reg5**, but the user spends 10 minutes to construct  $RE_{wl}$  and 50 minutes for labeling.
- **Reg20**. The same as **Reg5**, but the user spends 20 minutes to construct  $RE_{wl}$  and 40 minutes for labeling.
- **Reg40**. The same as **Reg5**, but the user spends 40 minutes to construct  $RE_{wl}$  and 20 minutes for labeling.

Since there was no time for hyperparameter tuning in our real-time scenario, we had to select the hyperparameters in advance. Although it is not completely fair, in our experiments, we fixed all hyperparameters to a combination that appeared robust on all tasks in Section 4.5:  $lr = 2^{-7}$ ,  $emsize = 70$ ,  $nhidden = 125$ ,  $nlayers = 5$ .

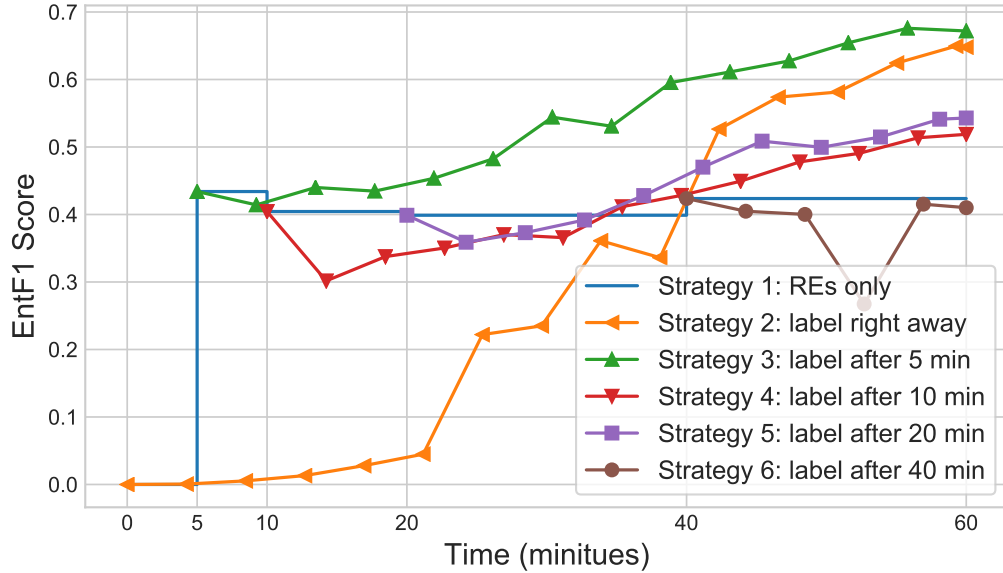


Figure 4.4: Time efficiency of volunteer M on **DATETIME** task.

#### 4.6.2 Experimental Results

Figure 4.4 shows the results for volunteer M, who had the most extensive expertise in writing regexes among our volunteers, on **DATETIME** recognition task. The figure shows EntF1 score as a function of time for the 6 different strategies. The figure allows us to compare different strategies for several different time budgets  $T = [5, 10, 20, 40, 60]$ .

Table 4.4: The performance of the 6 strategies under different time budgets of volunteers C, J, and A for **DATETIME** and **COUSENUMBER** recognition.  $\times$  means not applicable.

Strategy	Volunteer C on <b>DATETIME</b>					Volunteer J on <b>DATETIME</b>					Volunteer A on <b>DATETIME</b>				
	Time budget (min)					Time budget (min)					Time budget (min)				
<b>RegAll</b>	<b>0.002</b>	<b>0.412</b>	<b>0.456</b>	<b>0.593</b>	0.593	0.0	0.0	0.001	0.061	0.061	<b>0.002</b>	0.003	0.005	0.011	0.011
<b>Label</b>	0.001	0.006	0.036	0.377	<b>0.648</b>	<b>0.001</b>	<b>0.006</b>	0.036	0.377	<b>0.648</b>	0.001	<b>0.006</b>	0.036	0.377	<b>0.648</b>
<b>Reg5</b>	$\times$	0.0	0.0	0.0	0.0	$\times$	0.0	<b>0.417</b>	<b>0.489</b>	0.597	$\times$	0.0	<b>0.17</b>	<b>0.429</b>	0.614
<b>Reg10</b>	$\times$	$\times$	0.33	0.429	0.55	$\times$	$\times$	0.066	0.334	0.57	$\times$	$\times$	0.038	0.297	0.389
<b>Reg20</b>	$\times$	$\times$	$\times$	0.383	0.54	$\times$	$\times$	$\times$	0.271	0.385	$\times$	$\times$	$\times$	0.261	0.393
<b>Reg40</b>	$\times$	$\times$	$\times$	$\times$	0.626	$\times$	$\times$	$\times$	$\times$	0.232	$\times$	$\times$	$\times$	$\times$	0.416
Strategy	Volunteer C on <b>COUSENUMBER</b>					Volunteer J on <b>COUSENUMBER</b>					Volunteer A on <b>COUSENUMBER</b>				
	5	10	20	40	60	5	10	20	40	60	5	10	20	40	60
<b>RegAll</b>	<b>0.554</b>	<b>0.592</b>	0.532	0.338	0.338	<b>0.145</b>	0.184	0.19	0.19	0.19	<b>0.563</b>	<b>0.6</b>	<b>0.675</b>	0.702	0.702
<b>Label</b>	0.046	0.361	0.652	<b>0.761</b>	<b>0.815</b>	0.046	0.361	0.652	<b>0.761</b>	<b>0.815</b>	0.046	0.361	0.652	<b>0.761</b>	<b>0.815</b>
<b>Reg5</b>	$\times$	<b>0.591</b>	<b>0.682</b>	0.75	0.796	$\times$	<b>0.433</b>	<b>0.654</b>	0.751	0.795	$\times$	0.571	0.66	0.72	0.777
<b>Reg10</b>	$\times$	$\times$	0.679	0.734	0.771	$\times$	$\times$	0.611	0.732	0.798	$\times$	$\times$	0.651	0.698	0.769
<b>Reg20</b>	$\times$	$\times$	$\times$	0.733	0.796	$\times$	$\times$	$\times$	0.684	0.797	$\times$	$\times$	$\times$	0.681	0.727
<b>Reg40</b>	$\times$	$\times$	$\times$	$\times$	0.707	$\times$	$\times$	$\times$	$\times$	0.681	$\times$	$\times$	$\times$	$\times$	0.699

The figure reveals several interesting observations. If the time budget is only 5 minutes,

$RE_{wl}$  generated by volunteer M is superior in accuracy to an NN trained on candidate substrings labeled within 5 minutes. After 10 minutes, NNs become superior to using  $RE_{wl}$  only. The best trade-off between regex writing and labeling is achieved by **Reg5**. It seems that placing an extensive effort in improving  $RE_{wl}$  does not pay off: the regex created after 5 minutes is comparable in accuracy to the one created after 40 minutes. **Label** is not competitive initially, but after 60 minutes it catches up with the overall best **Reg5**.

To examine the generalizability of the conclusions with volunteers and different recognition tasks, we repeated the analysis with volunteer C, J, A on tasks **DATETIME** and **COURSENUMBER** in Table 4.4.

From Table 4.4 we see that volunteer C is different from volunteer M. While C's  $RE_{wl}$  produced after 5 minutes is not accurate, there is a steady increase in C's EntF1 accuracy after 10, 20, and 40 minutes. The observed accuracy after 40 minutes is higher than that of volunteer M. **RegAll** is better than the rest until the 50 minute mark is reached. After 50 minutes, the **Label** becomes more accurate than any regex-based approach. **Reg5** is extremely inaccurate. This is a surprising finding because, unlike **Label**, we do not observe any accuracy improvement with the increase in number of labeled candidate substrings. It seems that the NN pretrained using  $RE_{wl}$  created by volunteer C after 5 minutes has properties that prevent successful fine-tuning with labeled substrings.

Table 4.4 also shows results for volunteers J and A on task **DATETIME**. One can observe that neither volunteer manages to come up with a good regex in 40 minutes. Interestingly, the overall behavior of these 2 volunteers is more similar to volunteer M than to volunteer C. **Reg5** is the best overall in the first 40 minutes. **Label** becomes competitive with **Reg5** after around 40 minutes. **Reg10, 20, 40** show that it does not pay off to spend a large amount of time on writing and refining  $RE_{wl}$ .

Table 4.5 provides an insight into the differences between  $RE_{wl}$  produced after 5 minutes by volunteers C, J, and A. None of the volunteers is able to create an accurate  $RE_{wl}$ . This is expected knowing that they were exposed to 1,000 strings with 100-character lengths

and asked to write a regex within only 5 minutes. Interestingly, volunteer C created a very specific regex with precision 1 and very low recall, while volunteers J and A created regexes with very low precision and recall. Volunteer C’s  $RE_{wl}$  had only 34 matches in  $S_0$ , which meant that the resulting weakly labeled data set  $S_{wl}$  was extremely imbalanced with virtually all negative labels. We hypothesize that the high imbalance resulted in a very poorly pretrained NN, to the extent that it could not have been improved by fine-tuning. Unlike volunteer C, although volunteers J and A were not more successful with regexes, their  $RE_{wl}$  resulted in a more balanced weakly labeled data set, that allowed successful fine-tuning.

Table 4.5: Regexes created by C, J and A after the first 5 minutes for the **DATETIME** task.

Source	Regex and their properties			
C	\d{4}-\d{1,2}-\d{1,2}T\d{1,2}:\d{1,4}-\d{1,4}			
J	([0-9][0-9][0-9][0-9]  [0-9][0-9])\ / ([0-9][0-9]\ / [0-9][0-9])			
A	(Monday Tuesday Wednesday Thursday Friday Saturday Sunday) {0,1} \s* [0-9]{1,2}			
	No. of matches in $S_0$	EntPrec	EntRecall	EntF1
C	34	1.0	0.001	0.001
J	161,299	0.0	0.0	0.0
A	3,795,149	0.001	0.027	0.002

Table 4 shows that we obtain comparable results on the **COURSENUMBER** task with volunteers C, J, and A. An overall theme emerges from the user study and can be summarized as:

- If the time budget is less than 40 minutes, it is useful to spend a few minutes to construct  $RE_{wl}$  for weak labeling and the remaining allotted time for labeling.
- If the time budget is over 40 minutes, the weak labeling step could potentially be skipped and it might be sufficient to focus all effort on labeling of candidate substrings.

**Limitation of our study.** Before concluding the section, we point out that a limitation of our study is that we ignore the time needed to train and fine-tune an NN. Our assumption

is that the training is instantaneous. We use a standard PC with a single GeForce GTX 1080 Graphics Card in our actual experiments. For all tasks, excluding **DATE TIME**, pretraining an NN on  $S_{wl}$  took in the range of 20 minutes to an hour, and it took almost 2 hours for **DATE TIME**. Each round of fine-tuning on all data sets ranged from 2 to 20 minutes. Thus, it appears that the user would waste time waiting for an NN to be pretrained and fine-tuned. However, this limitation is not necessarily a fatal flaw of our study due to several reasons: (1) a user could proceed with manual labeling and regex construction while waiting for NN training, (2) a user could switch to some other task while waiting, (3) the training time could be significantly improved if it were implemented on a more powerful computer system, (4) our study did not focus on training speedups, and it is possible that with some tuning the training time could be further reduced.

## 4.7 Conclusions

We investigated the problem of entity extraction, where entities follow or closely resemble patterns described using regular expressions. Industrial strength entity recognizers for this class of entities employ regex. Regex is either manually crafted or learned. The main drawbacks of regexes are that they tend to be complex to achieve high coverage, are difficult to maintain, and are not resilient to noise, such as typos. In the wake of data deluge, deep learning algorithms are an attractive alternative, but they require large amount of human annotated data. We propose a framework that combines the advantages of regexes and deep learning, coupled with weak supervision and active learning.

We conducted extensive experiments with data from 5 application domains: email, course number, phone number, datetime, and bill date. We also conducted a user study with 4 volunteers. The experiments showed that we can build ML models that are regex-oblivious, achieve high accuracy, and are resilient to small noise. The user study provided interesting insights about the trade-offs between constructing regexes and manually labeling the unlabeled text.

## CHAPTER 5

# ITERATIVE REFINEMENT OF WORD AND CHARACTER EMBEDDING MODELS

### 5.1 Introduction

Representing words as real-valued vectors is crucial for modern natural language processing. Standard word embedding models such as Word2Vec and GloVe *Mikolov et al. (2013); Pennington et al. (2014); Le and Mikolov (2014)* learn to map a finite set of frequent words to vectors given a large unlabeled corpus by following the distributional hypothesis which states that similar words occur in similar contexts. As a result, semantically (*like vs love*) and syntactically (*take vs took*) similar words are likely to have similar embeddings.

Inability to represent rare and unseen words, which is referred to as the out-of-vocabulary (OOV) problem, limits applicability of the standard embedding approaches. In many downstream tasks, the OOV problem is prominent and inevitable. For example, in the e-recruitment domain there is an extremely large number of ways to write a job title for the same type of a job. LinkedIn users who are software engineers can describe themselves by phrases such as *software developer*, *sw engi.*, *sw developppppper* or *sw and web application developer*, which are caused by different variants, abbreviations, misspellings, or compounding. In this case, standard embedding for a finite set of job titles is not helpful when recommending jobs to users with rare or unseen job titles. Similar issues exist in many other domains such as microblogs analysis or query rewriting *Grbovic et al. (2015,0)*.

An embedding model that can effectively handle the OOV problem should preserve both semantic and syntactic characteristics of words and phrases. Recently proposed **mim-**

**icking** approach is a promising solution to the OOV problem. Its two representative algorithms are Mimick *Pinter et al. (2017)* and GWR *Kim et al. (2018)*. Their main idea is to train a character-level embedding neural network (NN) that can reconstruct, or *mimic*, the embedding from word embedding models. The trained character-level model is then used to generate both semantically and syntactically relevant embeddings for arbitrary character sequences. While Mimick and GWR used BiLSTM and CNN character-level NNs, similar approaches were recently proposed in *Schick and Schütze (2019)*; *Zhao et al. (2018)* with several other architectures.

Let us denote the word embedding model as  $W$  and the character embedding model as  $G$  in the mimicking mechanism. Let us consider model  $W$  produced by training on some Twitter data. The top 5 neighbors for word *food* in the vector space induced by  $W$  are *chinese, foodi, snacks, meals, foood*. After fitting  $G$  to mimick embeddings from  $W$ , the top 5 neighbors of *food* in the vector space induced by  $G$  become *foods, foood, foood, foood, snacks*, many of which are rare words that were not placed in the neighborhood by  $W$ . Thus, the syntactic similarities become influential in word embedding by model  $G$ , which is helpful when dealing with the OOV and rare word problem. As evidenced from word *snacks* found in the neighborhood, a well-trained  $G$  also retains ability to map different semantically related words into the same neighborhood. A well-trained  $G$  is also capable of avoiding orthographic errors.

We observed that the embedding of less common words produced by  $W$  is often not precise and that it negatively influences training of  $G$ . We also observed that initializing  $W$  with embeddings from  $G$  might improve word-level embedding of those less common words. Thus, to better capture the syntactic and semantic similarities between words while preventing orthographic mistakes, we developed an iterative mimicking framework that finds a good balance between word-level and character-level representations of words. In the proposed framework,  $W$  training is initialized by  $G$  embeddings,  $G$  is trained by mimicking  $W$  embeddings, and the process is repeated several times. As will be empirically

demonstrated in Section 5.7.1, the resulting procedure produces improved character-level embeddings on common, rare, and OOV words. We call the resulting approach the Iterative Mimicking (IM).

We make the following contributions:

- We propose a framework that iteratively refines  $W$  and  $G$  models. The character embedding model  $G$  in the final iteration is used to assign vectors to any input sequence such as OOV words.
- In intrinsic and extrinsic evaluations on five word similarity and 3 sentiment analysis tasks, we illustrate that our proposed IM is superior to the baselines.
- Finally, we demonstrate the effectiveness of the IM approach on job title normalization, which is an important e-recruiting challenge.

## 5.2 Related Work

In this section, we discuss relevant approaches that are able to generate representations for OOV words.

**Aggregation of  $n$ -grams or morphemes.** FastText *Bojanowski et al. (2016)* algorithm directly learns embeddings of  $n$ -grams with a Word2Vec objective (CBOW or SkipGram). The embedding for an OOV word is simply aggregated as the average of the word’s  $n$ -gram embeddings. In Charagram *Wieting et al. (2016)*, a nonlinear aggregation function is used instead of averaging  $n$ -gram embeddings. Aggregation over morphemes is discussed in *Qiu et al. (2014)*; *Cotterell and Schütze (2015)*; *Botha and Blunsom (2014)*.

**Character-aware modules in deep NNs.** Stacking deep NNs on top of character-level modules is a popular recent trend. For example, character BiLSTMs or CNNs are co-trained with a language model in ELMo *Peters et al. (2018)*, BERT *Devlin et al. (2018)*, GPT *Radford et al. (2018)*, and charCNN *Kim et al. (2016)*. Similar to mimicking, the

character-aware modules in those methods are able to assign vectors to OOV words. However, they are computationally expensive due to a need to train a heavy-weight language model on a large corpus to be effective.

There are also approaches that co-train the character-aware modules with auxiliary supervised tasks, such as part-of-speech tagging *Santos and Zadrozny (2014)*, text classification *Zhang et al. (2015)* and machine translation *Luong and Manning (2016)*, but the resulting embeddings are not necessarily reusable for other downstream tasks.

**Mimicking.** This approach allows learning light-weight character embedding models in a two-step manner *Pinter et al. (2017)*; *Kim et al. (2018)*; *Schick and Schütze (2019)*; *Zhao et al. (2018)*. It was illustrated that on multiple intrinsic and extrinsic evaluation tasks, mimicking yields superior performance to other baselines. Our proposed method is closely related to this line of research as will be described in detail in Section 6.3.

### 5.3 Methodology

We start this section by introducing the vanilla **mimicking** mechanism used in Mimick *Pinter et al. (2017)* and GWR *Kim et al. (2018)*. We then describe our proposed iterative mimicking (IM) mechanism, which is a generalization of the vanilla approaches.

#### 5.3.1 Mimick and GWR

The two algorithms share the same idea that a character embedding model  $G$  is learnt to mimic the word embedding model  $W$ .

**Word embedding model ( $W$ ).** Given a corpus  $D$  and a vocabulary  $V$ , word embedding models such as SkipGram and CBOW output a vector for each word in  $V$ , resulting in an embedding set  $\{(w_i, \mathbf{v}_{w_i}^W) | i, \dots, M\}$ , where  $w_i \in V$  is a word and  $\mathbf{v}_{w_i}^W \in \mathbb{R}^d$  is its vector representation. Word embedding models are said to be good at preserving semantic similarities where semantically similar words are assigned similar word embeddings.

**Character embedding model ( $G$ ).** Mimick and GWR generalize the fixed-vocabulary

semantic space by training a character neural network  $G$  (char-BiLSTM in Mimick or char-CNN in GWR) to mimick the embeddings from  $W$ . Generator  $G$  provides mapping from a character sequence to a vectors in the same space as  $W$  and is good at preserving syntactic similarities. After training, the generator is able to produce a vector for any input string.

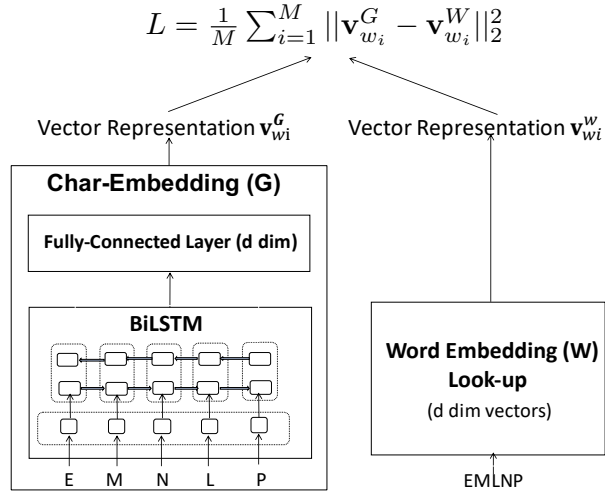


Figure 5.1: Mimick model.  $G$  is a character BiLSTM.

Figure 5.1 illustrates the architecture used in Mimick.  $G$  takes word  $w$ , which is a sequence of characters and generates a  $d$ -dimensional vector  $\mathbf{v}_w^G$  as output. Note that  $d$  has to match with the dimensionality of the embedding vectors produced by  $W$ . To train  $G$ , we minimize the squared Euclidean distance between  $\mathbf{v}_w^G$  and  $\mathbf{v}_w^W$  for  $w \in V$ , where  $M$  is the size of vocabulary  $V$ .

$$L = \frac{1}{M} \sum_{i=1}^M \|\mathbf{v}_{w_i}^G - \mathbf{v}_{w_i}^W\|_2^2 \quad (5.3-1)$$

After convergence, we use  $G$  to generate vectors for any character sequence.

### 5.3.2 Iterative Mimicking

When we fit  $G$  only once on  $W$ , we observe that the syntactic similarities dominate relationships between the words and that semantic similarities weaken as compared to those of embeddings produced by  $W$ . To address this issue, we propose an Iterative Mimicking (IM) mechanism as illustrated in Figure 5.2.

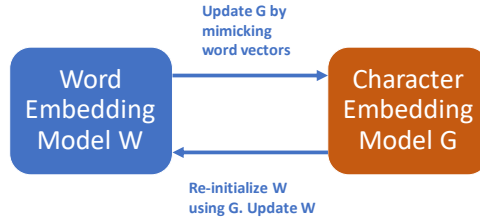


Figure 5.2: Illustration of iterative mimicking

In IM, the word embedding model  $W$  and character embedding model  $G$  retrofit each other in multiple iterations. After fitting  $G$  on  $W$  once, IM continues with re-training  $W$  by initializing  $\mathbf{v}_{w_i}^W$  with the current  $\mathbf{v}_{w_i}^G$  for each  $w_i \in V$ , as can be seen in Algorithm 1. Thanks to different initialization,  $W$  training will result in a different local minimum that should better represent the morphological information captured by  $G$ . This retrofitting will not only correct the vectors that  $G$  wrongly placed in the semantic space, but also enhance the learning of  $W$  by exploiting knowledge about syntactically similar words learned by  $G$ . Upon retraining  $W$ , IM proceeds by updating  $G$  based on the updated  $W$ , and the whole iterative process is repeated several times.

---

**Algorithm 1:** Iterative Mimicking

---

**Input:** Corpus  $D$ , vocabulary  $V$ ,  $N$  iterations  
**Output:** Character embedding model  $G$

- 1 **Start**
- 2 Initialize word embedding model  $W$  with random word vectors;
- 3 Initialize character embedding model  $G$  (e.g. char-LSTM or CNN) with random weights;
- 4 **for**  $i = 1 : N$  **do**
- 5     If  $i > 1$ : Replace  $\mathbf{v}_{w_i}^W$  with  $\mathbf{v}_{w_i}^G$  induced by current  $G$  for all  $w_i \in V$ ;
- 6     Train a word embedding model  $W$  on corpus  $D$  for all words  $w_i$  in  $V$ ;
- 7     Train  $G$  until convergence by minimizing  $L = \frac{1}{M} \sum_{i=1}^M \|\mathbf{v}_{w_i}^G - \mathbf{v}_{w_i}^W\|_2^2$  where  $\mathbf{v}_{w_i}^W$  come from  $W$ ;
- 8 **end**
- 9 **Return**  $G$

---

The algorithm terminates after  $N$  iterations, after the gap between the word space  $W$  and the character-embedding space produced by  $G$  is sufficiently small, or after the embedding by  $G$  stabilizes. Mimick *Pinter et al.* (2017) and GWR *Zhao et al.* (2018) are special cases

Table 5.1: Pearson correlation for different embedding models on the 5 data sets

Training Corpus		RG65	Simlex	WS353	RW	MEN-3000
Text8	<b>FastText</b>	0.551	0.273	0.621	<b>0.422</b>	0.613
	<b>Mimick</b>	0.418	0.142	0.439	0.206	0.390
	<b>Mimick + IM</b>	0.463	0.238	0.512	0.282	0.474
	<b>GWR</b>	0.572	0.270	0.627	0.276	0.624
	<b>GWR + IM</b>	<b>0.643</b>	<b>0.294</b>	<b>0.673</b>	0.317	<b>0.664</b>
Twitter	<b>FastText</b>	0.662	0.196	0.415	<b>0.277</b>	<b>0.628</b>
	<b>Mimick</b>	0.312	0.061	0.204	0.102	0.316
	<b>Mimick + IM</b>	0.355	0.092	0.242	0.118	0.385
	<b>GWR</b>	0.560	0.202	0.403	0.161	0.578
	<b>GWR + IM</b>	<b>0.683</b>	<b>0.221</b>	<b>0.467</b>	0.189	0.612

of our algorithm when the number of iterations is  $N = 1$ .

## 5.4 Word Similarity

We first perform intrinsic evaluation of embedding models on word similarity tasks. Given word pairs and the word similarity judgements by a human, the quality of embedding models is measured as the correlation between the human judgment and the cosine similarity of word embeddings.

### 5.4.1 Experimental Setup

We use five standard word similarity data sets for English language: RG-65 *Rubenstein and Goodenough* (1965), Simlex *Hill et al.* (2015), WordSim-353 *Agirre et al.* (2009), Stanford’s RW *Luong et al.* (2013), and MEN-3000 *Bruni et al.* (2014).

We compare the following embedding models: (1) **FastText**. Embedding of a word is averaged embedding of the word’s  $n$ -grams. (2) **Mimick**. We implement the original Mimick algorithm by using the same char-BiLSTM architecture as in the original paper *Pinter et al.* (2017), except that we set the hidden dimension in the char-BiLSTM to 300. (3) **Mimick + IM**. We use the same char-BiLSTM as in Mimick, but update it with the proposed iterative mimicking (IM) mechanism. (4) **GWR**. We implement the original GWR

algorithm and use the same architecture for char-CNN as in the original paper *Kim et al.* (2018). (5) **GWR + IM**. We use the same char-CNN as in GWR, but update it with the proposed iterative mimicking (IM) mechanism.

To train FastText and the word embedding model  $W$  in the four mimicking models we used two different data sets: (1) Text8 <sup>1</sup>, which consists of the first 100,000,000 bytes of Wikipedia text; (2) Twitter set *Wang et al.* (2015), which contains 4 millions Tweets about different topics. Twitter corpus is used to see the effect of training on informal and noisy sentences.

For models trained on both data sets, we generate an embedding for the words in a pair. Then, we calculate cosine similarity between the two generated embeddings. Finally, we calculate the Pearson correlation between the cosine similarities and the similarity scores provided by human annotators. We repeat each experiment 3 times and average the results.

**Hyperparameters:** We set the dimensionality of embeddings to 100 for all five models. For FastText, we choose  $n$ -grams with  $n$  ranging from 2 to 5. We instantiate the word embedding model  $W$  as SkipGram in models (2)-(5). For both FastText and SkipGram, we set the context window at 5. We run FastText for 10 epochs. In Mimick and GWR, we train SkipGram for 10 epochs, followed by 100 training epochs of character embedding model  $G$ . For iterative mimicking models, we run 5 iterations, during each of which we train 20 epochs for  $G$  and 2 epochs for  $W$ .

## 5.4.2 Results

Table 5.1 shows the Pearson correlation for the 5 models on the 2 data sets. Most models perform better when trained on the Text8 corpus, due to its more diverse and formal vocabulary. We observe that GWR outperforms Mimick on all data sets, showing that character-level CNN is superior to LSTM both in performance and training time. Both Mimick and GWR gain significant improvements through our proposed iterative process

---

<sup>1</sup><http://mattmahoney.net/dc/textdata.html>

on most data sets, confirming our hypothesis that iteratively retrofitting the word and the character-level neural network does improve the embedding quality.

## 5.5 Sentiment Analysis

We also wanted to see how different embedding approaches would affect downstream tasks such as sentiment analysis. To test this, we let each model described in Section 5.4 produce embeddings and use the embeddings to feed them into a simple LSTM for sentence classification. We also consider a naive baseline that embeddings of OOV words is a random vector.

### 5.5.1 Experimental Setup

We evaluate embeddings on 3 different sentiment analysis data sets: Sentiment140 Tweets *Go et al.* (2009), IMDB Movie Reviews *Maas et al.* (2011), and Stanford’s Sentiment Treebank (STT) *Socher et al.* (2013). Each data set contains a training set and a test set. For the first two data sets, we train all word and character-level models on the training set only. For STT, due to the small size of the training corpus, we opt to train  $W$  on Text8 to produce embeddings for the downstream task.

## 5.5.2 Results

Table 5.2: Test accuracy on sentiment analysis data sets

Model	Dataset		
	Sentiment140	IMDB	STT
<b>Word2Vec + Rand</b>	0.819	0.874	0.742
<b>FastText</b>	0.824	0.874	<b>0.751</b>
<b>Mimick</b>	0.781	0.797	0.692
<b>Mimick + IM</b>	0.816	0.855	0.743
<b>GWR</b>	0.830	0.863	0.720
<b>GWR + IM</b>	<b>0.847</b>	<b>0.889</b>	0.747

Table 5.2 summarizes the accuracy of all models on the 3 test sets. Again, iterative GWR outperforms all models on 2 out of 3 data sets. However, FastText proves to be a competitive baseline, outperforming our models on the STT dataset. Even so, our iterative algorithm still shows improvements compared to both Mimick and GWR, indicating that the resulting embeddings are also beneficial to downstream predictive tasks.

## 5.6 Job Title Normalization

We dedicate the rest of this paper for the application of our algorithm on the task of job title normalization. We formally describe the **job title normalization** problem as follows. We are given a corpus  $D = \{(t_i, d_i) | i = 1, \dots, N\}$ , where  $t_i$  is the job title and  $d_i$  is the job description. Both a job title and a job description are free text entered by a user of a professional network such as LinkedIn, Indeed, or CareerBuilder. There are typically no hard constraints imposed by the professional networks when it comes to entering job titles and descriptions, which results in a very diverse and noisy corpus. Let us denote the vocabulary of job titles as  $\mathcal{T}$ , and the vocabulary of words in job descriptions as  $\mathcal{V}$ . Both  $\mathcal{T}$  and  $\mathcal{V}$  are infinite sets. Let us also suppose there is a job title taxonomy  $\mathcal{O} =$

$\{o_1, o_2, \dots, o_m\}$ , which is a finite set denoting  $m$  distinct job categories. An example of such a taxonomy is the Standard Occupational Classification (SOC), whose O\*NET-SOC 2010 release contains 1,100 job titles *Elias et al.* (2010). The task is to normalize job title  $t \in \mathcal{T}$  into one job category  $o \in \mathcal{O}$  from the taxonomy.

To classify a job title into a category, we first learn how to embed an input string representing a job title into a vector space. Using the resulting embedding model we also map all job categories into the same vector space. Given a particular job title, we assign it to the most similar job category in the vector space. Due to the large variety of ways people can write job titles, models that are able to generate high quality embeddings for rare or OOV job titles are preferable. We note that this approach does not require manually labeled training data, unlike the job normalization approach proposed in *Neculoiu et al.* (2016). All that is required is a corpus of (job titles, job description) pairs to train the embedding model.

### 5.6.1 Data Set

We obtained a list of 1.1 million (job title, job description) pairs from a large professional social network. Example pairs are listed in Table 5.3. The job titles in the data set encompass a broad range of industries and occupations. We process the job titles by replacing the white spaces with underscores and add a # at the beginning of a job title to distinguish a job title from a word in job descriptions. We remove punctuations in the job descriptions. All job titles and words in the data set are lowercased. The number of unique job titles is 630,000 and the number of unique words in job descriptions is 920,000.

We plot the distribution of job title frequencies in rounded logarithm scale in Figure 5.4. As we can see, most of the job titles (around 300,000) appear only once in  $D$ .

**Test Job Titles** We select 1,000 testing job titles ranging widely in frequencies. Specifically, we sample 125 job titles randomly from a frequency range  $[2^r, 2^{r+1})$ , where  $r$  is selected from  $[4, 5, 6, 7, 8, 9]$ , resulting in 750 job titles. Another 125 random job titles are

Table 5.3: Job title and its description

<i>software engineer</i>
I'm a python developer building a Web application for job recruiters to search multiple job boards at once like Kayak.com. For recruiters, it's a highly concurrent application using MongoDB for the backend.
<i>visiting assistant professor</i>
Research in differential algebra and mathematical logic. Taught graduate and undergraduate courses in Logic Model Theory, Theory of Computation.

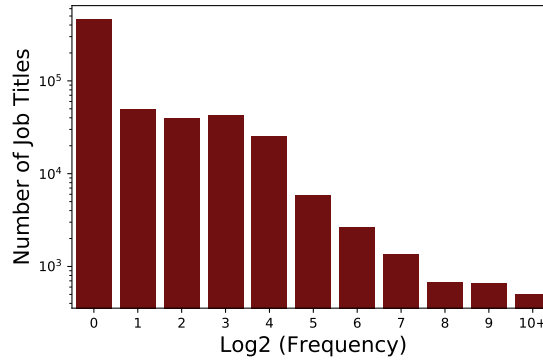


Figure 5.3: Distribution of job title frequencies. X-axis is the rounded logarithm of job title frequencies; Y-axis is the number of job titles with a corresponding frequency in log scale.

sampled from range  $[1, 2^4)$  and 125 random job titles from range  $[2^{10}, \infty)$ . Let us denote the testing job titles from the 8 different ranges as  $\mathcal{T}_1, \mathcal{T}_2, \dots, \mathcal{T}_8$ , where  $\mathcal{T}_1$  contains the rarest job titles and  $\mathcal{T}_8$  contains the most frequent job titles.

## 5.6.2 Customized IM Embedding Models

We customize the embedding models  $W$  and  $G$  in the IM framework to handle the corpus  $D$  coming as (job title, job description) pairs. We call our model as **CNN-LSTM-Attn + IM**.

**Word embedding model  $W$ .** We preprocess our data set  $D$  in order to learn word embedding model  $W$  for tokens in both  $\mathcal{T}$  and  $\mathcal{V}$ . We insert each job title  $t_i$  into the job description  $d_i$  at  $n$  random positions where  $n = \frac{\text{length}(d_i)}{10}$ . Thus, we form a new data set  $S = \{d'_i | i = 1, \dots, N\}$ , where  $d'_i$  is obtained by adding  $t_i$  to  $d_i$ . We then feed the sequences

$S$  to SkipGram Word2Vec algorithm. As a result, job titles and words in job descriptions are in the same vector space. We note that this job title insertion process results in fundamentally the same embedding as if we applied a joint embedding algorithm such as StarSpace Wu *et al.* (2018) on the original (job title, job description) pairs.

**Character embedding model  $G$ .** Based on our experimental results in Sections 5.4 and 5.5, we used a character CNN-LSTM with attention model for  $G$  as illustrated in Figure 5.4. The resulting model combines the strengths of both Mimic and GWR models. A job title such as *sr java developer* is preprocessed as a set of tokens  $\{ 'sr', 'java', 'developer', 'sr\_java\_developer' \}$ . Each token  $w_i$  is first fed into a character-level CNN with Highway layers to obtain its word embedding  $e_i^w$ . The CNN layers are identical to the architecture used in Kim *et al.* (2016) and Kim *et al.* (2018). Each  $e_i^w$  is then taken as an input to a Bi-directional LSTM to exploit sequential information and obtain outputs  $h^{blstm}$ . Finally, a self-attention layer is placed on top of the Bi-LSTM to pool the most important time steps and obtain a single vector  $v^{lstm}$ , before a linear-transformation to obtain the final embedding  $v_{w_i}^G$ .

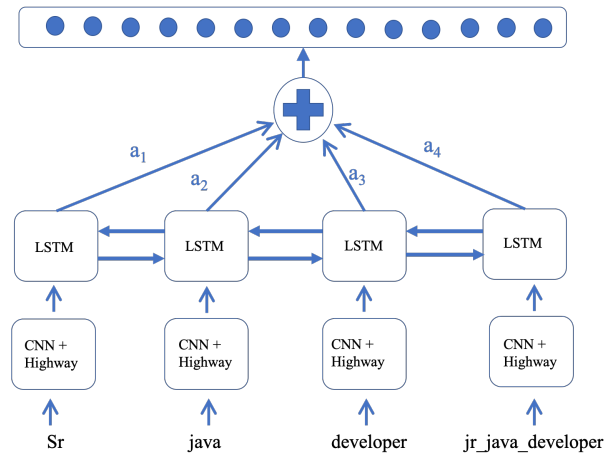


Figure 5.4: Architecture of CNN-LSTM attention model. Job title *jr java developer* is broken into a set of tokens  $\{jr, java, developer, jr\_java\_developer\}$ .

Raw	freq_range	AutoCoder	FastText	Mimick	Mimick+IM	CNN-LSTM-Attn+IM
Registered Family Mediator Ba In Child And You...	T1	Child, Family, And School Social Workers	Enbalmers	Mental Health And Substance Abuse Social Workers	Child, Family, And School Social Workers	Child, Family, And School Social Workers
Econometrics Intern	T2	Not Classified	Economists	Social Science Research Assistants	Financial Quantitative Analysts	Economists
Consultantowner	T3	Rehabilitation Counselors	Industrial-Organizational Psychologists	Wind Energy Project Managers	Logistics Managers	Chief Executives
Studio Intern	T4	Laborers And Freight, Stock, And Material Move...	Graduate Teaching Assistants	Agricultural Technicians	Multimedia Artists And Animators	Multimedia Artists And Animators

Figure 5.5: Example of job title normalization results from 5 competing models

### 5.6.3 Baselines

We compare our method with three baselines. Two of them are automatic vector space normalization methods similar to ours and one is a job normalization system maintained for years by domain experts.

(1) **AutoCoder** is a commercial software <sup>2</sup>. It implements a keyword searching algorithm, which is capable of mapping any job title to an O\*Net-SOC category. It is an enhanced variant of a software that was originally developed for the U.S. Department of Labor. (2) **FastText**. In this baseline, vectors of  $n$ -grams are directly learned by FastText algorithm on corpus  $S$ . In our experiment, we set  $n = [2, 3, 4, 5]$ . (3) **Mimick**. We use the char-BiLSTM from *Pinter et al. (2017)* to mimick all embeddings from  $W$ , including both job titles and regular words from job descriptions that occur more than 10 times. (4) **Mimick + IM**. It is the iterative Mimick version following our proposed approach.

For all models, once the vectors are obtained for job titles and job taxonomy, we normalize a job title to its nearest category in the vector space.

### 5.6.4 Hyperparameters

We set the embedding dimension as 100 for all embedding models.

We select job titles or words with frequency larger than  $2^3$  to train FastText and Skip-Gram models.

<sup>2</sup><https://www.onetsocautocoder.com/plus/onetmatch>

Around 80,000K common job titles in  $\mathcal{T}$  and 71,000K common regular words in  $\mathcal{V}$  are used to train the two models.

FastText is trained for 10 epochs. SkipGram and character BiLSTM in Mimick are trained for 10 and 100 epochs respectively. Iterative mimicking models are trained for 5 iterations, with training SkipGram for 2 epochs and character embedding models for 20 epochs in each iteration.

### 5.6.5 Evaluation Metrics

We show qualitative normalization results in Figure 5.5. For the 4 testing titles with different frequency, CNN-LSTM-Attn + IM consistently normalizes them to a relevant category. More results can be found in Appendix 5.7.2. We also conduct quantitatively evaluation on 5 models.

**Evaluating job title normalization:** We ask 6 human evaluators to judge the job title normalization quality of the 5 competing models. We design our evaluation scheme similar to *Liu et al. (2015)*. For each testing job title, we list randomly shuffled normalization results produced by each model. Then, human evaluators are asked to choose the most related job category among the shown normalization results. If a model produces the winner category, it receives 1 point; otherwise, it receives 0 point. If multiple models produce the winning category, all of them receive 1 point. The normalization quality of a model is the sum of the points divided by 1000.

Table 5.4: Testing Scores of each model by different human judges

	Judger1	Judger2	Judger3	Judger4	Judger5	Judger6	Avg
<b>AutoCoder</b>	0.636	0.679	0.546	0.628	0.540	<b>0.698</b>	0.621
<b>FastText</b>	0.552	0.581	0.476	0.518	0.511	0.503	0.524
<b>Mimick</b>	0.501	0.527	0.485	0.571	0.437	0.554	0.513
<b>Mimick + IM</b>	0.673	0.672	0.583	0.631	<b>0.646</b>	0.646	0.642
<b>CNN-LSTM-Attn + IM</b>	<b>0.758</b>	<b>0.767</b>	<b>0.647</b>	<b>0.719</b>	0.587	0.685	<b>0.694</b>

We note that there are cases when it is difficult for human evaluators to pick the most related category from the list, especially for rare titles, when all models produce different

Table 5.5: Average testing Scores of each model in different frequency ranges

Frequency Ranges	$\mathcal{T}_1$	$\mathcal{T}_2$	$\mathcal{T}_3$	$\mathcal{T}_4$	$\mathcal{T}_5$	$\mathcal{T}_6$	$\mathcal{T}_7$	$\mathcal{T}_8$
<b>AutoCoder</b>	0.466	0.501	0.537	0.614	0.625	<b>0.721</b>	<b>0.777</b>	0.732
<b>FastText</b>	0.408	0.419	0.481	0.519	0.581	0.582	0.635	0.611
<b>Mimick</b>	0.371	0.368	0.536	0.513	0.479	0.649	0.67	0.616
<b>Mimick + IM</b>	0.587	<b>0.665</b>	0.588	0.557	0.653	0.667	0.749	0.723
<b>CNN-LSTM-Attn + IM</b>	<b>0.592</b>	0.623	<b>0.658</b>	<b>0.668</b>	<b>0.746</b>	0.718	0.756	<b>0.792</b>

but similarly good results. For example, job titles can be blended such as *engineer and statistician*. Some models may normalize the job title to *engineers* and others to *statisticians*. In such cases, we allowed the human evaluators to choose more than one category as co-winners. Rare job titles can also be ambiguous, such as *energy healing and body work for women*. For ambiguous job titles, human evaluators were allowed to search on Google to find their meaning. We also observe that the categories in the O\*NET-SOC taxonomy can be very similar. For instance, *acrobat software engineer* is normalized to *software developers-application*, *software developers-systems software*, *computer systems engineers/architects* by different models, all of which are equally acceptable. In such cases, human evaluators were also allowed to choose more than one category as co-winners.

### 5.6.6 Results

Table 5.4 shows the scores given by 6 human evaluators. Compared to the baselines, our proposed framework achieves the best normalization quality. Five out of 6 human evaluators rate our IM models superior when it comes to normalizing job titles, with CNN-LSTM-Attn + IM being the best model. Experimental results also show that by applying IM to the Mimick LSTM model, the quality of the normalization significantly improves. We also report the average score by the 6 evaluators for each of the frequency ranges in Table 5.5. As expected, all models perform better on the more frequent job titles. When compared to the commercial AutoCoder software, we can see that our best model is comparable on the frequent job titles. AutoCoder outperforms our model in on  $\mathcal{T}_6$  and  $\mathcal{T}_7$  and our model is better on the most frequent group  $\mathcal{T}_8$ . However, for the less frequent job titles covering

groups  $\mathcal{T}_1$  to  $\mathcal{T}_5$ , our approach is significantly more accurate than AutoCoder. This is an impressive result considering that our model is trained in a few hours on an unsupervised document corpus, while AutoCoder is based on multiple years of painstaking tuning. More examples on title normalization can be found in Table in Supplementary Section.

## 5.7 Case Studies

### 5.7.1 Nearest Neighbors Analysis

We discussed our intuition of iterative mimicking in the introduction Section. In this section, we conduct nearest neighbor analysis to quantitatively validate our intuition. In Table 5.6, we list the nearest neighbors of example words in two domains based on the embeddings obtained from 3 embedding models. The embedding models for words in Tweets are from Section 5.5; and those for job titles are obtained from Section 5.6. As clearly can be seen, although Mimick is able to mimic the nearest neighbors of Word2Vec to some extent, it relies too much on the surface form of words or job titles. By retrofitting Word2Vec and Char-BiLSTM iteratively, our proposed Iterative Mimicking generate nearest neighbors with better quality.

### 5.7.2 Job Title Normalization

In this section, we show complementary results for Section 5.6. We list the normalization results for 8 testing job titles with different frequency. In the first example, only our proposed CNN-LSTM-Attn + IM succeed to recognize *Ta* is the abbreviation of *Teaching Assistant*. In other cases, we are comparable to Mimick and Mimick + IM. Certainly, we make mistakes for *Operations Support Manager*.

Table 5.6: Nearest Neighbors of different embedding models

Word	Word2Vec Neighbors	Mimick	Mimick + IM
<b>calculus tutor</b>	<i>private mathematics tutor</i>	<i>associate tutor</i>	<i>math tutor</i>
	<i>math lab tutor</i>	<i>cs tutor</i>	<i>instructor of mathematics</i>
<b>Job titles</b>	<i>course grader</i>	<i>mathematics tutor</i>	<i>mathematics instructor</i>
	<i>mathstat tutor</i>	<i>gcse tutor</i>	<i>science and mathematics tutor</i>
	<i>master tutor</i>	<i>mathematics instructor</i>	<i>teaching assistant private tutor</i>
	<i>graduate student data science lab</i>	<i>coresearcher</i>	<i>natural language processing research</i>
<b>nlp researcher</b>	<i>senior scientist</i>	<i>external researcher,</i>	<i>linguist researcher,</i>
	<i>phd student with the artificial intelligence group</i>	<i>nlp research engineer,</i>	<i>senior researcher part time</i>
	<i>natural language processing research</i>	<i>research linguist,</i>	<i>nlp research engineer</i>
	<i>speech data evaluator</i>	<i>researcher natural language processing</i>	<i>linguistics undergraduate researcher</i>
<b>waitress</b>	<i>waiter</i>	<i>assistant to the mayor</i>	<i>barman</i>
	<i>bartender</i>	<i>media relations assistant</i>	<i>waiter</i>
	<i>hostess</i>	<i>assistant to head officer of public relations</i>	<i>front desk runner</i>
	<i>server</i>	<i>catman assistant</i>	<i>catering assistant</i>
	<i>barman</i>	<i>welders assistant</i>	<i>food server</i>
<b>sofftware engineeecer</b>	-	<i>research staff member senior software engineer</i>	<i>c software engineer</i>
	-	<i>researchersoftware engineer</i>	<i>sr software engineer</i>
	-	<i>researcher software engineer</i>	<i>software rd engineer</i>
	-	<i>external rd software engineer</i>	<i>staff software developer</i>
<b>(OOV)</b>	-	<i>researcher software engineering unit</i>	<i>engineer developer</i>
<b>teaching assistant for principles of data science</b>	-	<i>teaching assistant computer science</i>	<i>teaching assistant big data mining</i>
	-	<i>teaching assistant data analyst</i>	<i>data science instructor</i>
	-	<i>teaching assistant computer science dept</i>	<i>instructor data scientist in residence</i>
<b>(OOV)</b>	-	<i>teaching assistant at department of computer science</i>	<i>teaching assistant dept of computer science</i>
	-	<i>teaching assistant dept of computer science</i>	<i>research associate data science lecturer</i>
<b>Tweets</b>	<i>nyc</i>	<i>manhattans</i>	<i>manhattans</i>
	<i>queens</i>	<i>manhatan</i>	<i>midtown</i>
	<i>midtown</i>	<i>cmanhattan</i>	<i>aftersandy</i>
	<i>dumbo</i>	<i>nyc</i>	<i>nyc</i>
	<i>williamsburgbridge</i>	<i>manhattan</i>	<i>dumbo</i>
<b>lightning</b>	<i>downpours</i>	<i>lightening</i>	<i>lightning</i>
	<i>thundering</i>	<i>lighting</i>	<i>struck</i>
	<i>explosions</i>	<i>lightninglt</i>	<i>fog</i>
	<i>torrential</i>	<i>flash</i>	<i>lightningggg</i>
	<i>light</i>	<i>lightway</i>	<i>thundering</i>
<b>develop</b>	<i>create</i>	<i>developing</i>	<i>developing</i>
	<i>establish</i>	<i>develops</i>	<i>develops</i>
	<i>bargaining</i>	<i>develope</i>	<i>improve</i>
	<i>sustain</i>	<i>crease</i>	<i>create</i>
<b>(OOV)</b>	<i>create</i>	<i>developed</i>	<i>developed</i>
<b>prelecture</b>	-	<i>lecturing</i>	<i>lecturer</i>
	-	<i>innocence</i>	<i>lecturing</i>
<b>(OOV)</b>	-	<i>liban</i>	<i>advising</i>
	-	<i>humanist</i>	<i>philosophy</i>
<b>fishering</b>	-	<i>preparados</i>	<i>discussion</i>
	-	<i>fishermen</i>	<i>fisherman</i>
<b>(OOV)</b>	-	<i>craftbeer</i>	<i>fishers</i>
	-	<i>slobbering</i>	<i>fishery</i>
	-	<i>gathering</i>	<i>fishes</i>
	-	<i>farms</i>	<i>water</i>
	-		<i>nord</i>

## 5.8 Conclusions

In this work we introduce a light-weight framework that enhances the existing mimicking models by iteratively retrofitting a word-level and a character-level embedding neural network. Intrinsic and extrinsic evaluations show that our algorithm outperforms the baselines.

We also show that our algorithm can be successfully applied to the task of job title normalization, a challenging problem in the e-recruitment domain. Unlike previous work such as *Neculoiu et al. (2016)*, our approach can be applied directly on an unlabelled corpus consisting of job title and job description pairs. Unlike previous approaches and the existing job normalization software, our framework is not dependent on any specific taxonomy.

Raw	freq_range	AutoCoder	FastText	Mimick	Mimick+IM	CNN-LSTM-Attn+IM
Art Teacher Cochair Sdh Art Dept	T1	Art, Drama, And Music Teachers, Postsecondary	Kindergarten Teachers, Except Special Education	Education Teachers, Postsecondary	Art, Drama, And Music Teachers, Postsecondary	Art, Drama, And Music Teachers, Postsecondary
Db Udb Tester Developer Coop	T1	Database Administrators	Software Quality Assurance Engineers And Testers	Database Administrators	Data Warehousing Specialists	Software Quality Assurance Engineers And Testers
Founding Board Member	T4	Electrical And Electronic Equipment Assemblers	Fundraisers	Fundraisers	Chief Executives	Chief Executives
Senior Systems Programmer	T5	Computer Programmers	Computer Programmers	Computer Systems Engineers/Architects	Computer Programmers	Computer Systems Engineers/Architects
Head Golf Professional	T5	Coaches And Scouts	Recreation And Fitness Studies Teachers, Posts...	Athletes And Sports Competitors	Athletes And Sports Competitors	Athletes And Sports Competitors
Hr Intern	T6	Human Resources Assistants, Except Payroll And...	Human Resources Managers	Human Resources Assistants, Except Payroll And...	Human Resources Assistants, Except Payroll And...	Human Resources Assistants, Except Payroll And...
Gis Intern	T6	Geographic Information Systems Technicians	City And Regional Planning Aides	Geoscientists, Except Hydrologists And Geograp...	Geospatial Information Scientists And Technolo...	Geographic Information Systems Technicians
Social Media Intern	T7	Public Relations Specialists	Public Relations Specialists	Advertising And Promotions Managers	Public Relations Specialists	Public Relations Specialists
Store Manager	T8	General And Operations Managers	First-Line Supervisors Of Retail Sales Workers	Food Service Managers	First-Line Supervisors Of Retail Sales Workers	First-Line Supervisors Of Retail Sales Workers

Figure 5.6: Example of job title normalization

While we find that O\*NET-SOC is a useful taxonomy that captures a broad range of occupations, some may not find it to be sufficiently fine-grained for modern job market *Javed et al.* (2015). Thus, if users have an alternative taxonomy which they believe is better, they can simply use our trained character-level neural network to project their taxonomy into the job title vector space. Alternatively, in cases when users already have a large database of job postings or resumes, they may discover their own job taxonomy by using clustering algorithms such as hierarchical clustering on the vector space of job titles.

## CHAPTER 6

# GRAPH NEURAL NETWORKS FOR PROPERTY ESTIMATION

### 6.1 Introduction

Accurate estimation of chemical and physical properties is critical for new discoveries of molecules and materials. Numerical approximation methods such as ab initio DFT calculation have been studied for centuries by physicians. On one hand, it is costly in computation for large molecules. On the other hand, it can not make use of the large amount of experimental data accumulated through high throughput experiments. Therefore, In the last decade, we have witnessed machine learning being applied to the field, since machine learning algorithms are efficient and effective given enormous training data.

In this paper, we study two tasks on inorganic molecules: the band gap ( $E_g$ ) and formation energy ( $E_f$ ) prediction, using supervised learning algorithms. We summarize our contributions as below:

- We give a brief summary of the literature on property estimation tasks from the machine learning view-point. We group the machine learning algorithms into two types based on how they handle the molecule as input: *feature-driven algorithms* and *neural networks*.
- We propose a simple graph representation for molecules and three neural network architectures that is able to directly learn predictive functions from graphs.
- We evaluate 10 algorithms on two tasks with Material Project data: band gap predic-

tion and formation energy prediction. Six out of the ten algorithms are feature-driven and four are neural networks. To our best knowledge, this is the first time a comprehensive comparison of the literature is done on property estimation problem for inorganic molecules. We discovered that, it is true neural networks are superior than feature-driven algorithms for formation energy prediction. However, the superiority can not be reproduced on band gap prediction. We also discovered that our proposed simple shallow neural networks perform comparably with the state-of-the-art deep neural networks.

## 6.2 Related Work

Supervised machine learning is applied with two phases, training phase, and testing phase. Taking  $E_g$  prediction as an example, at training phase, a supervised algorithm learns a mapping  $f(M; \mathbf{W}) \rightarrow y$  given a training dataset  $D_{train} = \{(M_i, y_i) | i = 1, \dots, N\}$ , where  $M_i$  represents the  $i$ th molecule and  $y_i$  represents its corresponding  $E_g$ .  $\mathbf{W}$  contains the learnable parameters of  $f$  and is updated during training. At the testing phase,  $E_g$  of a new molecule is estimated using  $f(M_{new})$ . We group the machine learning algorithms in the literature into two types based on how they handle a molecule:

### 6.2.1 Feature-driven algorithms

In this type of researches, the effort of a researcher is spent mainly on feature engineering. Feature engineering tries to represent a material as a  $d$ -dimensional vector, denoting as  $\mathbf{x} \in R^d$ . The representation vectors should be carefully chosen, invariant to transformations of molecules, such as translation, rotation, and permutations of like elements *Ramprasad* (2016). A widely used set of features contain 145 values, being purely engineered based on atomic attributes of a material *Ward et al.* (2016). They reported the predictive power of the 145 attributes on  $E_g$  and  $E_f$  prediction tasks with OQMD dataset [REF]. Starting with the formula of a material, the 145 atomic attributes fall into 4 categories: stoichio-

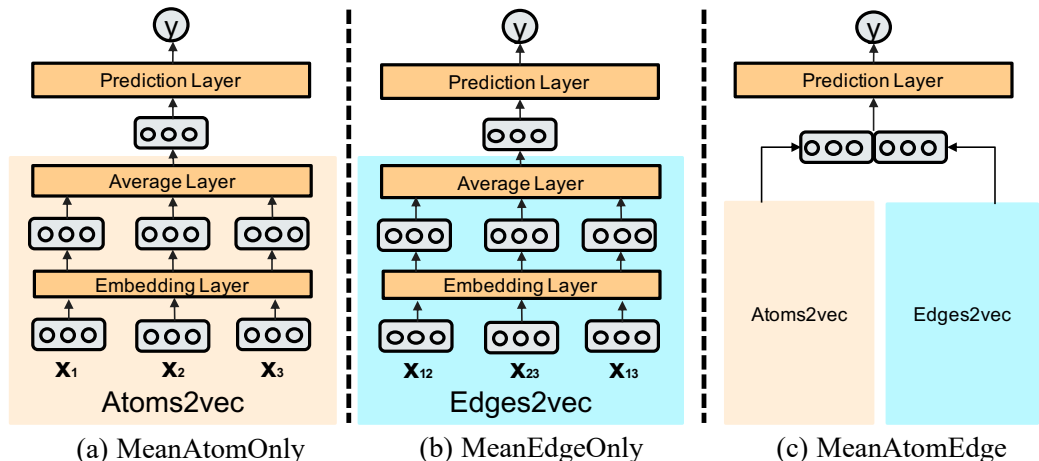


Figure 6.1: Architectures of the proposed neural networks.

metric attributes, e.g., 0 if it is Fe otherwise 1; elemental property statistics, e.g., average atomic number, average period number; electronic structure attributes, e.g., average  $s/p/d/f$  configuration on the valence band; and ionic compound attributes, e.g., electronegativity measurement. Similar features have also been proposed in *Faber et al. (2016)*; *Meredig et al. (2014)* with different data sets. 6 features are used to predict  $E_f$  of all possible  $ABC_2D_6$  compounds for new material discovery *Faber et al. (2016)*. The same task is done with 135 features for all possible materials in ternaries *Meredig et al. (2014)*. The number of such kind of molecular features can easily reach to billions by purely linear and non-linear combination of atomic attributes *Ramprasad et al. (2017)*. Another kind of features take molecular structures into consideration, including Extended-Connectivity Fingerprints (ECFP) *Rogers and Hahn (2010)*, graphlet kernels *Shervashidze et al. (2009,0)*, bag of substructures *Leslie et al. (2001)*; *Ramon and Gärtner (2003)*. Readers are recommended to find details in the original papers. We introduce the Coulomb Matrix (CM), proposed by *Rupp et al. (2012)*; *Montavon et al. (2013)* as an example of such kind of features. CM borrows idea from ab initio electronic structure calculations. Nuclear charges of atoms  $\mathbf{Z}$  and the corresponding Cartesian coordinates  $\mathbf{R}$  are used to create CM. It is defined below, encoding the atomic self-energies and internuclear coulomb repulsion

operator.

$$CM_{i,j} = \begin{cases} 0.5 * \mathbf{Z}_i, i = j \\ \frac{\mathbf{Z}_i \mathbf{Z}_j}{\|\mathbf{R}_i - \mathbf{R}_j\|^2}, i \neq j \end{cases} \quad (6.2-1)$$

Concatenating the values in CM creates a vector representation of a molecule.

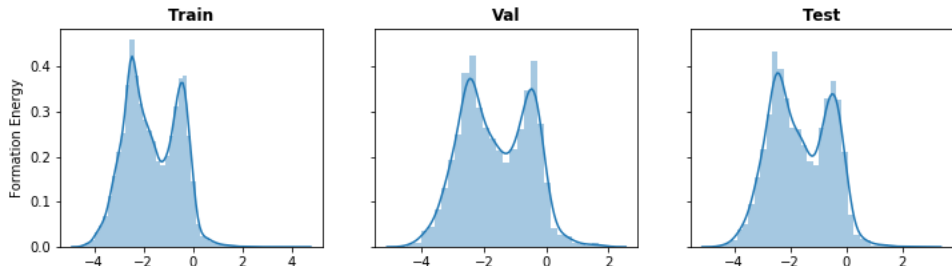
Popular algorithms that deal with vector representations to learn  $f(\mathbf{x}; \mathbf{W}) \rightarrow y$  have been tried out including but not limited to lasso, elasticnet, random forests, support vector machines, kernel ridge regression *Rupp et al. (2012)*; *Blum and Raymond (2009)*; *Montavon et al. (2013)*; *Ward et al. (2016)*; *Faber et al. (2016)*; *Meredig et al. (2014)*.

### 6.2.2 Neural networks

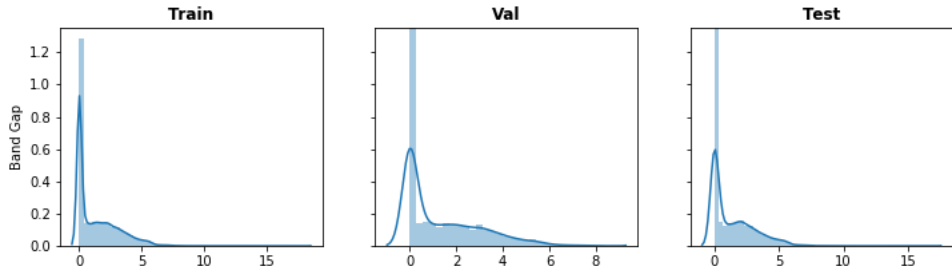
Molecules are naturally treated as attributed graphs. We denote a molecule graph as  $G = (V, E, \mathbf{X}_V, \mathbf{X}_E)$ , where  $V$  and  $E$  contain atoms and bonding edges in a unit-cell of a molecule. We note that the edges can be actual bonds in molecules (e.g. XXX), or virtual edges based on closeness between two atoms.  $\mathbf{X}_V = \{\mathbf{x}_v \in R^p | v \in V\}$  contains the attribute vectors associated with atoms, and  $\mathbf{X}_E = \{\mathbf{x}_{vv'} \in R^q | v \in V, v' \in V\}$  contains the attribute vectors associated with edges. There is no need to spend much effort on hand-crafting atom or edge attributes. We introduce the atom and edge attributes in Section 6.3.

The emergence of deep neural networks makes it possible to learn the mapping directly from attributed graphs  $f(G, \mathbf{X}_V, \mathbf{X}_E; \mathbf{W}) \rightarrow y$ . The underlying regularities in molecules are supposed to be revealed in a layer-wise fashion. ANI-1 *Smith et al. (2017)* is an architecture proposed in 2017 to predict properties of organic molecules. It runs fully connected layers on each atom in the molecule, obtaining atomic energy predictions; then atomic predictions are averaged for the molecular formation energy prediction. DTNN *Schütt et al. (2017b)* utilizes the many-body Hamiltonian concept for the construction of the DTNN architecture. Graph convolutional neural networks (GCNN) *Schütt et al. (2017a)*; *Gilmer et al. (2017)*; *Li et al. (2017)* are designed based on message passing mechanism from the graph theory. Without loss of generality, GCNNs passes messages among atoms through

the neighborhood relationships described by edges. After several message passing steps, predictions from each atom are aggregated to form a molecular prediction of a certain property.



(a)  $E_f$  distribution in train, val, test set.



(b)  $E_g$  distribution in train, val, test set.

Figure 6.2: Distributions of formation energy and band gap.

### 6.3 Methodology

Our proposed methods belong to neural networks. We treat each molecule as an attributed graph  $G = (V, E, \mathbf{X}_V, \mathbf{X}_E)$  as explained before. In this paper, we only use atomic number of the  $v$ th atom to create a one-hot vector  $\mathbf{x}_v$ , i.e., 1 on the  $j$ th position if the atomic number equals to  $j$  and 0 on all other positions. To construct edge attribute vector  $\mathbf{x}_{vv'}$ , we concatenate the one-hot vectors  $\mathbf{x}_v$  and  $\mathbf{x}_{v'}$ , and the euclidean distance between atom  $v$  and  $v'$ .

We propose three NN architectures learning from such graphs. The **MeanAtomOnly**, **MeanEdgeOnly** and **MeanAtomEdge** algorithms are illustrated in Figure 6.1.

The major components of the algorithms are three types of layers (Embedding, Average,

Prediction), each of which define an operation on the outputs from their previous layer.

The Embedding Layer maps a vector  $\mathbf{x}$  to another vector  $\mathbf{x}'$  with  $\mathbf{x}' = g(W\mathbf{x})$ , where  $g$  is an activation function such as sigmoid, tanh, etc. The Average Layer calculates the average vector given input vectors followed by a nonlinear transformation of the average vector, i.e.,  $\mathbf{x}' = g(\frac{1}{n} \sum_{i=1}^n \mathbf{x}_i)$ . The Prediction Layer maps a vector to the output, i.e.,  $y = g(W\mathbf{x})$ .

In the **MeanAtomOnly**, we only utilize  $\mathbf{X}_V$  to predict the material properties; in **MeanEdgeOnly**, edge attributes  $\mathbf{X}_E$  are only utilized to predict properties; while in **MeanAtomEdge**, both  $\mathbf{X}_V$  and  $\mathbf{X}_E$  are utilized. The output vectors after the two Average Layers from **MeanAtomOnly** and **MeanEdgeOnly** are concatenated to predict  $y$ .

## 6.4 Experiments

In this paper, we comprehensively evaluate algorithms for  $E_f$  and  $E_g$  prediction without any constrain on material structures.

### 6.4.1 Dataset

The dataset we use are 69,640 materials downloaded from Material Projects that have a  $E_f$  measurement available. The structures and compositions in one unit cell of materials are extracted. Two properties are given for each material,  $E_f$  per atom (eV/atom), which is the calculated formation energy from the elements normalized to per atom in the unit cell, and the calculated  $E_g$  (eV). The distributions of the two properties are showing in 6.2.

The largest unit cell contains 296 atoms, while the smallest unit cell has only 1 atom in it. The average number of atoms per unit cell is 22. The collection spans a large range from metals, semi-conductors to insulators.

### 6.4.2 Experimental setup

**Data splitting** We split the dataset into train, validation and test. The number of materials in the three sets are 60,000, 4500 and 5140, respectively. Training set is used to train models, while validation set is used for tuning hyperparameters, and test set is used for evaluation of models.

**Models in comparison** We compare 6 feature-driven algorithms and 4 neural networks. The 6 feature-driven algorithms are: **Lasso**, **ElasticNet**, Random Forest (**RF**), Rotation Forest (**REFT**), Gradient Boosting Regression Tree (**GBRT**), **XGBRegressor**. The 4 neural networks contain the 3 simple architectures proposed by us (**MeanAtomOnly**, **MeanEdgeOnly**, **MeanAtomEdge**) and 1 state-of-art architecture **SchNet** *Schiütt et al. (2017a)* based on message passing mechanism.

The detailed architecture of **SchNet** can be found in the original paper *Schiütt et al. (2017a)*. Simply speaking, at layer  $t$ , atomic feature vector is reconstructed using the vectors in the previous layer with the formula:  $\mathbf{x}_v^t = \mathbf{x}_v^{t-1} + \frac{1}{|\mathcal{N}(v)|} \sum_{v' \in \mathcal{N}(v)} M_t(\mathbf{x}_{v'}^{t-1}, \mathbf{x}_{vv'})$ , where  $\mathcal{N}(v)$  contains the neighbors of atom  $v$  in the edge set  $E^m$  and  $M_t$  is the message passing function that maps neighborhood influences to  $v$ . The final prediction is averaged from each atom,  $y = \frac{1}{|V|} \sum_{v \in V} y_v$  and  $y_v = R(\mathbf{x}_v^T)$  is a readout neural network which generates a real number estimation for the  $i$ th atom after  $T$  message passing steps.

**Molecular representation** For feature-driven algorithms, we use the same 146 compound features manually crafted from the composition of materials as in *Meredig et al. (2014)*. The final material representation contains fractions of elements, average atomic mass, average radii, electronegativity, s/p/d/f electron configurations and so on.

For the neural networks in our comparison, we use the same attributed graph representation as described in Section 6.3. The resulting atom features is with length 108. The edge features vector is with length 217, which is the concatenation of the one-hot vectors of the

two atoms connected by the edge plus the length of the edge.

**Hyperparameters** For each machine learning algorithms, the hyperparameters with the best validation performance are selected. For **Lasso** and **ElasticNet**, we tune the penalty coefficients of  $l_1$  and  $l_2$  norm. For the tree-based algorithms (**Random Forest**, **GBRT**, **Rotation Forest**, **XGBRegressor**), the *number of estimators*, *maximum tree depth*, *minimum number of samples to split a tree*, *minimum number of samples in leafs* are tuned. For the **SchNet** model, we use the best hyperparameters from Schütt *et al.* (2017a), i.e, we run  $T = 6$  message passing steps; the initial learning rate is 0.0001 and an exponential learning rate decay with ratio 0.96 every 100,000 steps is used; the dimension of the hidden representation is 64. The hyperparameters for **MeanAtomOnly**, **MeanEdgeOnly** and **MeanAtomEdge** keep the same as **SchNet** except that the initial learning rate is 0.001. We train all deep models with the ADAM optimizer with 32 materials per mini-batch.

## 6.5 Results

### 6.5.1 Qualitative results

We collect the best mean absolute error (MAE) on validation set and the final test MAE in Table 6.1 and Table 6.2.

The results demonstrate a comprehensive comparison across various representations and machine learning algorithms on two regression tasks for unconstrained materials. We have several significant observations from the table:

1. On both of the two tasks, deep learning models taking attributed graphs as inputs outperform the conventional models taking regular representations as inputs.
2. The simplest **MeanAtomOnly** model, which only takes the atomic numbers as inputs, outperform the best conventional model (**XGBRegressor** with manually crafted representations) on both two tasks.

Table 6.1: Regression results of 10 models on formation energy prediction.

Models	Formation Energy	
	Best Val MAE	Test MAE
<b>Lasso</b>	0.3117	0.3001
<b>ElasticNet</b>	0.3117	0.3003
<b>RF</b>	0.1594	0.1528
<b>REFT</b>	0.1333	0.1279
<b>GBRT</b>	0.1040	0.1003
<b>XGBRegressor</b>	0.1003	0.0968
<b>MeanAtomOnly</b>	0.0841	0.0820
<b>MeanEdgeOnly</b>	0.0944	0.0910
<b>MeanAtomEdge</b>	0.0871	0.0842
<b>SchNet</b>	<b>0.0358</b>	<b>0.0375</b>

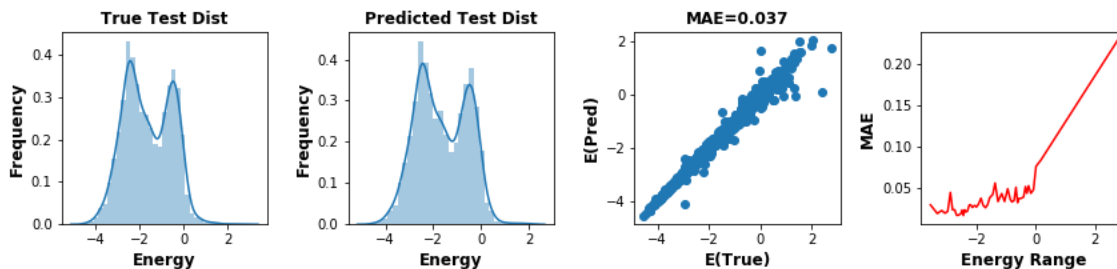
Table 6.2: Regression results of 10 models on band gap prediction.

Models	Band Gap	
	Best Val MAE	Test MAE
<b>Lasso</b>	0.9669	0.9566
<b>ElasticNet</b>	0.9669	0.9566
<b>RF</b>	0.5574	0.5357
<b>REFT</b>	0.5141	0.4961
<b>GBRT</b>	0.4354	0.4282
<b>XGBRegressor</b>	0.4298	0.4241
<b>MeanAtomOnly</b>	0.4110	<b>0.4077</b>
<b>MeanEdgeOnly</b>	0.4916	0.4853
<b>MeanAtomEdge</b>	0.4466	0.4372
<b>SchNet</b>	<b>0.3922</b>	<b>0.4063</b>

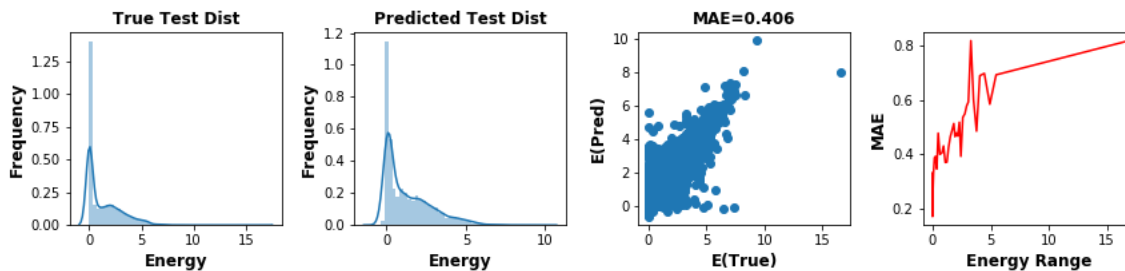
3. On formation energy prediction task, a message passing architecture (**SchNet**) can have way more accurate predictions than a simple architecture (**MeanAtomOnly**); while on band gap prediction task, the well-designed architecture fails to be more accurate. It leads to a conclusion that there is no general deep learning architecture that is effective in all tasks.

### 6.5.2 Error analysis

To better understand the error made by the most accurate deep learning model (**SchNet**), we plot the scatter plots between the predictions and the ground true calculations on the test



(a) Formation energy predictions and errors in test set.



(b) Band gap predictions and errors in test set.

Figure 6.3: Subfigures from left to right are: ground true property distributions; predicted property distributions; scatter plot of predictions vs. true values; MAE by true measurement range.

set. We also do the following error analysis: we first sort the test data in ascending order based on the property measurement, then we calculate the MAE for every 100 materials in order. By this means, it helps us to identify the range where the most error is made. The errors are illustrated in Figure 6.3. For the formation energy task, the errors are accumulated on materials with large energy. However, for the band gap task, less errors are made for metals. MAEs are almost spanned in the whole range of materials.

## 6.6 Conclusions

Graph neural networks have been shown great success in the domain of drug design and material sciences, where organic molecules and crystal structures of materials are represented as attributed graphs. A deep learning architecture that is capable of learning from graph nodes and graph edges is crucial for property estimation of molecules. In this dissertation, We propose a simple graph representation for molecules and three neural network

architectures that is able to directly learn predictive functions from graphs. We discover that, in true graph networks are superior than feature-driven algorithms for formation energy prediction. However, the superiority can not be reproduced on band gap prediction. We also discovered that our proposed simple shallow neural networks perform comparably with the state-of-the-art deep neural networks. We propose to incorporate the motif structures in our future work. Motifs picture the local environment of a crystal, which has been demonstrated critical for band gap.

## BIBLIOGRAPHY

- Abel, F., C. Hauff, G.-J. Houben, R. Stronkman, and K. Tao (2012), Twitcident: fighting fire with information from social web streams, in *Proceedings of the 21st International Conference on World Wide Web*, pp. 305–308, ACM.
- Agirre, E., E. Alfonseca, K. Hall, J. Kravalova, M. Paşca, and A. Soroa (2009), A study on similarity and relatedness using distributional and wordnet-based approaches, in *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pp. 19–27, Association for Computational Linguistics.
- Alashkar, T., S. Jiang, S. Wang, and Y. Fu (2017), Examples-rules guided deep neural network for makeup recommendation., in *AAAI*, pp. 941–947.
- Ashktorab, Z., C. Brown, M. Nandi, and A. Culotta (2014), Tweedr: Mining twitter to inform disaster response, *Proc. of ISCRAM*.
- Banko, M., M. J. Cafarella, S. Soderland, M. Broadhead, and O. Etzioni (2007), Open information extraction from the web., in *IJCAI*, vol. 7, pp. 2670–2676.
- Bartoli, A., G. Davanzo, A. De Lorenzo, M. Mauri, E. Medvet, and E. Sorio (2012), Automatic generation of regular expressions from examples with genetic programming, in *Proceedings of the 14th annual conference companion on Genetic and evolutionary computation*, pp. 1477–1478.
- Bartoli, A., G. Davanzo, A. De Lorenzo, E. Medvet, and E. Sorio (2014), Automatic synthesis of regular expressions from examples, *Computer*, 47(12), 72–80.
- Bartoli, A., A. De Lorenzo, E. Medvet, and F. Tarlao (2016), Inference of regular expressions for text extraction from examples, *IEEE Transactions on Knowledge and Data Engineering*, 28(5), 1217–1230.
- Bartoli, A., A. De Lorenzo, E. Medvet, and F. Tarlao (2018), Active learning of regular expressions for entity extraction, *IEEE transactions on cybernetics*, 48(3), 1067–1080.
- Bergstra, J., and Y. Bengio (2012), Random search for hyper-parameter optimization, *Journal of Machine Learning Research*, 13(Feb), 281–305.
- Blum, L. C., and J.-L. Reymond (2009), 970 million druglike small molecules for virtual screening in the chemical universe database GDB-13, *J. Am. Chem. Soc.*, 131, 8732.

- Bojanowski, P., E. Grave, A. Joulin, and T. Mikolov (2016), Enriching word vectors with subword information, *arXiv preprint arXiv:1607.04606*.
- Botha, J., and P. Blunsom (2014), Compositional morphology for word representations and language modelling, in *International Conference on Machine Learning*, pp. 1899–1907.
- Brauer, F., R. Rieger, A. Mocan, and W. M. Barczynski (2011), Enabling information extraction by inference of regular expressions from sample entities, in *Proceedings of the 20th ACM international conference on Information and knowledge management*, pp. 1285–1294, ACM.
- Brown, P. F., P. V. Desouza, R. L. Mercer, V. J. D. Pietra, and J. C. Lai (1992), Class-based n-gram models of natural language, *Computational linguistics*, 18(4), 467–479.
- Bruni, E., N.-K. Tran, and M. Baroni (2014), Multimodal distributional semantics, *Journal of Artificial Intelligence Research*, 49, 1–47.
- Bui, D. D. A., and Q. Zeng-Treitler (2014), Learning regular expressions for clinical text classification, *Journal of the American Medical Informatics Association*, 21(5), 850–857.
- Cetinkaya, A. (2007), Regular expression generation through grammatical evolution, in *Proceedings of the 9th annual conference companion on Genetic and evolutionary computation*, pp. 2643–2646, ACM.
- Chang, C.-H., M. Kayed, M. R. Girgis, and K. F. Shaalan (2006), A survey of web information extraction systems, *IEEE Transactions on Knowledge and Data Engineering*, 18(10), 1411–1428.
- Chen, Y., T. A. Lasko, Q. Mei, J. C. Denny, and H. Xu (2015), A study of active learning methods for named entity recognition in clinical text, *Journal of biomedical informatics*, 58, 11–18.
- Chiu, J. P., and E. Nichols (2015), Named entity recognition with bidirectional lstm-cnns, *arXiv preprint arXiv:1511.08308*.
- Cochran, R. A., L. D’Antoni, B. Livshits, D. Molnar, and M. Veanes (2015), Program boosting: Program synthesis via crowd-sourcing, in *ACM SIGPLAN Notices*, vol. 50, pp. 677–688, ACM.
- Cotterell, R., and H. Schütze (2015), Morphological word-embeddings, in *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 1287–1292.
- Denis, F. (2001), Learning regular languages from simple positive examples, *Machine Learning*, 44(1-2), 37–66.
- Devlin, J., M.-W. Chang, K. Lee, and K. Toutanova (2018), Bert: Pre-training of deep bidirectional transformers for language understanding, *arXiv preprint arXiv:1810.04805*.

- Donahue, J., Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, and T. Darrell (2014), Decaf: A deep convolutional activation feature for generic visual recognition, in *International conference on machine learning*, pp. 647–655.
- Elias, P., M. Birch, et al. (2010), Soc2010: revision of the standard occupational classification, *Economic & Labour Market Review*, 4(7), 48–55.
- Erhan, D., Y. Bengio, A. Courville, P.-A. Manzagol, P. Vincent, and S. Bengio (2010), Why does unsupervised pre-training help deep learning?, *Journal of Machine Learning Research*, 11(Feb), 625–660.
- Etzioni, O., M. Cafarella, D. Downey, A.-M. Popescu, T. Shaked, S. Soderland, D. S. Weld, and A. Yates (2005), Unsupervised named-entity extraction from the web: An experimental study, *Artificial intelligence*, 165(1), 91–134.
- Faber, F. A., A. Lindmaa, O. A. Von Lilienfeld, and R. Armiento (2016), Machine learning energies of 2 million elpasolite (a b c 2 d 6) crystals, *Physical review letters*, 117(13), 135,502.
- Felbo, B., A. Mislove, A. Sjøgaard, I. Rahwan, and S. Lehmann (2017), Using millions of emoji occurrences to learn any-domain representations for detecting sentiment, emotion and sarcasm, *arXiv preprint arXiv:1708.00524*.
- Fernau, H. (2009), Algorithms for learning regular expressions from positive data, *Information and Computation*, 207(4), 521–541.
- Finkel, J. R., and C. D. Manning (2009), Nested named entity recognition, in *EMNLP*, pp. 141–150, Association for Computational Linguistics.
- Fries, J., S. Wu, A. Ratner, and C. Ré (2017), Swellshark: A generative model for biomedical named entity recognition without labeled data, *arXiv preprint arXiv:1704.06360*.
- Giannakopoulos, A., C. Musat, A. Hossmann, and M. Baeriswyl (2017), Unsupervised aspect term extraction with b-lstm & crf using automatically labelled datasets, *arXiv preprint arXiv:1709.05094*.
- Gilmer, J., S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl (2017), Neural message passing for quantum chemistry, *arXiv preprint arXiv:1704.01212*.
- Gimpel, K., et al. (2011), Part-of-speech tagging for twitter: Annotation, features, and experiments, in *ACL*, pp. 42–47, Association for Computational Linguistics.
- Go, A., R. Bhayani, and L. Huang (2009), Twitter sentiment classification using distant supervision, *CS224N Project Report, Stanford*, 1(12), 2009.
- Grbovic, M., N. Djuric, V. Radosavljevic, F. Silvestri, and N. Bhamidipati (2015), Context- and content-aware embeddings for query rewriting in sponsored search, in *Proceedings of the 38th international ACM SIGIR conference on research and development in information retrieval*, pp. 383–392, ACM.

- Grbovic, M., N. Djuric, V. Radosavljevic, F. Silvestri, R. Baeza-Yates, A. Feng, E. Ordentlich, L. Yang, and G. Owens (2016), Scalable semantic matching of queries to ads in sponsored search advertising, in *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval*, pp. 375–384, ACM.
- Hill, F., R. Reichart, and A. Korhonen (2015), Simlex-999: Evaluating semantic models with (genuine) similarity estimation, *Computational Linguistics*, 41(4), 665–695.
- Hu, Z., X. Ma, Z. Liu, E. Hovy, and E. Xing (2016), Harnessing deep neural networks with logic rules, *arXiv preprint arXiv:1603.06318*.
- Huang, Z., W. Xu, and K. Yu (2015), Bidirectional lstm-crf models for sequence tagging, *arXiv preprint arXiv:1508.01991*.
- Imran, M., and C. Castillo (2015), Towards a data-driven approach to identify crisis-related topics in social media streams, in *Proceedings of the 24th International Conference on World Wide Web*, pp. 1205–1210, ACM.
- Imran, M., C. Castillo, J. Lucas, P. Meier, and S. Vieweg (2014), Aidr: Artificial intelligence for disaster response, in *ACM WWW*, pp. 159–162.
- Imran, M., C. Castillo, F. Diaz, and S. Vieweg (2015), Processing social media messages in mass emergency: A survey, *ACM Computing Surveys (CSUR)*, 47(4), 67.
- Imran, M., P. Mitra, and C. Castillo (2016a), Twitter as a lifeline: Human-annotated twitter corpora for nlp of crisis-related messages, *arXiv preprint arXiv:1605.05894*.
- Imran, M., P. Mitra, and J. Srivastava (2016b), Cross-language domain adaptation for classifying crisis-related short messages, *arXiv preprint arXiv:1602.05388*.
- Inc, G. O. (2014), Gurobi optimizer reference manual, <http://www.gurobi.com>.
- IQAndreas (2014), What is meant by “now you have two problems”?
- Javed, F., Q. Luo, M. McNair, F. Jacob, M. Zhao, and T. S. Kang (2015), Carotene: A job title classification system for the online recruitment domain, in *Big Data Computing Service and Applications (BigDataService)*, 2015 IEEE First International Conference on, pp. 286–293, IEEE.
- Kim, Y. (2014), Convolutional neural networks for sentence classification, *arXiv preprint arXiv:1408.5882*.
- Kim, Y., Y. Jernite, D. Sontag, and A. M. Rush (2016), Character-aware neural language models., in *AAAI*, pp. 2741–2749.
- Kim, Y., K.-M. Kim, J.-M. Lee, and S. Lee (2018), Learning to generate word representations using subword information, in *Proceedings of the 27th International Conference on Computational Linguistics*, pp. 2551–2561, Association for Computational Linguistics, Santa Fe, New Mexico, USA.

- Krishnakumar, A. (2007), Active learning literature survey, *Tech. rep.*, Technical reports, University of California, Santa Cruz. 42.
- Lai, S., L. Xu, K. Liu, and J. Zhao (2015), Recurrent convolutional neural networks for text classification., in *AAAI*, vol. 333, pp. 2267–2273.
- Lample, G., M. Ballesteros, S. Subramanian, K. Kawakami, and C. Dyer (2016), Neural architectures for named entity recognition, *arXiv preprint arXiv:1603.01360*.
- Le, Q., and T. Mikolov (2014), Distributed representations of sentences and documents, in *International Conference on Machine Learning*, pp. 1188–1196.
- Leslie, C., E. Eskin, and W. S. Noble (2001), The spectrum kernel: A string kernel for svm protein classification, in *Biocomputing 2002*, pp. 564–575, World Scientific.
- Li, Y., R. Krishnamurthy, S. Raghavan, S. Vaithyanathan, and H. Jagadish (2008), Regular expression learning for information extraction, in *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pp. 21–30, Association for Computational Linguistics.
- Li, Z., X. Ma, and H. Xin (2017), Feature engineering of machine-learning chemisorption models for catalyst design, *Catalysis today*, 280, 232–238.
- Lin, Z., M. Feng, C. N. d. Santos, M. Yu, B. Xiang, B. Zhou, and Y. Bengio (2017), A structured self-attentive sentence embedding, *arXiv preprint arXiv:1703.03130*.
- Liu, X., J. Gao, X. He, L. Deng, K. Duh, and Y.-Y. Wang (2015), Representation learning using multi-task deep neural networks for semantic classification and information retrieval.
- Locascio, N., K. Narasimhan, E. DeLeon, N. Kushman, and R. Barzilay (2016), Neural generation of regular expressions from natural language with minimal domain knowledge, *arXiv preprint arXiv:1608.03000*.
- Luo, B., Y. Feng, Z. Wang, S. Huang, R. Yan, and D. Zhao (2018), Marrying up regular expressions with neural networks: A case study for spoken language understanding, *arXiv preprint arXiv:1805.05588*.
- Luong, M.-T., and C. D. Manning (2016), Achieving open vocabulary neural machine translation with hybrid word-character models, *arXiv preprint arXiv:1604.00788*.
- Luong, T., R. Socher, and C. Manning (2013), Better word representations with recursive neural networks for morphology, in *Proceedings of the Seventeenth Conference on Computational Natural Language Learning*, pp. 104–113.
- Ma, X., and E. Hovy (2016), End-to-end sequence labeling via bi-directional lstm-cnns-crf, *arXiv preprint arXiv:1603.01354*.

- Maas, A. L., R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts (2011), Learning word vectors for sentiment analysis, in *Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies-volume 1*, pp. 142–150, Association for Computational Linguistics.
- Mahajan, D., R. Girshick, V. Ramanathan, K. He, M. Paluri, Y. Li, A. Bharambe, and L. van der Maaten (2018), Exploring the limits of weakly supervised pretraining, *arXiv preprint arXiv:1805.00932*.
- Meredig, B., A. Agrawal, S. Kirklin, J. E. Saal, J. Doak, A. Thompson, K. Zhang, A. Choudhary, and C. Wolverton (2014), Combinatorial screening for new materials in unconstrained composition space with machine learning, *Physical Review B*, 89(9), 094,104.
- Mikolov, T., I. Sutskever, K. Chen, G. S. Corrado, and J. Dean (2013), Distributed representations of words and phrases and their compositionality, in *Advances in neural information processing systems*, pp. 3111–3119.
- Millner, R. (2008), Four regular expressions to check email addresses.
- Montavon, G., M. Rupp, V. Gobre, A. Vazquez-Mayagoitia, K. Hansen, A. Tkatchenko, K.-R. Müller, and O. A. Von Lilienfeld (2013), Machine learning of molecular electronic properties in chemical compound space, *New Journal of Physics*, 15(9), 095,003.
- Murtaugh, M. A., B. S. Gibson, D. Redd, and Q. Zeng-Treitler (2015), Regular expression-based learning to extract bodyweight values from clinical notes, *Journal of Biomedical Informatics*, 54, 186 – 190.
- Murthy, K., P. Deepak, and P. M. Deshpande (2012), Improving recall of regular expressions for information extraction, in *International Conference on Web Information Systems Engineering*, pp. 455–467, Springer.
- Nadeau, D., and S. Sekine (2007), A survey of named entity recognition and classification, *Linguisticae Investigationes*, 30(1), 3–26.
- Neculoiu, P., M. Versteegh, and M. Rotaru (2016), Learning text similarity with siamese recurrent networks, in *Proceedings of the 1st Workshop on Representation Learning for NLP*, pp. 148–157.
- Nguyen, H. T., and A. Smeulders (2004), Active learning using pre-clustering, in *ICML*, p. 79, ACM.
- Olteanu, A., C. Castillo, F. Diaz, and S. Vieweg (2014), Crisislex: A lexicon for collecting and filtering microblogged communications in crises., in *ICWSM*.
- Pennington, J., R. Socher, and C. Manning (2014), Glove: Global vectors for word representation, in *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pp. 1532–1543.



- Santos, C. D., and B. Zadrozny (2014), Learning character-level representations for part-of-speech tagging, in *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pp. 1818–1826.
- Schick, T., and H. Schütze (2019), Attentive mimicking: Better word embeddings by attending to informative contexts, *arXiv preprint arXiv:1904.01617*.
- Schütt, K., P.-J. Kindermans, H. E. S. Felix, S. Chmiela, A. Tkatchenko, and K.-R. Müller (2017a), Schnet: A continuous-filter convolutional neural network for modeling quantum interactions, in *Advances in Neural Information Processing Systems*, pp. 991–1001.
- Schütt, K. T., F. Arbabzadah, S. Chmiela, K. R. Müller, and A. Tkatchenko (2017b), Quantum-chemical insights from deep tensor neural networks, *Nature communications*, 8, 13,890.
- Shang, J., L. Liu, X. Ren, X. Gu, T. Ren, and J. Han (2018), Learning named entity tagger using domain-specific dictionary, *arXiv preprint arXiv:1809.03599*.
- Shen, W., J. Wang, and J. Han (2015), Entity linking with a knowledge base: Issues, techniques, and solutions, *IEEE Transactions on Knowledge and Data Engineering*, 27(2), 443–460.
- Shen, Y., H. Yun, Z. C. Lipton, Y. Kronrod, and A. Anandkumar (2017), Deep active learning for named entity recognition, *arXiv preprint arXiv:1707.05928*.
- Shervashidze, N., S. Vishwanathan, T. Petri, K. Mehlhorn, and K. Borgwardt (2009), Efficient graphlet kernels for large graph comparison, in *Artificial Intelligence and Statistics*, pp. 488–495.
- Shervashidze, N., P. Schweitzer, E. J. v. Leeuwen, K. Mehlhorn, and K. M. Borgwardt (2011), Weisfeiler-lehman graph kernels, *Journal of Machine Learning Research*, 12(Sep), 2539–2561.
- Simoës, S., P. Deepak, M. Sairamesh, D. Khemani, and S. Mehta (2018), Content and context: Two-pronged bootstrapped learning for regex-formatted entity extraction., in *AAAI*.
- Smith, J. S., O. Isayev, and A. E. Roitberg (2017), Ani-1: an extensible neural network potential with dft accuracy at force field computational cost, *Chemical science*, 8(4), 3192–3203.
- Socher, R., A. Perelygin, J. Wu, J. Chuang, C. D. Manning, A. Ng, and C. Potts (2013), Recursive deep models for semantic compositionality over a sentiment treebank, in *Proceedings of the 2013 conference on empirical methods in natural language processing*, pp. 1631–1642.
- Temnikova, I., C. Castillo, and S. Vieweg (2015), Emterms 1. 0: a terminological resource for crisis tweets, in *ISCRAM 2015 proceedings of the 12th international conference on information systems for crisis response and management*.

- Wang, H., E. Hovy, and M. Dredze (2015), The hurricane sandy twitter corpus, in *Workshops at the twenty-ninth AAAI conference on artificial intelligence*.
- Ward, L., A. Agrawal, A. Choudhary, and C. Wolverton (2016), A general-purpose machine learning framework for predicting properties of inorganic materials, *npj Computational Materials*, 2, 16,028.
- Wieting, J., M. Bansal, K. Gimpel, and K. Livescu (2016), Charagram: Embedding words and sentences via character n-grams, *arXiv preprint arXiv:1607.02789*.
- Wu, L. Y., A. Fisch, S. Chopra, K. Adams, A. Bordes, and J. Weston (2018), Starspace: Embed all the things!, in *Thirty-Second AAAI Conference on Artificial Intelligence*.
- Zeng, Q. T., S. Goryachev, S. Weiss, M. Sordo, S. N. Murphy, and R. Lazarus (2006), Extracting principal diagnosis, co-morbidity and smoking status for asthma research: evaluation of a natural language processing system, *BMC Medical Informatics and Decision Making*, 6(1), 30.
- Zhang, S., and S. Vucetic (2016), Semi-supervised discovery of informative tweets during the emerging disasters, *arXiv preprint arXiv:1610.03750*.
- Zhang, S., L. He, S. Vucetic, and E. Dragut (2018a), Regular expression guided entity mention mining from noisy web data, in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 1991–2000.
- Zhang, S., A. Pal, K. Kant, and S. Vucetic (2018b), Enhancing disaster situational awareness via automated summary dissemination of social media content, in *2018 IEEE Global Communications Conference (GLOBECOM)*, pp. 1–7, IEEE.
- Zhang, S., L. He, S. Vucetic, and E. Dragut (2019), How to invest my time: Lessons from human-in-the-loop entity extraction, acm sigkdd conference on knowledge discovery and data mining, in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ACM.
- Zhang, X., J. Zhao, and Y. LeCun (2015), Character-level convolutional networks for text classification, in *Advances in neural information processing systems*, pp. 649–657.
- Zhao, J., S. Mudgal, and Y. Liang (2018), Generalizing word embeddings using bag of subwords, *arXiv preprint arXiv:1809.04259*.