UNIVERSITY OF GOTHENBURG

DEPARTMENT OF PHILOSOPHY,
LINGUISTICS AND THEORY OF SCIENCE

EXPLOIT UNLABELED DATA WITH LANGUAGE MODEL FOR TEXT CLASSIFICATION

Comparison of four unsupervised learning models

Sung-Min Yang

Master's Thesis:	30 credits
Programme:	Master's Programme in Language Technology
Level:	Advanced level
Semester and year:	Spring, 2018
Supervisor:	Asad Sayeed, Aron Lagerberg
Examiner:	Lars Borin
Report number:	
Keywords:	Text classification, Semi-supervised learning, Unsupervised learning, Transfer learning, Natural Language Processing.

Abstract

Within a situation where Semi-Supervised Learning (SSL) is available to exploit unlabeled data, this paper shows that Language Model (LM) outperforms the three models in text classification, which three models are based on Term-Frequency Inverse Document Frequency (Tf-idf) and two pre-trained word vectors. The experimental results show that the LM outperforms the other three unsupervised learning models whether the task is easy or difficult, which the difficult task consists of imbalanced data.

To investigate not only how the LM outperforms the other models but also how to maximize the performance of the LM in a small quantity of labeled data, this paper suggests two techniques to improve the performance of the LM in neural networks: (1) obtaining information from the neural network layers and (2) employing a proper evaluation for trained neural networks models.

Finally, this paper explores the various scenarios where SSL is not available, but only Transfer Learning (TL) is accessible to exploit unlabeled data. With two types of Self-Taught Learning and Multi-Tasks in TL, the results of the experiments show that exploiting dataset which has wider domain benefits the performance of the LM.

Acknowledgements

This paper has been done as a final project of the Master's Programme in Language Technology at University of Gothenburg. As Axel Adler Foundation has been financially supported me during entire study time, for the first, I am grateful to Axel Adler for all the supports. Furthermore, it should be mentioned the fact that Ernst Wigforss encouraged Axle Adler to donate money to the society. Thanks to two generous men who valued education and research, I could come to Sweden and continue studying.

It was impossible to start this thesis without supervisor Asad Sayeed. I would like to thank him for guiding me to right direction and supporting me with valuable discussions. Without his encouragements, I could not finish this paper.

There is a no word to express my appreciation to supervisor Aron Lagerberg. If it was not him, this paper would not even exist. I specially thank him for teaching me how to question critically. He inspired me to have a scientific mindset and guided me with his strong mathematical background.

I would like to thank Seal-Software for providing great research environments with a number of GPUs and workstations. Without their supports, it would take much longer time to finish this research. In fact, this study could not be even started.

Thanks to my family who always supports me with unconditional love, I was able to finish this study without any concerns. Finally, but most importantly, I thank my fiancé Nafi, who let me open my eyes wider and have been encouraging me to follow my dream. Without her, I would not be even in Sweden to write this paper in this moment.

Gothenburg, May, 31th, 2018

Sung-Min Yang

Contents

List of Abbreviations

CNN: Convolutional Neural Network

CV: Computer Vision

EM: Expectation-Maximization

LR: Logistic Regression

LM: Language Model

LSTM: Long-short term memory

ML: Machine Learning

MNB: Multinomial Naïve Bayes

MPU: Multi-Positive and unlabeled

MTL: Multi-Tasks Learning

NLM : Neural Language Model

NLP: Natural Language Processing

NN: Neural Networks

NU: Negative Unlabeled

OOV : Out of Vocabulary

PU: Positive and Unlabeled

RNN: Recurrent Neural Network

SSL: Semi-supervised learning

STL: Self-Taught Learning

SVM: Support Vector Machine

SVC: Support Vector Classifier

TC: Text Categorization, Text Classification

TF-IDF: Term frequency-inverse document frequency

TL: Transfer Learning

UL: Unsupervised Learning

1 Introduction

Text classification (TC) is a task to determine whether the text belongs to a particular category. This categorizing task has a long history of more than two hundred years. Specifically, Zechner (2013) mentions that TC is an old problem and studies in TC were done to verify authorship of the works of Shakespeare in the early 1800s. After the World Wide Web (WWW) was created, a massive quantity of online texts became available to the public. By the number of text-based contents has been increased, the importance of TC become one of the crucial tasks for real-world applications. This phenomenon can be found in a recent study by Allahyari et al. (2017) that states "Text classification has been broadly studied in different communities such as data mining, database, Machine Learning (ML) and Information Retrieval (IR*)*, and used in vast number of applications in various domains". By observing the statements by Zechner (2013) and Allahyari et al. (2017), we can assume that text classification has played been playing an important role in from the 1800s up to nowadays.

For conducting text classification tasks, labeled data must be provided to train ML algorithms. It is common that performance of ML models improves by the number of labeled data. However, obtaining a big number of labeled data is often not realistic and inefficient. For example, in specific domains such as legal and medical fields where lawyers and doctors are manually annotating the data, labeling data is not only expensive but also the process is slow. In order to overcome these cost and time issues, exploiting unlabeled data can be employed while building a ML model. Therefore, this paper aims to find a best model that uses both labeled and unlabeled data. Specifically, four different models are observed to find the best model in the experiments within Semi-Supervised Learning (SSL) which is a learning type that uses both labeled and unlabeled data.

When it comes to the Semi-Supervised Learning, there are two ways to exploit unlabeled data. The first way is to extract features from the unlabeled data and the other way is to build a feature extractor with unlabeled data. Since this paper hypothesizes that the second way to exploit unlabeled data will benefit the performance more than the first method, a second way of Language Model (LM) is investigated to reach the best performance among the four models. Furthermore, this paper proposes two additional techniques to maximize the performance of the model based on LM.

1.1 Motivation

Training ML algorithms for classification tasks requires labeled data as a part of the supervised learning procedures. Collecting labeled data always cost money since the process must conducted by human annotators. This cost problem becomes a serious issue when it comes to expensive domains such as legal or medical fields where annotators are lawyer and doctors. In order to overcome this high-cost labeling process, one possible way is exploiting unlabeled data (Nigam et al., 2000). Since unlabeled data was already collected before the labeling process, the amount is always bigger than labeled data. Therefore, if one knows how to extract meaningful information from unlabeled data, the unlabeled data gives help to build models in addition to using the labeled data. Briefly, this study is inspired by the difficulties of collecting labeled data in expensive domains where the unlabeled data is already obtained.

In addition to the cost problem of labeled data, the potential of unlabeled data motivated this work. Since unlabeled data exists as a non-task type, whereas classification dataset is a task type, data of non-task type exists everywhere without any annotations. In other words, collecting unlabeled data is unlimited availability. Therefore, this paper explores various methods to exploit unlabeled data which are both task type and non-task type. Specifically, two studies by Do & Ng (2005) and Raina et al. (2007) have inspired this thesis work, which address that using unlabeled data can improve the performances of ML models.

1.2 Focus of the Thesis

The primary interest of this paper is to investigate the performance of the LM model by comparing with three other unsupervised learning models. Specifically, the three other models are based on the Term frequency-inverse document frequency (Tf-idf) (Jones, 1972, 1973), recent neural network based models Word2Vec (Le & Mikolov, 2014), FastText (Bojanowski et al., 2016) and lately proposed the AWD-LSTM (Averaged Stochastic gradient WeightDrop Long short-term memory) (Merity et al., 2017) language model. Given these four models, controlling the number of labeled data is conducted to inspect how each model performs corresponding to the limited number of labeled data.

The next main focus of this thesis is to discover the best way to implement LM for TC tasks. In order to achieve the best performance in LM based models, finding not only a proper value of hyperparameters is crucial but a method to obtain information from Neural Networks (NN) is also important. These values of the hyperparameters and the methods to obtain the information will be empirically found within supervised learning. In addition, a proper evaluation for the trained NN models will be proposed in this paper.

Finally, this thesis aims to clarify the various types of ML which is defined by different distributions of data between two training stages in ML. Since SSL consists of unsupervised learning for feature extractors and supervised learning for classifiers, two data distributions exist for each stage. These two data distributions produce different types of ML such as Semi-Supervised Learning (SSL) and Transfer Learning (TL). Furthermore, this paper also covers the small branches of TL such as Self-taught Learning and Multi-Tasks Learning to inspect a case where SSL is not available. Therefore, the effectiveness of small branches of TL will be additionally studied for the further experiments.

1.4 Outline

This paper consists of three parts. The first part is background, which explains the previous works, basic knowledge of data distribution, ML algorithms and different types of ML. The next part demonstrates the methodology of the paper which specifies the datasets, preprocessing techniques, hyperparameters and the additional techniques for the experiments. Finally, the third part covers the results of the experiments and the discussion of the results, and then it gives the conclusion of the paper with possible future works.

2 Background

This chapter provides essential knowledge and the related works that support the methodology of this paper. First of all, previous studies and types of data distribution will be explained. The third section then describes the three main unsupervised learning methods which are related to building feature extractors. After understanding the use of unsupervised learning, supervised learning will be described for classification tasks with two types of classifiers. The fourth section illustrates two learning methods of semi-supervised learning and transfer learning which are defined by relations between two training stages in Machine Learning (ML). Finally, the last section describes regularization techniques for Neural Network (NN) models. To understand the general process and elements of Text Classification (TC), it is recommended to read a paper written by Mirończuk & Protasiewicz (2018) along with this paper.

2.1 Previous works

Nigam et al.(2000) addressed how to exploit unlabeled data to improve performance of feature extractors. Despite the fact that the term ‘Semi-Supervised Learning (SSL)’ (Scudder, 1965; Chapelle et al., 2006) was not used in their paper, their research is based on SSL which is main topic of this paper. They have shown how to use unlabeled text to build a better feature extractor by using the Expectation-Maximization (EM) with the Naïve Bayes algorithm. Specifically, a word and the position of the word in documents are used as features to generate document texts as a variant of Language Modeling (Markov, 1953). Nigam et al. (2000) also described how the quantity of unlabeled data affects the performance of the model corresponding to limited labeled data. According to their results which are presented in Figure 1, the less labeled data is trained for a model, the more benefits unlabeled data gives for the model performance. The lines in Figure 1 illustrate the effects of the number of unlabeled data corresponding to the number of labeled data for TC classification. After showing the result in Figure 1, Nigam et al.(2000) concluded that using unlabeled data improves the performance of the model in SSL. To extend their work, this paper focuses on comparing four unsupervised models rather than one model of the EM with the Naïve Bayes algorithm (Nigam et al., 2000). Moreover, this paper will use a fixed number of unlabeled data which is different from using various quantity of unlabeled data (Nigram et al., 2000). The details of four unsupervised models for exploiting unlabeled data are described in the section 2.3 ‘unsupervised learning for feature extractors’.

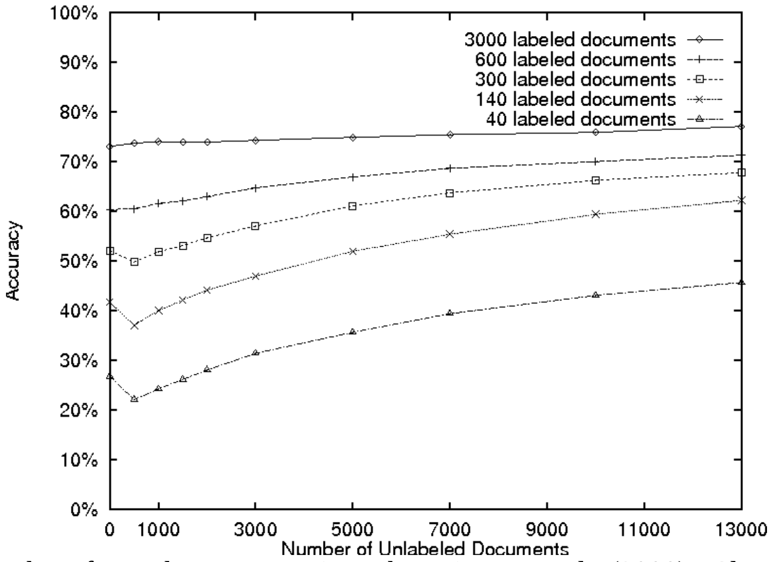


Figure 1: Figure taken from the paper written by Nigam et al. (2000). Classification accuracy corresponding to different numbers of unlabeled and labeled documents. The 20 Newsgroups dataset which is collected by Lang (1995) is used for the experiment.

In addition to SSL, researching in Transfer Learning (TL) (Pratt, 1993; Caruana, 1998) is another main objective of this paper. Specifically, two types of transfer learning will be explored in the experiments, which are Multi-Tasks Learning (MTL) (Caruana, 1998) and Self-Taught Learning (STL) (Raina et al., 2007). Since STL can use any type of data as inputs and goal of this paper is to exploit any type of unlabeled data, only STL will be observed in the experiments. Specifically, Do & Ng (2005) proposed novel text classification models for MTL which exploit task-type of unlabeled text with Term-frequency-Inverse-Document-frequency (Tf-idf) function as a feature extractor with Softmax and Support Vector Machine (SVM) as classifiers. To extend data distribution of MTL from task-type to non-task-type, Raina et al. (2007) introduces STL, which uses any type of data as inputs to build feature extractors, where non-task-type data does not share class labels or generative distribution of data set from target tasks. They have shown that exploiting non-task type unlabeled data help model's performances under any circumstances. Specifically, their experiments cover various data type such as image, sound and text and shows improvements with STL. As a result, this thesis assumes using TL with any type of data will give help to build models and conducts experiments based on the assumption.

2.2 Positive and Unlabeled data in particular domains

Positive and Unlabeled (PU) data (Li & Liu, 2005). is one of the data distributions which does not contain unknown classes in the data set. For instance, all unlabeled data in PU data belongs to known classes, which are already revealed in the labeled data. To provide the all possible data distribution including PU and Negative Unlabeled (NU) data, Figure 2 illustrates examples of the text data corresponding to data distribution diagram. As the grey rectangle area indicates PU whereas grey circle denotes positive and labeled data, the rest of the white area represents negative (unknown) data. In practice, data for real application generally follows positive labeled and unlabeled data distribution. As Tao et al. (2017) addresses that real world appliations can be generalized as PU problems, it is common that many tasks in practice require PU data, particularly for supervised tasks such as classification. In addition to PU distribution, NU data can be exploited to build a model which is a learning method called TL.

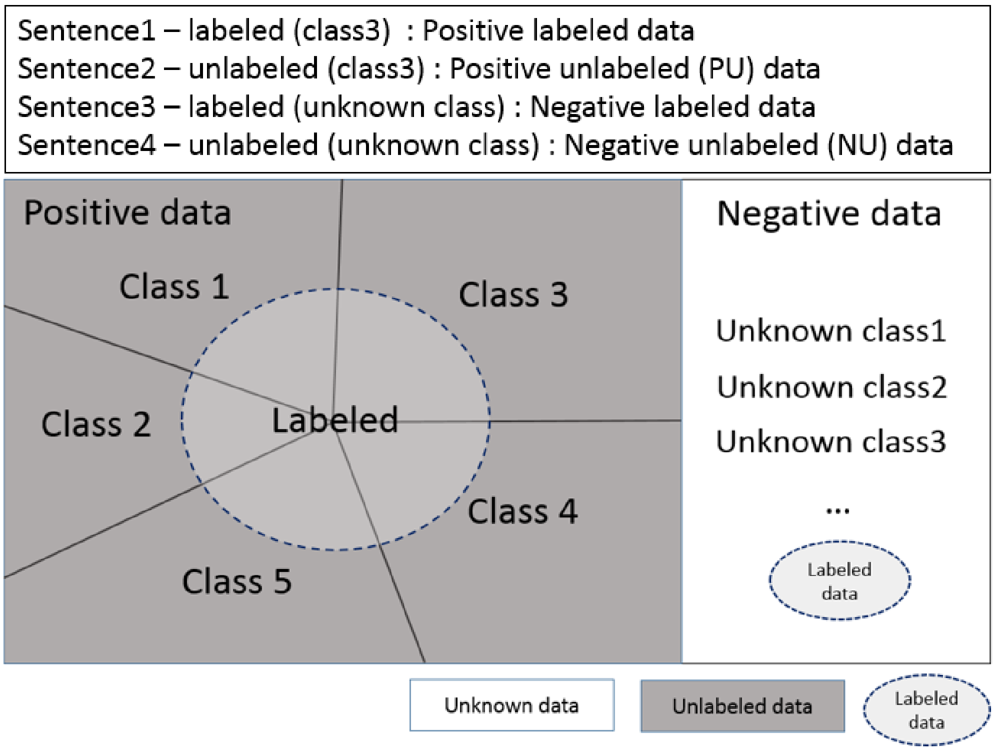

Sentence1 – labeled (class3) : Positive labeled data
Sentence2 – unlabeled (class3) : Positive unlabeled (PU) data
Sentence3 – labeled (unknown class) : Negative labeled data
Sentence4 – unlabeled (unknown class) : Negative unlabeled (NU) data
Positive data
Class 1
Class 3
Class 2
Labeled
Class 4
Class 5
Negative data
Unknown class1
Unknown class2
Unknown class3
...
Labeled data
Unknown data
Unlabeled data
Labeled data

Figure 2: Description of Positive and Negative data regarding the presence of label.

In particular fields such as legal and medical areas where annotators are mostly lawyers, judges and doctors which means that holding a large amount of the labeled data is expensive. Finding a cost-efficient model consuming the least labeled data is therefore crucial. To determine whether the amount of labeled data is enough to train a model, Matykiewicz & Pestian (2012) has studied how the size of labeled data affect model performance in medical fields dataset for TC tasks. However, their focus was on determining minimum labeled data for building a model but not exploiting unlabeled data. Since the primary focus of this paper is not only determining enough labeled data for training models but also finding the best model for exploiting unlabeled data. Therefore, unsupervised learning for using unlabeled data must be conjointly studied.

2.3 Unsupervised Learning for feature extractors

Unsupervised Learning (UL) is a statistical study that builds feature extractors with any types of data. This learning method is commonly considered as a key of understanding workflow of brain. As an example of supporting this view, Dayan et al. (1999) states "unsupervised learning is important since it is likely to be much more common in the brain than supervised learning". One of the advantages of UL is that more data construct better feature extractor. Since a massive amount of any types of data is available after the World Wide Web was created, building models with UL become popular. However, it is still challenging to implement UL since building statistical models for inputs and outputs requires a process of defining inputs and outputs. This is different from supervised learning which has predefined inputs and outputs for training ML models.

In this paper, three UL models are selected for the experiments. By adopting well-known models as baselines, the experiments provide fair results without weighting or giving poor score on the specific models. Specifically, two types of unsupervised learning list the following sections, which are rule based feature extractors and neural networks feature extractors.

2.3.1 Term frequency Inverse Document Frequency

The Term frequency-inverse document frequency (Tf-idf) was originally proposed by the female computer scientist, Karen Spärck Jones (1972, 1973). It was intended to find which words are important to a document or corpus. This can be found in the survey by Beel et al. (2016) where the authors states "TF-IDF was the most popular weighting scheme (70%) among those approaches for which the scheme was specified". These phenomena imply that the Tf-idf still plays an important role in IR areas. In fact, the Tf-idf method is often implemented in search engines or query systems in industrial areas.

The formula of the Tf-idf consists of two statistics, which are term frequency and inverse document frequency. Given by a collection or corpus of documents where each document contains multiple words, we can calculate the weight of each word. Each weight indicates whether the word is important for representing a document or not. For a term i in document j using the following equation:

$$W = tf_{i,j} \times log\left(\frac{N}{df_i}\right) \qquad (1)$$

Where Wstands for weight for term i, $tf_{i,j}$ is the number of occurrences of term frequency of term i in document j. In addition to term frequency, df_i stands for the number of document containing term i in the entire corpus, where N represents the number of total documents. Provided target term frequency and document frequency with contacting target word, we can calculate weights for all the terms for the corpus. To understand which values are derived by equation (1) for each term, Table 1 demonstrates the sample of Tf-idf weights of terms.

High score term	Score (weight)	Low score term	Score (weight)
A	10.90353755128617	the	1.344020275548612
deleting	10.210390370726225	and	1.5323546541931565
lulls	9.80492526261806	of	1.5729283469190423
knots	9.51724319016628	to	1.8122059658921892
fifth	9.29409963885207	that	2.095467396521632

Table 1. Terms with high and low Term-frequency inverse document frequency (Tf-idf) scores. The data set is Stanford Sentiment Treebank V1.0 (Socher et al., 2013) where one document is one sentence.

Table 1 shows how equation (1) produces high and low score terms that distinguish unique documents. In other words, low Tf-idf score words do not contribute to finding unique texts rather than high score terms. This attributes of the Tf-idf gives us a feature for each word. These features of word provided by the Tf-idf algorithm can be used for further ML such as supervised learning. Since this paper is based on classification tasks, these features will be forward to a supervised learning stage. Due to simplicity but robust performance of the Tf-idf, this algorithm is selected as a baseline feature extractor in this paper.

This unsupervised algorithm exploits unlabeled data as an input. Ko & Seo (2000) showed how to employ the Tf-idf for TC tasks with a Naïve Bayes classifier. However, they did not compare different classifiers but rather only changing their threshold for the classifier. Furthermore, a recent paper by Wang et al. (2017) has shown the comparisons of three different classifiers regarding three different features, which are the Tf-idf, Word2Vec[1] (Mikolov et al, 2013) and paragraph vector (Le & Mikolov, 2014). They showed that features with the Logistic Regression (LR) classifier outperforms the Multinomial Naïve Bayes or performs similarly to the Support Vector Classifier for classifying Chinese texts. Specifically, Wang et al. (2017) concluded in their paper that “logistic regression and support vector classifier with the Tf-idf or CounterVectorizer (term count matrix) feature attain the highest accuracy and are the most stable in all circumstances.” For these reasons, LR is selected as a classifier on top of the features provided by the Tf-idf.

2.3.2 Vector space model for word and character

The vector space is often providing high interpretability for observing the relation between points. Therefore, representing objects as points in the vector spaces increases readability of relations between objects. For examples, As Figure 3 demonstrates representations of locations with two attribute of latitude and longitude, it is intuitive to see how objects located close or far from each other. By comparing the distances of points in the vector space, similarity can be measured between objects.

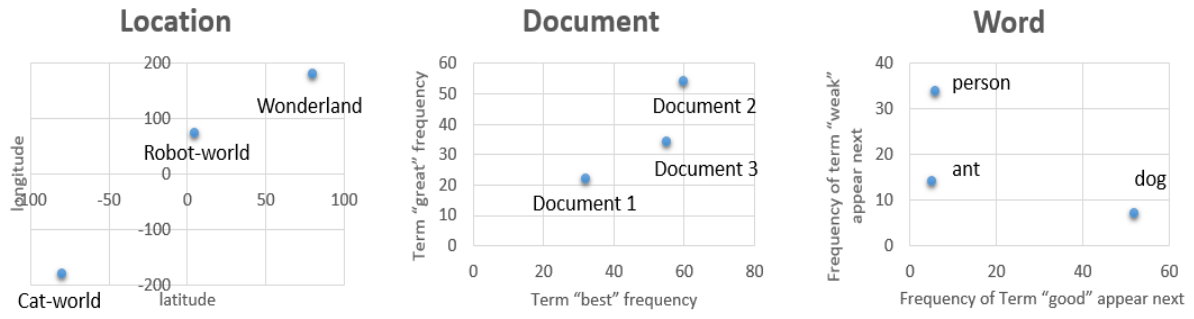


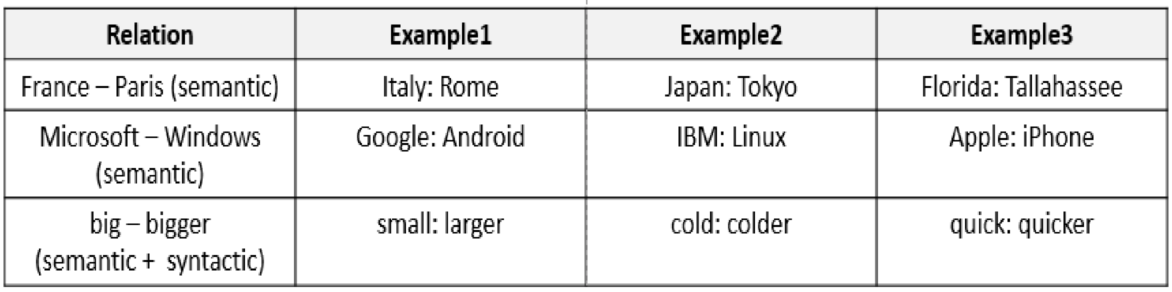
Figure 3: Examples of the vector representation in location, document and word vector space. One object exists as a point in the vector space.

[1] https;//code.google.com/p/word2vec/ [Accessed 10 May 2018]

The Vector Space Models (VSM) (Salton et al., 1975) were initially proposed in the IR system for indexing the documents. Salton et al. (1975) treated one document as a one vector so that each document in a corpus. In other words, one document exists as a point in a vector space with features, which are bags of word of N-grams. For example, *Document* space in Figure 3 describes a vector representation of document where two terms are the features of each document. Similar to the *Location* vector space, the example of a document space in Figure 3 shows that the relations between three documents are interpretable with the given features (axis).

In addition to the VSM of document, Turney & Pantel (2010) extend the VSM for Natural Language Processing (NLP) fields. They address that values of a word vector contains semantic meanings so that similarities of words, phrases, sentences and more elements of a language can be evaluated by calculating distance in vector spaces. For instance, a vector space for *Word* in Figure 3 describes that the term ‘dog’ is more related to a term ‘good’ rather than a term ‘weak’ while a word ‘ant’ is likely to hold the meaning of ‘weak’. Furthermore, the fact that the vector representation of words hold meaningful semantic information has been proven by Word2Vec (Mikolov et al., 2013a; 2013b) model.

Word2Vec (Mikolov et al., 2013a; 2013b) is an well-known unsupervised learning model for Neural Networks (NN). The main idea of this model shares the concepts from the quote “You shall know a word by the company it keeps.” by Firth, J. R (1957) and a notion from *distributional hypothesis* (Harris, 1954). These two ideas represent one hypothesis that words that occur in the same contexts tend to have similar meanings. Following this hypothesis, the Word2Vec framework is designed for predicting surrounded words given by a word or inverse way. After Mikolov et al. (2013a; 2013b) successfully found an efficient way to train a system in terms of memory and time cost, they found that trained word vectors contain linguistic attributes. To be specific, each word vector provides both semantic and syntactic information so that we can apply them to analogy or word sense tasks. As Mikolov et al. (2013a) show how to evaluate word vectors by using cosine distance between word vectors, Table 2 describes their evaluation results on semantic and syntactic similarity of words. Table 2 shows that their evaluation method gives a notion of not only semantic similarity but also syntactic similarity in the third row with a suffix ‘-er’.



Relation	Example1	Example2	Example3
France – Paris (semantic)	Italy: Rome	Japan: Tokyo	Florida: Tallahassee
Microsoft – Windows (semantic)	Google: Android	IBM: Linux	Apple: iPhone
big – bigger (semantic + syntactic)	small: larger	cold: colder	quick: quicker

Table 2. Relations of word pair and corresponding examples from the Table 8 in (Mikolov al., 2013a)

The Values of word vector is features of a word in VSM. These features of words can be used for inputs of NN, which also is known as an initialize word vector layer process. After these word vectors are preset by initialization in NN, classifiers are need for classification tasks. For selecting a proper classifier for word vector models, two branch of classifiers exist which are linear and non-linear types. Since both types of classifier are selected for classifiers for the word vector models in the experiments, word vectors trained by the Word2Vec conduct with non-linear classifier while word vectors trained by FastText (Bojanowski et al., 2016) uses linear classifier for classification. Specifically, a CNN based classifier will be used with word vector provided by Word2Vec model.

The vector representation of words can also be constructed by combinations of character vectors. FastText [2](Bojanowski et al., 2016) is one way to build character vectors. Bojanowski et al. (2016) used

[2] https://github.com/facebookresearch/FastText [Accessed 10 May 2018]

the Word2Vec model (Mikolov et al, 2013b) to build character vectors by replacing the inputs type from words to characters. After constructed character vectors, they show how to classify texts. Specifically, As Figure 4 shows a sample representation of document calculated by character vectors, Bojanowski et al. (2016) average a total of word vectors to build a representation of each document, which word vectors are summed up by the character vectors of the word. This simple approach to represent a document with a combination of character vectors for TC tasks is empirically proven that this model is fast and scalable to solve TC problems (Joulin et al., 2016). After obtaining the vector representations of documents, the FastText uses linear classifier with Softmax Regression (Multinomial Logistic) for the classifications. Moreover, Joulin et al. (2016) showed that the FastText is faster than the CNN based models for TC. For these advantages of simplicity and fast speed of the model, the FastText is selected as a second baseline for the experiments. By implementing the FastText which is based on a word vector model with linear classifier, two word vector models with linear classifier (FastText) and non-linear classifier (Word2Vec) are set for the experiments.

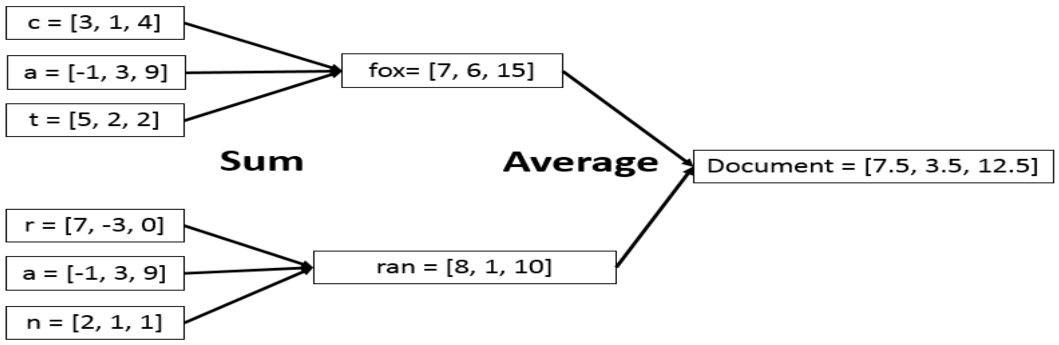


Figure 4. An example of a vector representation for one document in the FastText. One document consists of the average of word vectors which are the sums of character vectors.

2.3.3 Language Models

Language Model (LM) is a probabilistic model to generate texts. A Russian mathematician Andrey A. Markov (1953) introduced the LM to model the sequences of letters in the works of Russian literature. Despite the fact he did not use an exact term *Language Model* in his paper, Markov introduced the notion of the *Language Model*. The LM is based on the assumption (*Markov assumption*) of "The future is independent of the past given the present". In 1948, Shannon applied this assumption to describe his theory *Information theory* (C. E. Shannon, 1948). Shannon (1948) gave examples of modeling letters by sequences as Markov did in his work. Nowadays, this early LM based on a few time steps of letter sequences is commonly known as N-grams in linguistics. To describe Markov assumption formally:

$$\begin{aligned} &P(word_t, word_{t-1}, \dots, word_1) \\ &\quad = P(word_t \mid word_{t-1}, \dots, word_1)\, P(word_{t-1}, \dots, word_1) \qquad (\mathbf{2})\ \boldsymbol{Bayes' \ rule} \end{aligned}$$

$$P(word_t \mid word_{t-1}, \dots, word_1) = P(word_t \mid word_{t-1}) \qquad (\mathbf{3})\ \boldsymbol{Markov\ assumption}$$

$$\begin{aligned} P(word_t, word_{t-1}, \dots, word_1) &= P(word_1) \prod_{i=1}^{t} P\,(word_i \mid word_{i-1}) \\ &= P(word_1) P(word_2 | word_1) \dots P(word_t | word_{t-1}) \qquad (\mathbf{4})\ \boldsymbol{chain\ rule} \end{aligned}$$

Where t indicate a time step for all three equations, equation (2) is a conditional probability where all previous words must appear as a condition of current word. In contrast, equation (3) indicates that the current state is only dependent on the last previous state but not the entire previous words for each time step. With this assumption (3), the chain rule (4) can be derived as a product of (2) and (3). Despite the fact that this assumption is naive which considers only the last state at current time step, it has been broadly used for real applications in Science and Linguistics fields such as researches in Speech Recognition and DNA sequencing. To be specific, *Markov assumption* is playing important roles in NLP as the concepts of *N-gram*, *Hidden Markov Model*. These concepts are bases of Speech Recognition system, Part-Of-Speech Tagger, Dependency Parser and other purposes.

As computer was developed, NN was popularized among researchers in Science fields. For language modeling in NN, Little (1974) introduced a NN architecture which holds memory, and then Hopfield proposed the (Little-) Hopfield Network (1982) which is today called Recurrent Neural Network (RNN). Since this architecture has memory cells to store all previous information unlike Markov model which stores only last time step, RNN become popular for language modeling in NN. However, it was difficult to implement NN for language modeling due to limitation of computer hardware in the 1980s. As technology in Computer Science has been significantly developed in the 2000s, we are now easily able to build Neural Language Models (NLM) that benefit more than traditional N-gram language models.

Unfortunately, the RNN architecture has a drawback of long-term memory. The problem of long-term memory is caused by long sequences of input, which leads RNN to lose information at early sequences of inputs. Y. Bengio et al. (1994) pointed out this long-term memory problem that RNN has an *vanishing gradient problem* in proportion to length of sequences. In other words, long sequence inputs make RNN lose the long-term information but rather store only last short-term information. The RNN can handle long-term dependencies in theory but it is not capable to store all long-term and short-term information from in practice. In order to resolve this long-term memory issue, Hochreiter & Schmidhuber suggested Long-Short Term Memory (LSTM) (1997) by adding gates to check whether inputs are important or not to pass to RNN cells.

LSTM is an architecture of unit in NN which consists of additional gates over RNN. It has *forget*, *input* and *output* gates which are working as an internal memory controller. With two more gates of *input* and *forget*, we can decide whether to discard current input and previous recurrent output or save them. In other words, we filter input sequence by its importance. We describe the difference in architecture between RNN and LSTM in Figure 5. This figure shows that LSTM contains three extra gates which provide inputs for cells whereas RNN has only one single gate.

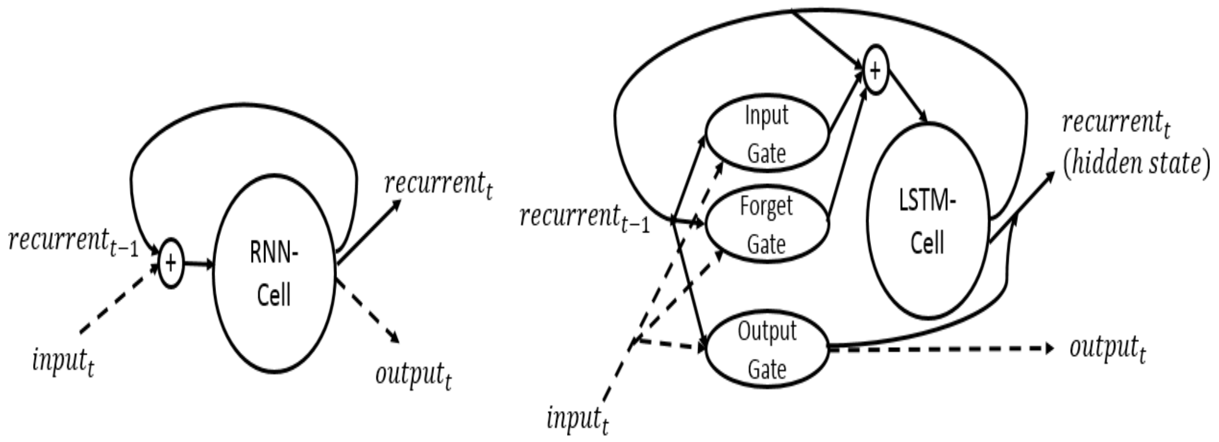


Figure 5: Two flows of input and output in Recurrent Neural Networks (RNN) and Long-Short Term Memory (LSTM)

In a formal way, equations for three extra gates and recurrent state (hidden state) can be written

$$f_t = \sigma_g\left(W_f x_t + U_f h_{t-1} + b_f\right) \quad (\mathbf{5})$$

$$i_t = \sigma_g(W_i x_t + U_i h_{t-1} + b_i) \quad (\mathbf{6})$$

$$o_t = \sigma_g(W_o x_t + U_o h_{t-1} + b_o) \quad (\mathbf{7})$$

$$c_t = f_t \circ c_{t-1} + i_t \circ \sigma_c(W_c x_t + U_c h_{t-1} + b_c) \quad (\mathbf{8})$$

$$h_t = \sigma_t \circ \sigma_h(c_t) \quad (\mathbf{9})$$

In the above equations, weight matrix are W and U for input x and hidden state h_{t-1} whereas b means bias for each gate. Moreover, all gates, cell state and hidden state has one or more activation functions where σ_g refers sigmoid function whereas σ_h and σ_t are hyperbolic tangent functions. In equation (5), (6) and (7), each f, i and o represent forget, input and output gate vectors, t indicates time step for present. After obtaining output vectors from input and forget gates f and i, we can calculate the cell state vector (c_t) as described in the equation (8). For the last step, hidden state (h_t) at time t is propagated to the next time step $(t+1)$ as a next input vector. Describing the above equations with Figure 5, hidden state (h_t) for time step t is equivalent to $recurrent_t$. Similarly, input (x_t) equals to $input_t$

Thanks to these extra three gates that handle inputs differently from RNN, LSTM is able to distinguish whether to discard or to keep inputs for training a model. This selective process enables LSTM to overcome the long-term memory problem so that LSTM covers more than sequences input with a thousand time steps. For this advantage, LSTM became one of the popular NLM regarding sequencing inputs. However, NLM has one more drawback which is known as *the curse of dimensionality* (Bengio et al., 2003) which occurred by a large size of the vocabulary.

Another issue of RNN is the *Curse of dimensionality* (Bengio et al., 2003) which refers to a phenomenon where the size of dimension causes exponential load of calculation in statistics. For instance, each ten of row and column generate one hundred parameters of matrix, whereas one hundreds of row and column makes ten thousand size of matrix. To solve the burden of calculations increased by the size of the matrix, the higher specification of hardware and more time are required. Since vocabulary size of human language can be easily up to 10,000, building vocabulary matrix can be challenging. For example, even a small size of 2000 vocabulary makes 4million parameters of matrix. In order to fight this curse, Bengio et al. (Bengio et al., 2003) demonstrated how to compress vocabulary matrix in an efficient way by introducing a word embedding layer, look up tables and small dimension of feature (few hundred dimension). By their method, features of each words are no longer interpretable but it gives efficiency of memory and fast speed for calculation.

After these two problems have been fairly resolved, traditional statistical language models have been replaced by NLM. For example, models based on mutual information matrix are replaced by neural networks stacked by word-embedding and LSTMs. Since this architecture based on word-embedding, LSTM became a standard of NN architecture for NLP tasks. For examples, two famous LSTM taxed based NLP of this architecture is that *bi-LSTM* (Schuster & Paliwal, 1997; Graves & Schmidhuber, 2005) and *encoder-decoder* model for machine translation (Cho et al., 2014). These two papers address how to use effectively (Schuster & Paliwal, 1997; Graves & Schmidhuber, 2005) and efficiently (Cho et al., 2014) for text based NLP tasks.

2.4 Supervised Learning for classifiers

In the previous section, three unsupervised learning approaches have been described for building extract features. Now that features are obtained from unsupervised feature extractors, classifier need to use them for classifications tasks. This supervised learning section describes how the training stage for classifiers differs from building unsupervised feature extractors. Finally, the two type of classifiers will be listed, linear and neural network models for the experiments.

2.4.1 Relation between labels and unlabeled data

When it comes to data, two steps exist for collecting labeled data. The first step is to obtain raw (unlabeled) data, and then annotate collected raw data for specific tasks. In the last section, we have seen that unsupervised learning was used for exploiting only unlabeled (raw) data. In contrast to unsupervised learning, supervised learning requires labeled data to train a classifier for classification tasks. These classification tasks need a component called a classifier to decide which data belongs to which category. Since a classifier makes a decision for predicting labels for input data, it plays an important role in classification tasks in addition to feature extractors. In this paper, two types of classifiers are used for the experiments, which are linear and non-linear classifiers.

2.4.2 Linear classifier

Linear classifier is based on linear algorithms such as simple *linear* or *logistics* (*logit*) function. These algorithms are normally used when data is linearly separable. One advantage of using a linear classifier is that it is interpretable to determine the relation between linear functions and the data. For this reason, all models in the experiments except one word-vector model use linear classifiers as a default. Specifically, *logistic function* is selected for the Tf-idf and *Softmax function* used for the character-vector and the LM.

The *Logistic function* is implemented in the three models in this paper. This function was developed by David Cox (1958) and Walker & Duncan (1967) to estimate a response of dependent variables for binrary classification. This function provides probabilties of category of input variables which typically refer to feature of data. Moreover, a generalized logistic function, *Softmax function (Multinomial Logistic Regression)* can be implemented for multiple categories problems. Since most neural networks use *convex maximum entropy function* (*cross-entropy*) for calculating loss during a training models, classifiers in neural networks behave the same way as Softmax function does. As most neural network models use Softmax fution for classifiers, all neural network models in this paper also use logistic function as a default. Moreover, the character-vector model also use Softmax function for a classifier. To express logistic and Softmax funtions in a formal way,

$$log\left(\frac{P(class = y|\, x)}{1 - P(class = y|\, x)}\right) = \ b_0 + b_1x_1 + b_2x_2 + \cdots + b_nx_n \quad (10)$$

$$P(class = y \mid x) = \frac{e^{b_0+b_1x_1+b_2x_2+\cdots+b_nx_n}}{1 + e^{b_0+b_1x_1+b_2x_2+\cdots+b_nx_n}} \quad (11)$$

$$P^{-1}(class = y \mid x) = \frac{1}{1 + e^{-(b_0+b_1x_1+b_2x_2+\cdots+b_nx_n)}} \quad (12) \quad \boldsymbol{logistic\,funtion}$$

$$P(class = y \mid x) = \frac{e^{x\cdot w_y}}{\sum_{k=1}^{K} e^{x\cdot w_k}} \quad (13) \quad \boldsymbol{softmax\,function}$$

Where x indicates variables whether its response belongs to class y or not, b and w denotes the weight corresponding to each varibale x. Final logistic funtion is (12) provides outcome as a probability of varibales where class y is a binary dependent variable. To generalize binary class to multi-class tasks, each probability of class y can be computed as a part of the sum of probablities of the total classes y as described in (13). With this generalized version of logistic (Softmax) funtion, Softmax classifiers can be emplyed for multiclass classifications.

However, Softmax linear classifier cannot cover situations where data is not linearly separable. For example, Y. Wang et al. (2017) have shown that word features obtained by the Word2Vec does not perform in TC tasks with linear classifiers. One way to improve the performance is changing the features of inputs which can be linearly separately. Another way to overcome performance problem is to replace linear classifiers with non-linear classifiers. Both approaches are available for the word-vector features. An example of the first method for word-vector features can be observed in FastText (Joulin et al., 2016; Bojanowski et al., 2016). Joulin et al. (2016) implemented the first approach and found that features obtained by combination of word-vectors are linearly separable, and then they apply (linear) *Softmax* classifier for classification. The second way to treat word vector feature is to use non-linear classifier instead of linear classifier. Since the first method is already implemented in one experiment, the second approach will be investigated with the word-vector models in the experiment. Specifically, the CNN non-linear classifier (Kim, 2014) is adopted for the second approach.

2.4.3 Non-linear classifier

Convolution neural networks (CNN*)* (Fukushima, 1988; LeCun et al., 1999) is one of the popular architectures used in Computer Vision (CV) for detecting and recognizing object tasks. This architecture is based on filters and pooling methods between neural network layers for CV tasks, but also can be applied to text based tasks as a classifier. For example, Kim (Kim, 2014) has shown how to apply CNN for TC tasks with word vectors features. He demonstrates how CNN as a classifier with existing word vector features can be employed for texts by setting one dimensional filter rather than two dimensions in CV tasks. Since this architecture is based on non-linear activation function, it can cover non-linear separable input data. Moreover, Kim (2014) addresses the non-linear CNN classifier performs well with word-vectors and Wang et al. (2017) shows that linear classifiers underperform with word-vector features. For these performance reasons, the performance between non-linear and linear classifier for word-vector features will be compared in this paper.

In fact, non-linear classifier based on CNN has been recently studied and proven that the CNN classifiers outperform simple linear classifiers. For example, Johnson and Zhang (2016, 2017) describes that word and character based CNN classifiers reach the state-of-the-art for various TC tasks. For this performance reason, CNN is selected as a non-linear classifier to conduct with word vector features.

2.5 Relations between two training stages

For classification tasks, feature extractors can be built before the training classifiers stage. Thus, there are two training steps that exist. First is training feature extractor and second is for training classifier. The domain of building a feature extractor stage is known as a source domain and the domain of training a classifier is called target domain. There are several different relations exist between source and target domains. However, this paper will only cover two relations for exploiting unlabeled data, which are Semi-Supervised Learning and Transfer Learning. To prevent mixed definition of terms among research papers, this paper strictly follows the definitions of *semi-supervised learning* and *transfer learning* from the survey written by Pan & Yang (2010)

2.5.1 Semi-supervised Learning

The first relation between source and target domains is Semi-Supervised Learning (SSL). As Figure 6 describes, this relation is where the source domain (stage1) remains same during a training classifier (stage 2). This learning assumes that two stages share not only the domain but also the distribution of data. In other words, only PU data is used for training a LM feature extractor, so that only known classes exist in the entire dataset. To demonstrate the relation between two domains, each domain has a different color which is either blue or orange. Since objective of this paper is to exploit PU data for building feature extractors, SSL accounts for most parts of the experiments.

SSL is an efficient way to build a feature extractor with a small amount of labeled data, in expensive domains. Since the labeling process is expensive and slow in high-cost domains, only unlabeled data can be available. Moreover, the domain of unlabeled data is already known while collecting them, hence these unlabeled data follows PU distribution. In other words, unlabeled data in expensive domains follows PU data, and exploiting PU data to build a feature extractor results in SSL. As this paper clarified the main interest of this paper is to build feature extractors with PU data in the section 2.1, most experiments will be based on SSL in the experiments.

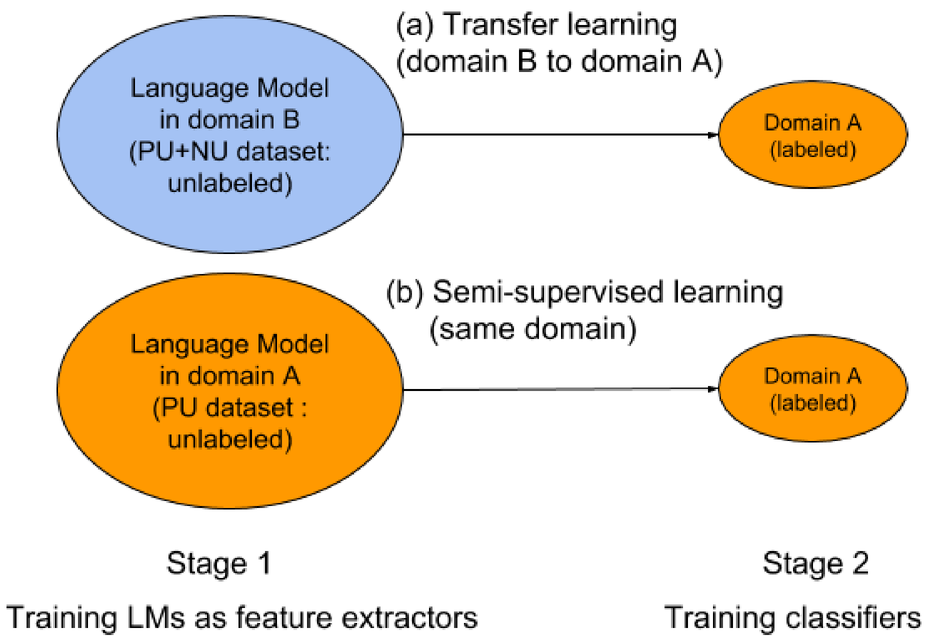
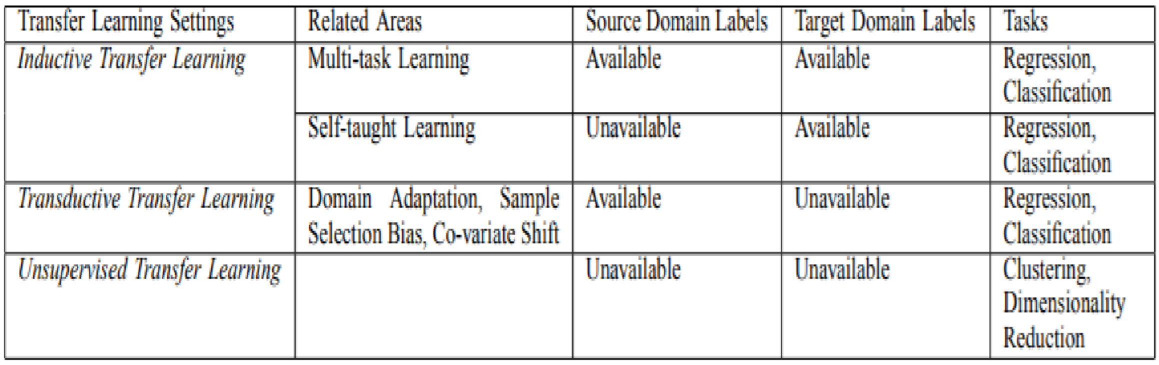

(a) Transfer learning
(domain B to domain A)
Language Model in domain B (PU+NU dataset: unlabeled)
Domain A (labeled)
(b) Semi-supervised learning
(same domain)
Language Model in domain A (PU dataset : unlabeled)
Domain A (labeled)
Stage 1
Training LMs as feature extractors
Stage 2
Training classifiers

Figure 6: Two learning methods which differ by the relations between the source and the target domains. The left circles are source domain while the right circles are target domains for classification tasks. Color of circle represents the domain of the dataset.

2.5.2 Transfer Learning

The second relation between source and target domains is Transfer Learning (TL). Transfer learning is a relation where the source domain is different from the target domain but still related. As Table 3 (Pan & Yang, 2010) shows different types of TL by availability of the source and the target domains, this paper is limited to *Inductive Transfer Learning* where the target domain labels are available. Therefore, the two types of TL will be explored in this paper. The first type of TL is called Self-Taught Learning (STL) (Raina et al., 2007) which uses not only PU but also Negative Unlabeled (NU) data for training feature extractors. The second type is Multi-Tasks Learning (MTL) which also uses both PU and NU dataset. The only difference between MTL and STL is the availability of the label in NU data. Specifically, STL is not limited to the availablity of the label but MTL requires labels of NU data. For the experiments, STL use Wikipedia texts for *stage1* in Figure 6 while MTL will be trained with different domain data as NU datasets.



Transfer Learning Settings	Related Areas	Source Domain Labels	Target Domain Labels	Tasks
Inductive Transfer Learning	Multi-task Learning	Available	Available	Regression, Classification
	Self-taught Learning	Unavailable	Available	Regression, Classification
Transductive Transfer Learning	Domain Adaptation, Sample Selection Bias, Co-variate Shift	Available	Unavailable	Regression, Classification
Unsupervised Transfer Learning		Unavailable	Unavailable	Clustering, Dimensionality Reduction

Table 3. Types of Transfer Learning. Above table is taken from a paper by Pan & Yang (2010)

TL is an efficient learning method to build feature extractors which uses both NU and PU data. Since any data can be exploited for TL, the coverage of TL is wider than SSL which require PU data for pre-training stage. Thanks to the broad coverage of TL, TL is used for many real applications in which labeled data is not available for pre-training stage. For example, for CV, pre-trained feature extractor is heavily implemented in NN architectures and functions as a feature extractor of input images. This TL can be successfully deployed in CV because many tasks in CV are not domain-sensitive or the types of tasks are not significantly different from other tasks. Typical CV tasks are limited in object detection, recognition and object prediction by each frame so that type of features are not considerably changed from one domain to others. However, TL in NLP is still challenging to implement for text based tasks due to sensitivity of domains and diversity of tasks.

When source and target domains are not related for TL, TL does not benefit to build a better model but rather TL hurts the performance of the model. This phenomenon is known as the *negative transfer* effect which is casued by the sensitivity of two source and target domains. In spite of the fact that TL has no limits to exploit any unlabeled data (Raina et al., 2007), TL often downgrades the performance of models because of the *negative transfer* effect. For NLP tasks, most problems are domain-sensitive so that most tasks are inevitable to encounter the *negative transfer* effect. As a result, TL is not easy to implement for NLP tasks.

In addition to the negative transfer issue, numerous types of NLP tasks make it difficult to implement TL. For example, there are various tasks in NLP such as sentimental analysis, topic modeling, question-and-answering, entailment, syntactical parsing, etc. It is not only the assumptions of these tasks that are unrelated but also the features of inputs are different. To be specific, features of a word in sentimental analysis do not contain the same information of syntactical parsing tasks but rather each of the features contains different meanings. This differs from CV fields which share features of input images for many tasks. With this attribute of not sharing features between different NLP tasks, applying TL in NLP is laborious or often cannot be resolved.

Unresolvable problems occur when reusability of a feature extractor is not available. Unlike a feature extractor in CV, a feature extractor NLP behaves differently while transferring pre-train knowledge to target tasks. For example, a feature extractor in object detection for CV tasks can be reused, which provides color feature of objects. However, a feature extractor for syntactic parsing in NLP gives a feature such as the presence of letter '-ly' as a suffix, which can only be reusable only if the target task is related to suffix. To be specific, CV feature extractors can be reusable when color feature of image still plays an important role for CV tasks. However, feature extractors in NLP do not provide shared features between the source and the target domain or tasks. An example of non-reusable NLP feature extractor is that a feature extractor that detects suffix of '-ly' cannot be reusable for textual entailment tasks in NLP. For this non-reusability reason, text based TL still remains difficult problems. However, many studies have done in transfer learning for NLP tasks and it became more important subject lately.

Recently, many studies in TL for NLP tasks have studied. For example, Mou et al. (2016) has observed the transferability of neural networks to determine when negative transfer effect become big issue. Moreover, Howard & Ruder (2018) shows how to apply self-taught learning to different tasks type with reaching the state-of-the-art performance. Since the primary goal of this paper is find a best model regarding SSL, TL is implemented to support SSL rather than as a main subject of this paper.

2.6 Regularization techniques

Regularization techniques are crucial to prevent overfitting model in Machine Learning. One of the practical regularization techniques in neural networks is *Dropout* (Hinton et al., 2012; Srivastava et al., 2014), which drops half of the units of each layer while forwarding previous units to next layer. Hinton et al (2012) introduced this concept to avoid overfitting problems in neural networks structures. It is a process randomly omitting half of the units of layers by only forwarding half of the units to next layers. Remarkably, it turned out that Dropout enables a model to learn generalized knowledge without overfitting on training dataset.

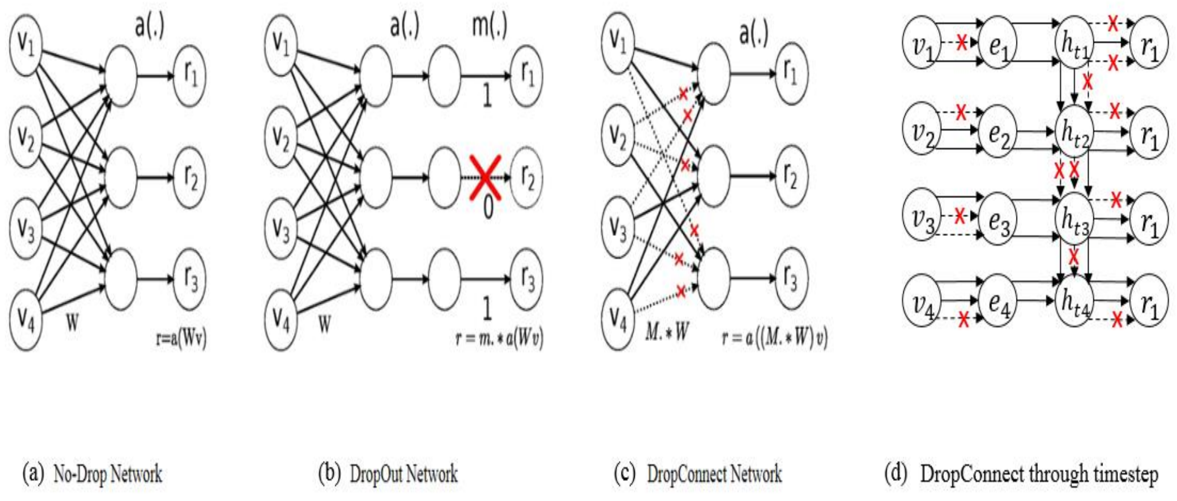


Figure 7. The figures (a), (b) and (c) are taken from slide[3] of a study written by Wan et al. (2013). (d) is where DropConnect is applied between hidden-to-hidden layers in Recurrent Neural Networks.

To extend the Dropout technique, Wan et al. (2013) proposed a technique *DropConnect* which is a generalized Dropout for weights in each unit. The difference between Dropout and DropConnect is that the first algorithm drops units of layer during the forwarding step while the second method drops elements (weights) of units. This difference between the two methods is described in Figure 7. In the figure, the left scenario (a) demonstrates non-Dropout which is a naive way to forward weights in the current layer to the next layer. The second situation illustrates that one unit of the current layer is dropped and discarded during forwarding weight to the next layer. The last description describes elements of units in the current layer is dropped rather than the entire elements of one unit is dropped as Dropout does. In conclusion, DropConnect is a generalized version of both No-Drop and Dropout networks, where dropout rate is either zero (No-drop) or sharing the same value for each unit (Dropout).

In Figure 7, r is a unit in a layer, W is fully-connected layer weights and a indicates the activation function. For the No-Drop networks, none of units or weights are dropped during the forwarding the units to the next layer. However, the Dropout network has an additional factor m which is masking for units so that it drops units with probability with p. For instance, If p is 0.5, it means half of the units in

[3] https://cs.nyu.edu/~wanli/dropc/dropc_slides.pdf [Accessed 10 May 2018]

a layer will not be forwarded to the next layer. Finally, the DropConnect network has an extra two components of M and v which are masking for weights and activation function. Given the DropConnect formula, Dropout and No-Drop networks are special cases of DropConnect network. Furthermore, DropConnect can be applied to RNN as described in (d) in Figure 7 where e is embedding element (word vector) and h_t is hidden state at a time step t.

However, applying DropConnect to the neural language model is not easy since the architecture of RNN is memory intensive. As (d) in Figure 7 demonstrates Dropout through time step, hidden state at the last time step will heavily drop weights in proportion to the length of the input sequence. By applying DropConnect to hidden states, LSTM cell behaves as simple RNN cell does. Recall that the previous section *2.3.3 Language Models* demonstrates how RNN cell architecture causes long-term memory loss. In other words, when one applies DropConnect between the RNN's hidden states, the long-term memory disappears again by many Dropout through many time steps. To resolve this difficulty, Zaremba et al. (2014) has shown how to apply DropConnect for hidden-to-hidden states that helps RNN to avoid the long-term dependency issue. To extend a study by Zaremba et al. (2014), Merity et al.(2017) has shown that DropConnect could be applied between the hidden-to-hidden weight matrix instead of the hidden-to-hidden states, and then they showed the result of the high performance of their approach.

In addition to the Dropout technique, Merity et al.(2017) suggested to use NT-ASGD (Non-monotonically Triggered Averaged Stochastic Gradient Descent) which is an variant of SGD. Briefly, this algorithm averages SGD by reducing the learning rate when it reaches the trigger point with a pre-defined patience. The trigger point can be set by model's performance such as perplexity (Shannon, 1948) and validation loss. More detail on NT-ASGD is described in the study by Merity et al. (2017).

Besides the previous regularization techniques, Merity et al. (2017) also employed the other additional optimization techniques such as weight tying and variable length for backpropagation through time (BPTT). Due to many but not complex regularization techniques for building LM in their approaches (Merity et al., 2017), AWD-LSTM language model is selected as a main LM in this paper.

3 Methodology

This chapter demonstrates the details of the experiments in general. First of all, the process on how the datasets were collected and the techniques in preprocessing are explained. The next part describes how to convert distributions of the datasets and normalize unbalanced data. In the hyperparameters setting section, it specifies the values of the hyperparameters for both of unsupervised learning and supervised learning. Most details regarding hyperparameters are related to the Language Model (LM) in Neural Networks (NN). After clarifying all the hyperparameters, different approaches of initializing neural networks layers will be explored. In the section "obtaining information from all layers", various methods to collect information from all the NN layers are introduced to maximize the performance in a small number of labeled data. Finally, a proper evaluation method for the trained NN models will be proposed, where only a few train dataset is available.

3.1 Collecting data and preprocessing datasets

A total of six datasets were collected for the TC tasks and additional texts from Wikipedia (Merity et al., 2017) was obtained for Transfer Learning (TL). The details of the dataset are described in Table 4. Four of the preprocessed datasets are Agnews, DBpedia, Yahoo Answer and Yelp Review which were collected by Zhang et al. (2015) and the other two datasets are the TripAdvisor Hotel Reviews and the Amazon product six categories (H. Wang, Lu, & Zhai, 2010; 2011). Table 4 describes the number of classes, the task type of the datasets and the size of each train dataset for supervised tasks, where only the Wikipedia dataset does not have any labels. Since the size of Agnews is the smallest of the six datasets, experimenting with Agnews dataset is faster than conducting with the other datasets. For this speed reason, this paper only uses the Agnews dataset for the experiments that are related to maximizing the performance of the LM.

	classes	Train set size(MB)	Test set size (MB)	Task type
Agnews	4	25	1.6	Topic
TripAdvisor Hotel Review	5	98	25	Sentiment
Amazon six categories	6	91	23	Topic
DBpedia	5	142	18	Topic
Yahoo Answers	10	325	25	Topic
Yelp Reviews	5	526	23	Sentiment
Wiki-103	-	526	-	-

Table 4. Details of collected datasets for text classification tasks. Only TripAdvisor hotel review and amazon six categories are parsed from the JSON files. Other datasets are already collected and preprocessed by Zhang et al. (2015) except the Wikipedia texts (Merity et al., 2017).

In practice, the proportions of each class are not equally normalized between classes. In order to mimic realistic situations, two unbalanced datasets of the Amazon Six Categories and the TripAdvisor Hotel Reviews are intentionally used for the experiments. In contrast to the four balanced datasets, these two imbalanced datasets consist of a different amount of data for each class. Specifically, the two classes with the smallest amount of data will be chosen as difficult tasks for TC. Despite the fact that it is not always the case that a class with small data is problematic, it is likely to be the difficult task in most cases. Therefore, the two classes of a small amount of the data were selected as difficult tasks in the experiments. By evaluating the four models with these two small classes, this paper can provide sensitivity of each model.

After collecting all the datasets which consist of six datasets and one Wikipedia texts, lowering case of letters and removing stop words are conducted, and then a parser for proper nouns was used for the preprocessing. For example, the parser converts proper nouns with two words into one word by replacing a space to an underscore symbol '_' such as 'New York' into 'New_York'. For the proper noun parser, 2,3 and 4 grams are used for all the datasets except the Wikipedia texts which was collected and preprocessed by Merity et al. (2017).

Preprocessing in the datasets for the LM is a critical step where the number of vocabulary could be problematic. For example, 260k vocabulary in the Wikipedia dataset is not feasible to the LM training stage even with a small batch size of 2. For this reason, a limited size of vocabulary is used for the preprocessing stage for the LM. To be specific, train datasets with the size under 200MB have 30k of fixed vocabulary size while over 200MB have limited to a 50k vocabulary size. By limiting the vocabulary size, most frequent words are only used for training the LM. Since the proper noun parser made 2-4 grams for common proper nouns, training dataset for the LM includes a small amount of 2-4 grams which is not a different order of words but is considered as one word. All these datasets are available at sungmin-github[4].

3.2 Converting data distribution and normalizing proportions of data.

In addition to collecting data and preprocessing for text data, Limiting the amount of labeled data is a crucial preprocessing part for building Semi-Supervised Learning (SSL) environments. With controlling the number of labeled instance, controlled data follows PU data distribution. Since the final step of all models in the experiments is a classification task, labeled data is always required for training classifiers. At this scope, controlling the number of labeled data for training classifier is a main key of SSL. In Figure 8, (a) describes a situation where the positive labeled data shifts to PU data by simply removing labels, with this approach (a) in Figure 8, data distribution is converted from supervised learning to SSL datasets. Specifically, this paper randomly sample labeled 10, 50, 100, 200, 500,1000, 2000 sentence (instance) per class are used for training classifiers. Besides limited labeled data, also the total trained data will be fed to train classifiers. As significant improvement is not observed after a few thousands of instance per class, 2000 will be the final limited number of labeled data.

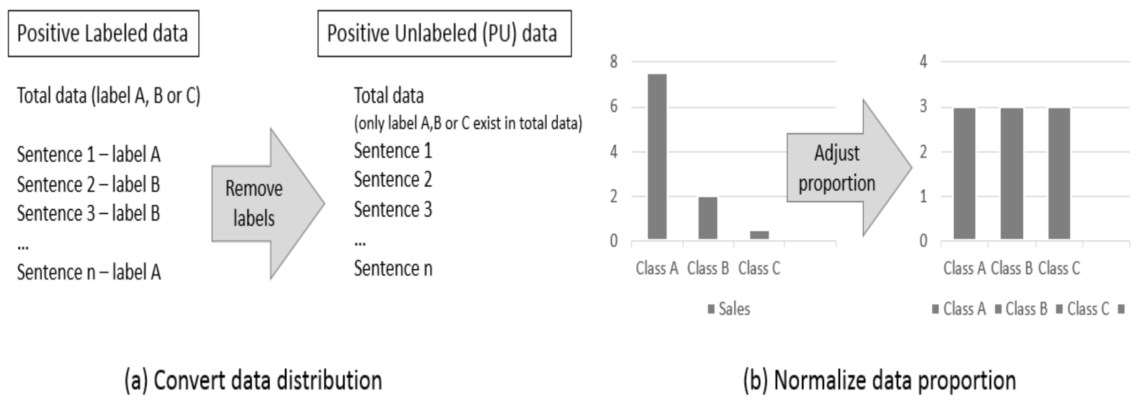


Figure 8. A conversion of labeled data and a normalization of unbalanced data. (a) describes a situation where converting data distribution from Positive Labeled data to Positive Unlabeled (PU) data for building semi-supervised learning environments, while (b) is a process to adjusting the amount of unbalanced proportion of data to be equal.

[4]https://github.com/sungmin-yang/four-unsupervised

Unlike building environment for SSL, data for TL does not require any splitting or controlling labeled data, since it does not need labels at all. Furthermore, other advantage of collecting TL data is that it consumes both PU and NU data, which exist everywhere. For this freedom of input types and broad availability, any texts can be fed to TL. For example, raw texts from Wikipedia or corpus of news articles can be fed to training feature extractor. Specifically, in the pre-training stage for TL in LM, NU text data of Wikipedia-103 (Merity et al., 2017) is used for training LM feature extractor.

Now that one has set all the data distribution for SSL and TL, different proportions of data remain as a problem for classification tasks. The different amount of data per class causes overfitting during training ML systems. As ML systems for classification are based on consuming data for a training models to predict each class, the number of training data per class is crucial. Without data normalization, using a class which accounts for substantial proportion of the entire dataset will cause overfitting on provided big proportion one class. To avoid this issue, both down-sampling and over-sampling are applied while controlling the amount of labeled data. As (b) in Figure 8 demonstrates two example of converting data distribution and normalizing proportion of data, we can think of a situation where only two class A and B exist where each data has 10 and 90 instance per class. In this particular case, sampling the fixed number of labeled data for 5 instances per class will over-sample class A while class B has less sampled regarding the proportion of A and B data. By this normalization technique, classifiers could properly learn without giving more weights on particular classes which account for a large proportion of data.

3.3 Hyperparameter setting

This section specifies the hyperparameters for two stages of training by unsupervised and supervised learning. To obtain features or build feature extractors, it can be built by unsupervised learning algorithms. All hyperparameters for a supervised model is also important as unsupervised learning training. Most experiments in unsupervised learning use default values provided from the original codes of scikit-learn, Gensim, FastText and AWD-LSTM.

Before setting the hyperparameter for the Tf-idf model, unigram and bigrams are used as inputs. After inputs are defined for Tf-idf, this paper follows the default hyperparameter values of smoothing and extra parameters can be found in scikit-learn[5] library. For obtaining word vectors, all vectors are trained with Gensim [6]library with dimension 100 with context size 7 and minimum word counts are 3 with 15 iterations for each dataset. For FastText[7], dimension 100 is selected for all data sets except 300 is used for Yelp and Yahoo dataset. All models are trained with 15 iterations.

Hyperparameters for NLM, all trained LM use default values specified in AWD-LSTM[8]. Specifically, back propagation through time (BPTT) is set to 70 with 3 layers RNN layers with 1150 hidden units. For Dropout for embedding layer, 0.65 and 0.1 are used for input and output where layer and RNN layers has 0.4 and 0.3 for each Dropout rate. WeightDrop (DropConnect) is set to 0.5 between RNN hidden-to-hidden matrix. In addition to hyperparameters in neural network layers, Stochastic Gradient Descent (SGD) optimizer is selected with learning rate of 30. Structures of LM consists of 400 dimension for word embedding layer and 3 layers of LSTM with the last layer sharing the same dimension as word embedding layer, which is 400. For training LM with AWD-LSTM, all LMs are trained on three GPUs of specification, two 1080Ti 11GB and one Titan X 12GB. Table 5 describes each number of epoch trained for LM and the size of datasets. Since training LM is computationally heavy, relatively small training epoch is used for the big datasets. For example, the total training time for 523 epochs in the Agnews dataset is similar to the time of 15 epochs in the Yelp Reviews dataset.

[5] http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.Tf-idfVectorizer.html [Accessed 10 May 2018]
[6] https://radimrehurek.com/gensim/models/word2vec.html [Accessed 10 May 2018]
[7] https://github.com/facebookresearch/fastText [Accessed 10 May 2018]
[8] https://github.com/salesforce/awd-lstm-lm [Accessed 10 May 2018]

	Epoch	Vocabulary size	File size (MB) (without labels)
Agnews	523	30k	21
TripAdvisor Hotel review	114	30k	95
Amazon six categories	121	30k	86
DBpedia	86	30k	128
Yahoo Answers	27	50k	340
Yelp Reviews	15	50k	467
Wiki-103	84	50k	450

Table 5. Specification and the training time of the Language Model for the six datasets.

In addition to hyperparameters in unsupervised learning, Hyperparameters also play important roles for supervised learning. As Tf-idf is jointly training with linear classifier which uses logistic function, default hyperparameters from scikit-learn[9] library are used for the experiments. For FastText, this paper uses epochs of 20 and learning rate 1.0 with provided other default hyperparameter values. CNN classifier (Kim, 2014) for word vectors uses 10 filters with size 3 and 8 instead of using the default values of 100 fileters with sizes of 3, 4 and 5. This CNN model uses Dropout of 0.5 for embedding layer and 0.8 hidden units of 100 for the last layer. Moreover, this model use Adam (Kingma & Ba, 2014) optimizer with learning rate 0.001 with beta1 and 2 of 0.9, 0.999. The CNN code is accessible to the public[10].

For neural networks models, all models use early stopping with patience 4. Table 6 shows the details of the batch size in different amount of train set. To avoid a case where a model cannot be trainable where train data set is extremely small such as at 10 instances per class, different batch size is adopted. Also the different iterations for training model between CNN and LM is described in Table 6, where CNN uses 15 epochs for training total dataset while only 5 and 2 epochs are trained in LM models. Since RNN-LSTM requires heavy calculation and takes more time than the CNN model, LM based models run training with small epochs. Despite the fact that this different epoch might cause a bias that CNN model gets more benefit, this paper uses different epoch to cover realistic situations where it is likely the training time is limited. Thus, training time for classifier is limited to less than 24 hours in this paper. Following this time restriction, a large size of train data such as Yelp and Yahoo datasets are trained with small epochs compared to other datasets.

	Batch size (epoch: default 15)							
Instances per class / Models	10	50	100	200	500	1000	2000	Total
CNN	2	2	2	2	20	20	20	256 (5)
CNN(Yelp, Yahoo)	2	2	2	2	20	20	20	256 (2)
LM	2	2	2	2	20	20	20	80 (5)
LM(Yelp, Yahoo)	2	2	2	2	20	20	20	80 (2)

Table 6. The batch sizes of two neural networks models in Convolutional Neural Networks (CNN) and Language Model (LM) for training the classifiers. The batch size differs by the amount of the train data.

[9] http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html [Accessed 10 May 2018]
[10] https://github.com/alexander-rakhlin/CNN-for-Sentence-Classification-in-Keras [Accessed 10 May 2018]

3.5 Obtaining information from the layers

When it comes to a small number of training data, the way to obtain information from neural networks is important. One of the most common ways to obtain information from LSTM layers is selecting a last hidden state. For example, NLP tasks such as sequence to sequence translation (Cho et al., 2014) or simple text classification task with LSTM select the last hidden state for forwarding to next layers. However, this is only possible when the number of training data is bigger than few thousands. Since this paper aims to address how to maximize performance with limited amount of training data, selecting only the last hidden state is not enough to catch detailed information of the pre-trained neural networks. Therefore, this section introduces two methods to obtain the information from all neural network layers.

First of all, selecting multiple last hidden states could be an alternative of obtaining the information from the LSTMs. Since AWD-LSTM uses multiple LSTM layers, the number of last hidden states exist corresponding to the number of LSTM layers. Specifically, in this paper, three LSTM layers are pre-trained so that information could be collected from the three last hidden states. After that, these multiple last hidden states are concatenated and forwarded into linear layer for further steps.

The next strategy to extract information is a mixed pooling methods of average and max pooling outputs of multiple LSTM layers. Since max pooling does not contain informative values but rather become noise to classifiers, this pooling is only employed for final LSTM layer in order to catch sensitivities as a support pooling. Unlike ineffectiveness of max pooling, it was empirically found that the information by average pooling from each LSTM layer increases not only the model's performance but also the stability of the model. Therefore, this mixed pooling method which includes selecting a last hidden state from the last layer is selected as the main pooling method to obtain information from the neural networks layers.

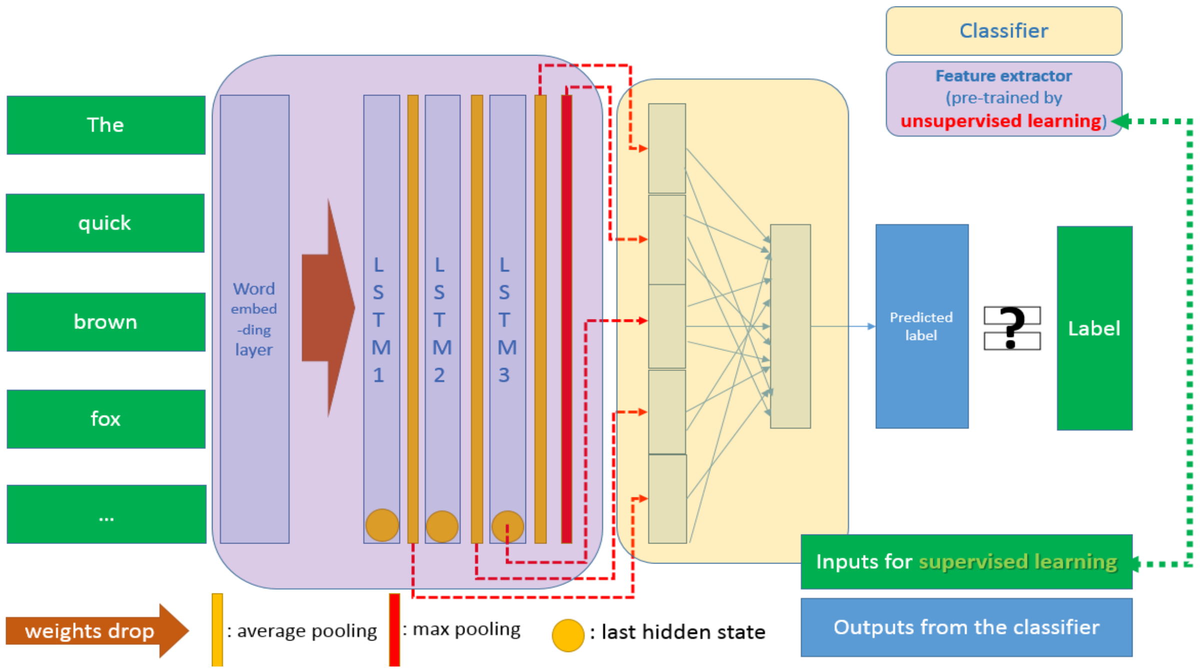


Figure 9. Description of the main neural language model with various pooling method in one picture. The word embedding layer can be initialized separately from other neural network layers. Additional feature extractor layers can also be initialized together only if these layers are pre-trained conjointly. Purple part is the feature extractors which pre-trained with PU data (unsupervised learning), while yellow area refers to the classifier which consists of the concatenated vectors from multiple layers and are fully connected to last linear layer. Only the classifier part is newly trained with labeled data (supervised learning).

To describe the workflow of the pooling process, Figure 9 illustrates the entire training steps from the inputs to outputs of the AWD-LSTM neural network for TC tasks. As purple area indicates feature extractor which consists of three LSTM layers, feature extractor must provide inputs for classifier which is the yellow area with simple linear layer. The input of classifier is obtained from the three LSTM layers by two average pooling from the first two layers and additional max pooling. Moreover, the last hidden state of the last layer is also used to support the mixed pooling method.

3.6 Semi-Transfer Learning: support language models

Initializing one or more pre-trained neural networks layers can be referred to TL. Since the multiple pre-trained LM exist, one can use non-target pre-trained LMs as supports feature extractors. Specifically, when initializing word embedding and the three LSTM layers, which are trained for target task such as classification for the Agnews dataset, we can import one or more non-target pre-trained LMs and parallelize with target LM. To provide a notion of additional LMs as support feature extractors, Figure 10 illustrates a scenario where the multiple pre-trained LMs used for initializing NN. This framework is a generalized version of the framework in Figure 9. The Support LM feature extractor in Figure 10 is frozen layers which differ from LM for target dataset. By freezing entire layers in support LMs, this NN model does not require backpropagation for updating weights. Thus, the training time for this model is similar to a model with only one pre-trained LM.

All the pre-trained LMs contain different information by datasets such as features of a word ‘dog’ will differ by type of pre-trained datasets. However, whether it gives a negative or positive effect, pre-trained LM will provide additional information. In order to inspect how TL affect the performance of SSL, a new method, Semi-Transfer Learning is proposed in this paper. Semi-Transfer Learning is combined by SSL and TL is to observe whether it benefits or hurts model’s performance in which SSL is not available.

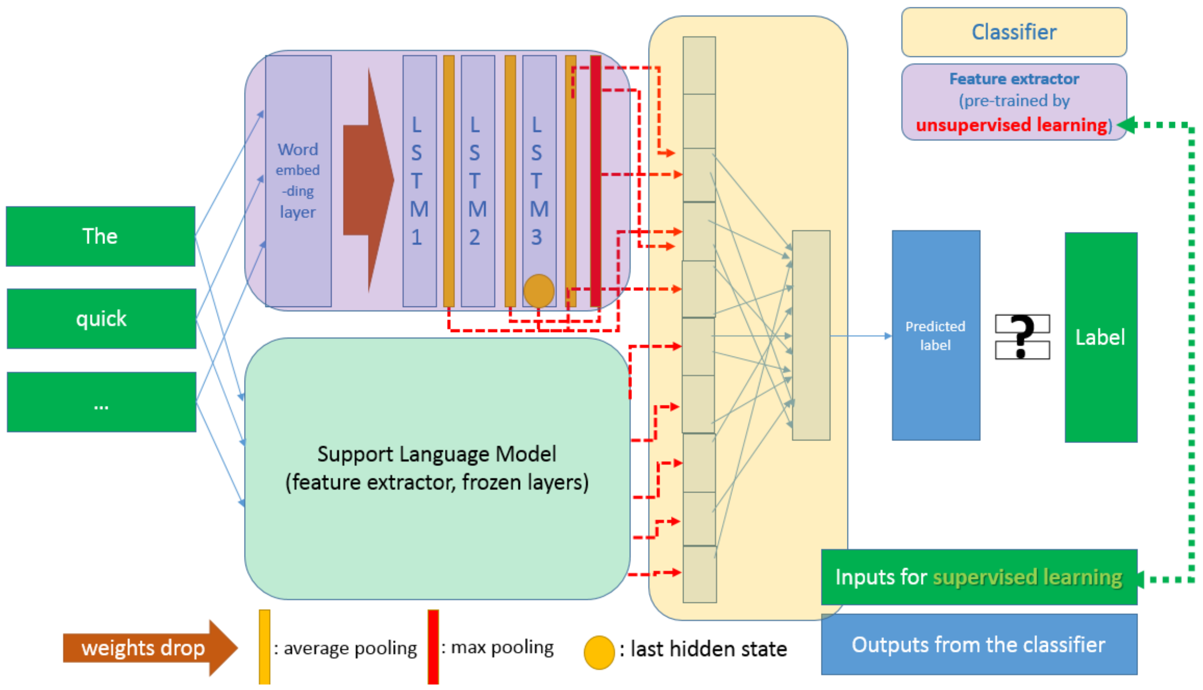


Figure 10. The workflow of the semi-transfer learning where the Support Language Model (LM) is pre-trained with the different datasets from the main LM feature extractor. The support LM is frozen to extract the fixed feature values as a support feature extractor, while the purple area of the main LM is trainable so that it does not give the fixed feature values.

3.7 Evaluating model in neural network

The most common way to evaluate trained neural networks (NN) models is using a validation set which is separated even before training. Since only a split train set is used for training neural networks, validation set is excluded from the training NN models. This separated validation set from train set is not a problem when the size of datasets is big enough such as where the amount of train set is over millions of data. However, when it comes to a small amount of data such as only 100 instances of data exist for train, separating validation and train set becomes problematic. Due to a small size of train data, validation size is extremely small so that the traditional evaluating model on validation set will be highly overfitting on only validation set. Finding the best model which is evaluated by a small validation set provides not only unstable model but also a biased model by overfitting on the validation set. Therefore, appropriate evaluation method must be conducted during the training NN models.

In order to resolve overfitting on small validation dataset, several possible approaches exist. First candidate is k-fold cross-validation (Stone, 1974), which is well studied by many statisticians and used as a evaluating trained model. This evaluation enables a model trained with entire train data without excluding a validation set. Therefore, it can avoid overfitting on the fixed validation set by using the entire train data. However, k-fold cross validation is not practical for training neural networks. This is because k-fold result in training a model at least k times or k*n times for evaluating where n is positive integers. In other words, a model cannot be stopped before exact k*n times training is over. For neural networks training, this is not practical way to train models where it needs to be stopped in the middle of the iterations. For this reason, a simpler and fundamental method for randomly selecting validation set is employed for the experiments.

Larson (1931) has shown a random sampling of a validation set for evaluating the models. This simple random division of samples is the early work of cross-validation, which is origin of the k-fold cross-validation (Stone, 1974). This random sampling has a degree of freedom to stop training a model whenever we need to. For this reason of freedom, the main model of this paper, NLM has been evaluated with the random sampling of a validation set (Larson, 1931).

For the details of evaluation of NN based model by each epoch, Figure 11 illustrates the difference between the common evaluation and random division of train set. (a) in Figure 11 describes the fixed validation is repeatedly used by epoch where (b) is using different validation sets that randomly selected by each epoch. By this random division of train set for evaluating trained models, trained models are expected to be stable when the size of train data is small.

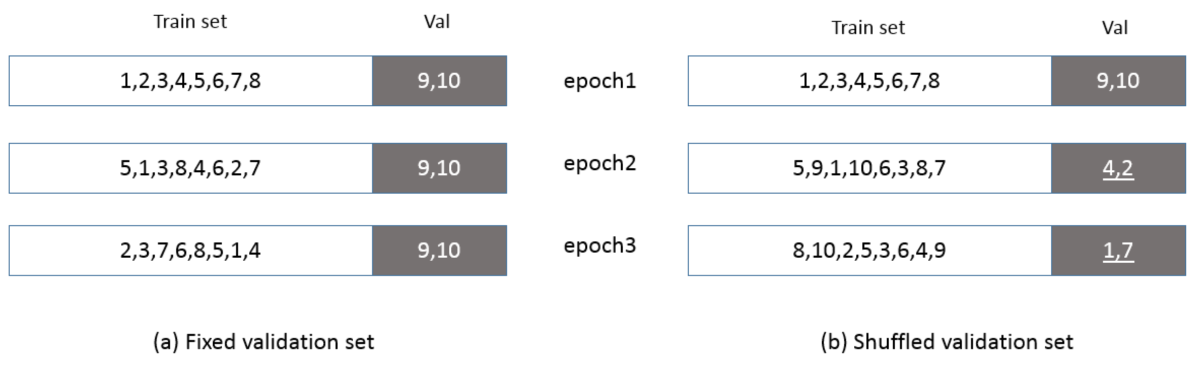


Figure 11. Two methods to sample validation set from train data. (a) is a common situation where uses the fixed validation set for every epoch for evaluating the trained model, whereas sampling validation set in (b) is done by random division of the entire train data by every epoch.

4 Results

This chapter provides the six results from the four experiments, which each experiments provides the performance of the model corresponding to the quantity of train dataset. The first experiment is comparing the four unsupervised learning models in the six datasets to determine the general performance of Language Model (LM). The second and third experiment are inspecting how different pooling methods and a proper evaluation of Neural Networks (NN) affect the model performance. The last experiment is to investigate a case where Semi-Supervised Learning is not available, so that Self-Taught Leaning (STL), Multi-Tasks Learning (MTL) and Semi-Transfer Learning are selected as the alternative learning methods. Consequently, the six results can be derived from the four experiments.

Agnews, 4classes	Tf-idf	FastText	Word2 Vec+ CNN	Wikipedia Self-taught(TL)	DBpedia Multi-tasks(TL)	Target LM + wiki LM (Semi-transfer)	Multi Hidden State Pooling (Semi)	Mixed Pooling (Semi)	Static validation set (Semi, Mixed Pool)
30000	90.36	91.38	91.76	91.24	90.63	91.92	**93.39**	**93.19**	92.8
2000	86.19	85.86	88.99	87	85.99	**91.24**	90.88	90.73	90.66
1000	85.28	82.86	87.92	86.38	85.37	87	90.75	**90.97**	90.97
500	83.34	79.59	85.84	85.12	83.78	86.38	90.09	**90.8**	89.35
200	79.815	71.8	83.01	81.04	80.61	85.12	**88.67**	**88.32**	87.8
100	76.315	63	81.78	79.7	70.37	81.04	75.52	**84.57**	83.9
50	71.15	54	75.28	68.52	70.76	79.7	**86.04**	**85.08**	79.96
10	48.78	30	55.84	66.92	53.39	68.52	66.4	**71.88**	68.87

Table 7. Accuracies of four different unsupervised learning models in Agnews dataset. Each of Tf-idf and FastText use Logistic and Softmax functions for a classifier where Word2Vec model is jointly working with Convolutional Neural Network classifier. All Language Models (LMs) from the 4-9 columns use linear layers as a Softmax classifier. The grey area indicates LM based experiments. Columns 4-6 provide a comparison of Self-taught, Multi-Tasks and Semi-transfer learning. Keeping column orders, Wikipedia, DBpedia datasets are used for Self-taught and Multi-Tasks learning in transfer learning while both Wikipedia and target task LM are used for Semi-transfer learning. Columns 7-8 uses different pooling methods to obtain information from layers while column 8-9 use different evaluation during the training stage.

DBpedia (10classes)	Tf-idf	FastText	CNN	LM
40000	**97.81**	**98.53**	**98.44**	98.37
2000	95.77	96.9	96.61	97.71
1000	94.92	96.08	95.93	97.86
500	93.95	94.54	94.59	98.15
200	92.39	91.90	89.97	97.18
100	90.99	88.79	88.23	96.96
50	88.99	82.42	79.79	**97.29**
10	77.56	36.65	38.05	88.71

Table 8. Accuracies of the four different unsupervised learning models in DBpedia dataset

The first result is that LM feature extractor outperforms the other three models. As Table 7 shows the performances of the four models in proportion to the quantity of training data in Agnews dataset, the LM attains higher accuracies than the other three models in general. When the number of labeled data is small, employing the LM gives more advantages. For instance, 10 instances per class are trained for the classifiers in Table 7, the LM has the accuracy of 71.88% while other three models' accuracies are under 50%. In other words, using the LM as a feature extractor in a small quantity of training classifier increases the performance approximately 50% in Agnews datasets. However, such datasets as DBpedia and Amazon six categories, it does not give 50% of increased performance while the LM still outperforms the other three models. For the DBpedia tasks described in Table 8, training classifiers with 50 instances per class with the LM already reaches same level as using full training datasets from the other three models.

The next finding from the experiments is that the model based on LM generally performs better than the others in the unbalanced-data tasks. Table 9 illustrates the performances of the four models in the Amazon Six Categories dataset which is an imbalanced dataset. The performance of the LM in Table 9 give us a notion that the LM catch the sensitive information better than the other models. To be specific, the LM outperforms the rest of models in general when over 50 instances per class are used for a training. In addition to the Amazon Six Categories dataset, the results of TripAdvisor Hotel Reviews dataset are described in Appendix B.

	Amazon-LAPTOPS, total instance:1655 F-1 score				Amazon-Video, total instance:2376 F-1 score			
Instance per class (6class)	Tf-idf	FastText	CNN	LM	Tf-idf	FastText	CNN	LM
Full (max: 29716)	0.85	0.86	**0.88**	0.86	0.73	0.75	0.**79**	0.78
2000	0.78	0.73	0.78	**0.80**	0.65	0.66	0.71	**0.71**
1000	0.76	0.70	0.76	**0.80**	0.65	0.64	0.69	**0.73**
500	0.75	0.61	0.76	0.75	0.64	0.60	0.68	**0.74**
200	**0.74**	0.58	0.73	0.72	0.62	0.52	0.62	**0.71**
100	0.70	0.35	0.54	**0.72**	0.61	0.36	0.62	**0.69**
50	0.65	0.27	0.27	**0.70**	0.59	0.24	0.28	**0.72**
10	**0.49**	0.01	0.03	0.39	**0.45**	0.01	0.00	0.44

Table 9. Performances in two classes of Amazon Six categories imbalanced datasets, which two classes account for smallest proportions.

The third result is that different pooling methods affect the performance of the LM when a small quantity of data used for a training classifier. Table 7 illustrates that different pooling methods play important roles when the number of labeled data for a training is extremely small such as 10 instance per class. The 7-8 columns in Table 7 shows that performances of the two different pooling methods within the same conditions of hyperparameters. According to Table 7, when 10 and 100 instances per class are used for a training, the accuracies of mixed pool are 5.5% to maximum 9% higher than multi-hidden pool. Moreover, the model with mixed pool seems more stable since variance of the accuracy is smaller than the other pooling method. However, there is no differences of the performance when over 500 instances per class are used for a training.

The other observation is that evaluating methods for a trained NN model are important. The last two columns of Table 7 describe the performances of different evaluations to find best NN models. Specifically, first of the last columns uses shuffled validation sets which is randomly sampled from trainsets by every epoch, whereas last column uses fixed validation dataset for all epochs. In other words, Table 7 shows that the random sampling of validation set by every epoch helps evaluation to find a better NN model. This random division of train and validation set tends to give more help when the number of instance per class are smaller.

Finally, the results show that the performance of Transfer Learning (TL) is not comparable to semi-supervised learning but the performance of TL still comparable to the performance of the other three models. For example, the first three columns of the grey area in Table 7 demonstrate that the performances of Self-taught, Multi-Tasks and Semi-Transfer Learning are similar to Tf-idf, FastText and Word2Vec models. Specifically, the result shows Semi-Transfer Learning does not perform better than Semi-Transfer Learning but rather gives *negative transfer* effect to the model. Furthermore, it is shown that STL outperforms MTL where the domain of STL is wider than the domain of MTL.

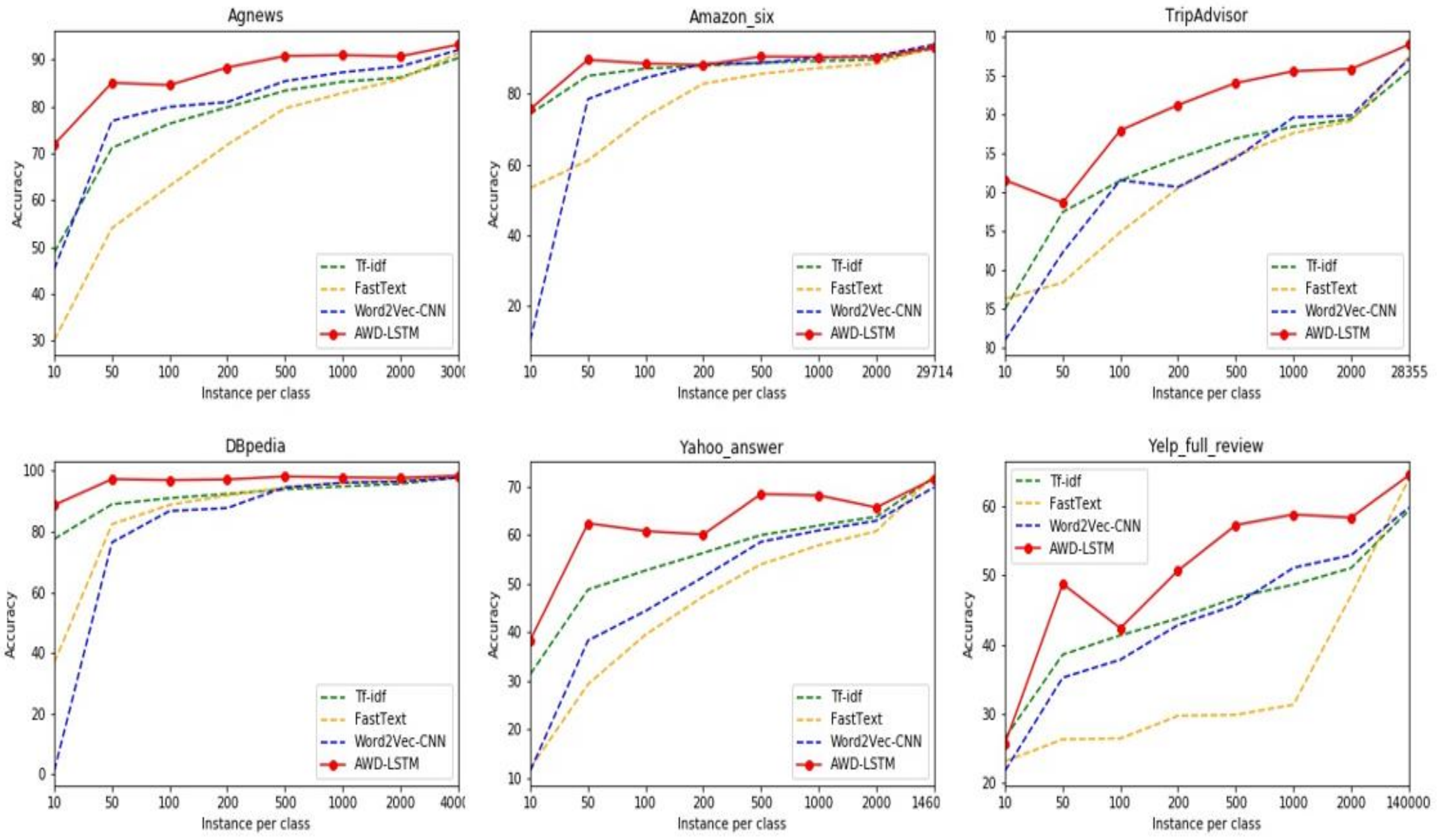


Figure 12. Accuracies of the four unsupervised learning models in the six datasets. The details are described in the Appendix A.

5 Discussion

This chapter provides a discussion of the results which show that Language Model (LM) generally outperforms the other models. The discussion can be summarized into three topics. The first subject is the factors of outperformance of the LM, which is the main interest of this paper. The second subject is an effectiveness of evaluating methods for Neural Networks (NN) models. The final topic is related to advantages of Transfer Learning (TL). Further discussions for improving the main model of this paper will be covered in the chapter 'Future work'.

First of all, the reason why the LM outperforms the other models can be derived from the number of pre-trained parameters. Unlike the pre-trained word vectors models such as Word2Vec (Le & Mikolov, 2014) and FastText (Bojanowski et al., 2016), the main model of this paper, AWD-LSTM (Merity et al., 2017) has extra three pre-trained LSTM layers as a feature extractor. In other words, this paper provides the comparisons between pre-trained feature and pre-trained feature extractor. This comparison might seem unfair to compare with each other. However, this unfairness is compromised by employing linear and non-linear classifiers on top of the pre-trained features. Furthermore, it is not guaranteed that the pre-trained feature extractor performs better than the pre-trained features. In fact, it could be a case where the feature extractor to generate noises to a classifier rather than give informative feature values to a classifier, which is the *negative transfer* effect. To summarize, the factor of high number of parameters allows the LM outperforms the other three models.

The second topic is related to efficient evaluation approaches for NN models for a small number of training data. The experimental results describe that the early type of cross-validation evaluation (Larson, 1931) allows us to find a better model during training NN. This result is expected since the early type of cross-validation is a compromised version of k-fold cross-validation. Since k-fold cross-validation cannot be adopted for NN models with other regularizing techniques such as early stopping in NN, random division of train and validation set is an alternative option for small training data in terms of training NN models. Briefly, by employing the random sampling for validation set by every epoch, it is possible to find a better model in a small amount of train data.

The final argument of this chapter is the effectiveness of TL. Despite the advantages of the TL for building feature extractors with unlimited data, which are shown by Do & Ng (2005) and Raina et al. (2007), the TL does not always improve performances of the models but rather worsen the performance, which is known as the *negative transfer* (Ben-David & Schuller, 2003). However, the experimental results has already shown that the performances of the TL are comparable to semi-supervised learning in spite of types of the TL. This results gives us a notion that the TL with exploiting any unlabeled data can achieve the same performance of Semi-Supervised Learning with collected (positive) and unlabeled (Positive and Unlabeled) data. In conclusion, a model based on TL with non-task (Negative and Unlabeled) data can replace the other three models with task (Positive and Unlabeled) data.

6 Conclusions

This thesis has shown that a model based on Language Model (LM) outperforms the other three unsupervised models for Text Classification (TC), which the other three models are based on Tf-idf and pre-trained word vectors. It is also found that a proper evaluating methods for trained Neural Networks (NN) models and a particular approach to obtain information from NN improve the performance of the LM. Moreover, the results show that the performance of the LM with Transfer Learning (TL) is comparable to the performance of Semi-Supervised Learning. To summarize, this paper gives the five following conclusions.

First of all, LM as a feature extractor outperforms the other unsupervised learning methods such as Tf-idf and the pre-trained word-vector models. When only limited labeled data are available, using LM is the best choice to exploit unlabeled data. For example, the performance of a model trained with 500 instances per class is comparable to the one with 30k instances per a class with Tf-idf model in Agnews dataset, which is described in Table 7.

Secondly, obtaining information from all NN layers improves the performance of the classifier. Different pooling methods to extract information from neural network layers affect performance in a small quantity of train data. However, when it comes to considerably big data being used for a training classifier, the pooling method does not give significant benefits.

Thirdly, the LM model is able to catch sensitive information for difficult classification tasks such as unbalanced-data tasks. As the two unbalanced datasets (H. Wang, Lu, & Zhai, 2010, 2011) were set up for simulating realistic situations, in the results, it is presented that LM generally performs better than the others. To be specific, approximately 100 instances per class is good enough for the LM model to be stable.

The next conclusion is that shuffling training and validation datasets by every epoch is recommended for NN models. Given a small number of data for training classifiers, all train data must be used for training by shuffling train and validation set by every epoch instead of fixing validation set. This random sampling for validation set enables us to find stable models. In fact, this shuffling method is a compromised version of the k-fold cross-validation, which is another way to exploit all trainset data for a training NN models.

Finally, this paper concludes that TL is a key to build strong feature extractors when Semi-Supervised Learning (SSL) is not available within LM. From the results of the experiments, it has been clearly shown that the performance of TL with LM is similar to the performance of the other three models of Tf-idf, and two word-vector models. Therefore, using TL with LM is encouraged to employ to build a model when SSL is not available.

To summarize these five conclusions, using LM for unsupervised learning is recommended to achieve maximum accuracy for Text Classification when only a small number of labeled data is available. The LM outperforms the other unsupervised learning models in both easy and difficult tasks with imbalanced dataset. Moreover, the LM gives a freedom of degree to exploit any data types such as negative and unlabeled (NU) data. For these reasons, building feature extractors with LM for TC seems reasonable choice over choosing the other models which are based on Tf-idf and pre-trained word vectors.

7 Future work

In addition to the works described in the previous sections, more possible studies exist to improve performance of the main model of this paper. There are three aspects to increase the model performance. The first aspect is speeding up backpropagation calculation for updating weights in neural networks. The another view is to optimize parameters to find a best model, which is known as Fine-tuning in Transfer Learning (TL). The final way to increase a model performance is to evaluate Neural Networks (NN) model by batch size level rather than every epoch. Three possible works are list by orders.

First of all, efficient backpropagation approaches could be studied for training Neural Language Model (NLM). Since this NLM uses Softmax function for classifiers which cause heavy calculations during a training NLM, other efficient methods to replace Softmax could be implemented for NLM. For example, Adaptive Softmax (Grave et al., 2016) which is efficient Softmax approximation for GPU will reduce the training time of NLM.

The second possible research is Fine-tuning in TL, which is optimization technique for hyperparameters of NN. When pre-trained models are used to initialize weights of neural networks for a different task, not only proper values of hyperparameters are important but methods how to use hyperparameters are also crucial. For example, Howard & Ruder (2018) address that TL with their Fine-tuning approaches on top of AWD-LSTM (Merity et al., 2017). And then, they show that their methods reach the state-of-the-art performance for not only TC but also other Natural Language Processing (NLP) tasks. In conclusion, optimization for NN by Fine-tuning is open research to improve the performance of TL.

Finally, different evaluating methods in NN can be studied for Semi-Supervised Learning (SSL) and TL. Since training Machine Learning (ML) models with a small number of train data is likely to have overfitting in early training stage, stopping training during an epoch must be considered to find a better model. For example, changes of weights of NN are too sensitive so that training must be stopped after training 20 batches during the middle of an epoch to find the best model. For this sensitivity issue, studying different ways on how and when to trigger evaluating the trained model during one iteration remains as a future work.

References

Allahyari, M., Pouriyeh, S., Assefi, M., Safaei, S., Trippe, E. D., Gutierrez, J. B., & Kochut, K. (2017). A Brief Survey of Text Mining: Classification, Clustering and Extraction Techniques. *ArXiv:1707.02919 [Cs]*. Retrieved from http://arxiv.org/abs/1707.02919

Beel, J., Gipp, B., Langer, S., & Breitinger, C. (2016). Research-paper recommender systems: a literature survey. *International Journal on Digital Libraries*, *17*(4), 305–338. https://doi.org/10.1007/s00799-015-0156-0

Ben-David, S., & Schuller, R. (2003). Exploiting task relatedness for multiple task learning. In *Learning Theory and Kernel Machines* (pp. 567–580). Springer.

Bengio, Y., Simard, P., & Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, *5*(2), 157–166. https://doi.org/10.1109/72.279181

Bengio, Yoshua, Ducharme, R., Vincent, P., & Jauvin, C. (2003). A Neural Probabilistic Language Model. *Journal of Machine Learning Research*, *3*(Feb), 1137–1155.

Bojanowski, P., Grave, E., Joulin, A., & Mikolov, T. (2016). Enriching Word Vectors with Subword Information. *ArXiv:1607.04606 [Cs]*. Retrieved from http://arxiv.org/abs/1607.04606

Caruana, R. (1998). Multitask Learning. In *Learning to Learn* (pp. 95–133). Springer, Boston, MA. https://doi.org/10.1007/978-1-4615-5529-2_5

Chapelle, O., Schölkopf, B., & Zien, A. (2006). *Semi-Supervised Learning (Adaptive Computation and Machine Learning)*. The MIT Press.

Cho, K., van Merrienboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014). Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. *ArXiv:1406.1078 [Cs, Stat]*. Retrieved from http://arxiv.org/abs/1406.1078

Cox, D. R. (1958). The Regression Analysis of Binary Sequences. *Journal of the Royal Statistical Society. Series B (Methodological)*, *20*(2), 215–242.

Dayan, P., Sahani, M., & Deback, G. (1999). Unsupervised learning. *The MIT Encyclopedia of the Cognitive Sciences*.

Do, C. B., & Ng, A. Y. (2005). Transfer Learning for Text Classification. In *Proceedings of the 18th International Conference on Neural Information Processing Systems* (pp. 299–306). Cambridge, MA, USA: MIT Press. Retrieved from http://dl.acm.org/citation.cfm?id=2976248.2976286

Fukushima, K. (1988). Neocognitron: A hierarchical neural network capable of visual pattern recognition. *Neural Networks*, *1*(2), 119–130. https://doi.org/10.1016/0893-6080(88)90014-7

Grave, E., Joulin, A., Cissé, M., Grangier, D., & Jégou, H. (2016). Efficient softmax approximation for GPUs. *ArXiv:1609.04309 [Cs]*. Retrieved from http://arxiv.org/abs/1609.04309

Graves, A., & Schmidhuber, J. (2005). Framewise phoneme classification with bidirectional LSTM and other neural network architectures. *Neural Networks*, *18*(5–6), 602–610. https://doi.org/10.1016/j.neunet.2005.06.042

Harris, Z. S. (1954). Distributional Structure. WORD, *10*(2–3), 146–162. https://doi.org/10.1080/00437956.1954.11659520

Hayes, B. (2013). First Links in the Markov Chain. *American Scientist*, *101*(2), 92. https://doi.org/10.1511/2013.101.92

Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. *ArXiv:1207.0580 [Cs]*. Retrieved from http://arxiv.org/abs/1207.0580

Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, *9*(8), 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735

Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, *79*(8), 2554–2558. https://doi.org/10.1073/pnas.79.8.2554

Howard, J., & Ruder, S. (2018). Fine-tuned Language Models for Text Classification. *ArXiv:1801.06146 [Cs, Stat]*. Retrieved from http://arxiv.org/abs/1801.06146

Johnson, R., & Zhang, T. (2016). Convolutional Neural Networks for Text Categorization: Shallow Word-level vs. Deep Character-level. *ArXiv:1609.00718 [Cs, Stat]*. Retrieved from http://arxiv.org/abs/1609.00718

Johnson, R., & Zhang, T. (2017). Deep Pyramid Convolutional Neural Networks for Text Categorization. *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, *1*, 562–570. https://doi.org/10.18653/v1/P17-1052

Jones, K. S. (1972). A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation*, *28*(1), 11–21. https://doi.org/10.1108/eb026526

Jones, K. S. (1973). Index term weighting. *Information Storage and Retrieval*, *9*(11), 619–633. https://doi.org/10.1016/0020-0271(73)90043-0

Joulin, A., Grave, E., Bojanowski, P., & Mikolov, T. (2016). Bag of Tricks for Efficient Text Classification. *ArXiv:1607.01759 [Cs]*. Retrieved from http://arxiv.org/abs/1607.01759

Kim, Y. (2014a). Convolutional Neural Networks for Sentence Classification. *ArXiv:1408.5882 [Cs]*. Retrieved from http://arxiv.org/abs/1408.5882

Kim, Y. (2014b). Convolutional Neural Networks for Sentence Classification. *ArXiv:1408.5882 [Cs]*. Retrieved from http://arxiv.org/abs/1408.5882

Kingma, D. P., & Ba, J. (2014). Adam: A Method for Stochastic Optimization. *ArXiv:1412.6980 [Cs]*. Retrieved from http://arxiv.org/abs/1412.6980

Ko, Y., & Seo, J. (2000). Automatic Text Categorization by Unsupervised Learning. In *Proceedings of the 18th Conference on Computational Linguistics - Volume 1* (pp. 453–459). Stroudsburg, PA, USA: Association for Computational Linguistics. https://doi.org/10.3115/990820.990886

Lang, K. (1995). Newsweeder: Learning to filter netnews. In *Proceedings of the Twelfth International Conference on Machine Learning* (pp. 331–339).

Larson, S. C. (1931). The shrinkage of the coefficient of multiple correlation. *Journal of Educational Psychology*, *22*(1), 45–55. http://dx.doi.org.ezproxy.ub.gu.se/10.1037/h0072400

Le, Q. V., & Mikolov, T. (2014). Distributed Representations of Sentences and Documents. *ArXiv:1405.4053 [Cs]*. Retrieved from http://arxiv.org/abs/1405.4053

LeCun, Y., Haffner, P., Bottou, L., & Bengio, Y. (1999). Object Recognition with Gradient-Based Learning. In D. A. Forsyth, J. L. Mundy, V. di Gesú, & R. Cipolla, *Shape, Contour and Grouping in Computer Vision* (Vol. 1681, pp. 319–345). Berlin, Heidelberg: Springer Berlin Heidelberg. https://doi.org/10.1007/3-540-46805-6_19

Li, X.-L., & Liu, B. (2005). Learning from Positive and Unlabeled Examples with Different Data Distributions. In *Machine Learning: ECML 2005* (pp. 218–229). Springer, Berlin, Heidelberg. https://doi.org/10.1007/11564096_24

Little, W. A. (1974). The Existence of Persistent States in the Brain. In *From High-Temperature Superconductivity to Microminiature Refrigeration* (pp. 145–164). Springer, Boston, MA. https://doi.org/10.1007/978-1-4613-0411-1_12

Markov, A. A. (1953). The Theory of Algorithms. *Journal of Symbolic Logic*, *18*(4), 340–341.

Matykiewicz, P., & Pestian, J. (2012). Effect of Small Sample Size on Text Categorization with Support Vector Machines. In *Proceedings of the 2012 Workshop on Biomedical Natural Language Processing* (pp. 193–201). Stroudsburg, PA, USA: Association for Computational Linguistics. Retrieved from http://dl.acm.org/citation.cfm?id=2391123.2391149

Merity, S., Keskar, N. S., & Socher, R. (2017). Regularizing and Optimizing LSTM Language Models. *ArXiv:1708.02182 [Cs]*. Retrieved from http://arxiv.org/abs/1708.02182

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient Estimation of Word Representations in Vector Space. *ArXiv:1301.3781 [Cs]*. Retrieved from http://arxiv.org/abs/1301.3781

Mikolov, T., Sutskever, I., Chen, K., Corrado, G., & Dean, J. (2013). Distributed Representations of Words and Phrases and their Compositionality. *ArXiv:1310.4546 [Cs, Stat]*. Retrieved from http://arxiv.org/abs/1310.4546

Mirończuk, M. M., & Protasiewicz, J. (2018). A recent overview of the state-of-the-art elements of text classification. *Expert Systems with Applications*, *106*, 36–54. https://doi.org/10.1016/j.eswa.2018.03.058

Mou, L., Meng, Z., Yan, R., Li, G., Xu, Y., Zhang, L., & Jin, Z. (2016). How Transferable are Neural Networks in NLP Applications? *ArXiv:1603.06111 [Cs]*. Retrieved from http://arxiv.org/abs/1603.06111

Nigam, K., Mccallum, A. K., Thrun, S., & Mitchell, T. (2000). Text Classification from Labeled and Unlabeled Documents using EM. *Machine Learning*, *39*(2–3), 103–134. https://doi.org/10.1023/A:1007692713085

Pan, S. J., & Yang, Q. (2010). A Survey on Transfer Learning. *IEEE Transactions on Knowledge and Data Engineering*, *22*(10), 1345–1359. https://doi.org/10.1109/TKDE.2009.191

Pratt, L. Y. (1993). Discriminability-Based Transfer between Neural Networks. In S. J. Hanson, J. D. Cowan, & C. L. Giles (Eds.), *Advances in Neural Information Processing Systems 5* (pp. 204–211). Morgan-Kaufmann. Retrieved from http://papers.nips.cc/paper/641-discriminability-based-transfer-between-neural-networks.pdf

Raina, R., Battle, A., Lee, H., Packer, B., & Ng, A. Y. (2007). Self-taught learning: transfer learning from unlabeled data (pp. 759–766). ACM Press. https://doi.org/10.1145/1273496.1273592

Salton, G., Wong, A., & Yang, C. S. (1975). A vector space model for automatic indexing. *Communications of the ACM*, *18*(11), 613–620. https://doi.org/10.1145/361219.361220

Sathasivam, S., & Abdullah, W. A. T. W. (2008). Logic Learning in Hopfield Networks. *ArXiv:0804.4075 [Cs]*. Retrieved from http://arxiv.org/abs/0804.4075

Schuster, M., & Paliwal, K. K. (1997). Bidirectional Recurrent Neural Networks. *IEEE TRANSACTIONS ON SIGNAL PROCESSING*, *45*(11), 9.

Scudder, H. (1965). Probability of error of some adaptive pattern-recognition machines. *IEEE Transactions on Information Theory*, *11*(3), 363–371.

Shannon, C. E. (1948). A Mathematical Theory of Communication, 55.

Socher, R., Bauer, J., Manning, C. D., & Andrew Y., N. (2013). Parsing with Compositional Vector Grammars. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)* (pp. 455–465). Sofia, Bulgaria: Association for Computational Linguistics. Retrieved from http://www.aclweb.org/anthology/P13-1045

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, *15*, 1929–1958.

Stone, M. (1974). Cross-Validatory Choice and Assessment of Statistical Predictions. *Journal of the Royal Statistical Society. Series B (Methodological)*, *36*(2), 111–147.

Tao, D., Xu, Y., Xu, C., & Xu, C. (2017). Multi-Positive and Unlabeled Learning, 3182–3188.

Turney, P. D., & Pantel, P. (2010). From Frequency to Meaning: Vector Space Models of Semantics. *ArXiv:1003.1141 [Cs]*. https://doi.org/10.1613/jair.2934

Walker, S. H., & Duncan, D. B. (1967). Estimation of the probability of an event as a function of several independent variables. *Biometrika*, *54*(1–2), 167–179. https://doi.org/10.1093/biomet/54.1-2.167

Wan, L., Zeiler, M., Zhang, S., LeCun, Y., & Fergus, R. (2013). Regularization of Neural Networks Using Dropconnect. In *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28* (pp. III–1058–III–1066). Atlanta, GA, USA: JMLR.org. Retrieved from http://dl.acm.org/citation.cfm?id=3042817.3043055

Wang, H., Lu, Y., & Zhai, C. (2010). Latent Aspect Rating Analysis on Review Text Data: A Rating Regression Approach. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 783–792). New York, NY, USA: ACM. https://doi.org/10.1145/1835804.1835903

Wang, H., Lu, Y., & Zhai, C. (2011). Latent Aspect Rating Analysis Without Aspect Keyword Supervision. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 618–626). New York, NY, USA: ACM. https://doi.org/10.1145/2020408.2020505

Wang, Y., Zhou, Z., Jin, S., Liu, D., & Lu, M. (2017). Comparisons and Selections of Features and Classifiers for Short Text Classification. *IOP Conference Series: Materials Science and Engineering*, *261*(1), 012018. https://doi.org/10.1088/1757-899X/261/1/012018

Zaremba, W., Sutskever, I., & Vinyals, O. (2014). Recurrent Neural Network Regularization. *ArXiv:1409.2329 [Cs]*. Retrieved from http://arxiv.org/abs/1409.2329

Zechner, N. (2013). The Past, Present and Future of Text Classification. In *2013 European Intelligence and Security Informatics Conference* (pp. 230–230). https://doi.org/10.1109/EISIC.2013.61

Appendix A: Accuracy for six datasets

The following tables shows the performances of the four unsupervised learning models for the six datasets. Specifically, the Tf-idf (Term-frequency Inverse Document Frequency) model uses a Tf-idf weight as a feature for a Logistic Regression (LR) classifier, where the FastText model uses word-vectors as features of word and Softmax regression as a classifier. For the neural networks models, the Word2Vec model conducts with a Convolutional Neural Networks (CNN) classifier and the Language Model (LM) model uses a simple linear classifier. The below tables give a notion that LM outperforms the other three models in general.

Agnews (4 classes)	Tf-idf+LR	FastText	Word2Vec+CNN	LM
30000	90.36	91.38	91.76	**93.19**
2000	86.19	85.86	88.99	**90.73**
1000	85.28	82.86	87.92	**90.97**
500	83.34	79.59	85.84	**90.8**
200	79.82	71.8	92.01	**88.32**
100	76.32	63	81.78	**84.57**
50	71.15	54	75.28	**85.08**
10	48.78	30	55.84	**71.88**

DBpedia (10 classes)	Tf-idf+LR	FastText	Word2Vec+CNN	LM
40000	97.81	**98.53**	98.44	98.37
2000	95.77	96.9	96.61	**97.71**
1000	94.92	96.08	95.93	**97.86**
500	93.95	94.54	94.59	**98.15**
200	92.39	91.90	89.97	**97.18**
100	90.99	88.79	88.23	**96.96**
50	88.99	82.42	79.79	**97.29**
10	77.56	36.65	38.05	**88.71**

Amazon6 (6 classes)	Tf-idf+LR	FastText	Word2Vec+CNN	LM
29716 (Average)	92.76	93.29	**94.07**	93.11
2000	89.68	88.5	**90.47**	90.38
1000	89.25	87.25	89.68	**90.43**
500	88.78	85.63	88.97	**90.5**
200	87.87	82.84	85.46	**88.1**
100	87.08	73.53	83.91	**88.54**
50	85.08	61.18	64.82	**89.59**
10	74.33	53.41	21.34	**75.67**

Hotel Reviews (5 classes)	Tf-idf+LR	FastText	Word2Vec+CNN	LM
28360 (Average)	65.53	67.34	67.47	**68.9**
2000	59.36	59.1	61.73	**65.79**
1000	58.37	57.53	60.20	**65.49**
500	56.84	54.60	56.93	**63.98**
200	54.29	50.44	54.56	**61.15**
100	51.42	44.79	45.20	**57.89**
50	47.37	38.29	37.96	**48.6**
10	34.9	36.23	23.50	**51.48**

Yahoo Answers (10 classes)	Tf-idf+LR	FastText	Word2Vec+CNN	LM
140k	**71.92**	71.14	71.53	71.62
2000	63.77	60.81	65.24	**65.69**
1000	61.95	57.88	63.96	**68.17**
500	60	53.97	59.82	**68.45**
200	56.31	47.34	52.9	**60.13**
100	52.69	39.47	47.15	**60.82**
50	48.74	29.24	30.68	**62.41**
10	31.29	11.9	11.45	**38.34**

Yelp Reviews (5 classes)	Tf-idf+LR	FastText	Word2Vec+CNN	LM
130k	59.35	62.14	60.58	**64.44**
2000	51.06	47.12	52.99	**58.35**
1000	48.68	31.36	44.65	**58.8**
500	46.80	29.89	42.61	**57.26**
200	43.82	29.75	34.95	**50.74**
100	41.36	26.48	29.48	**42.4**
50	38.58	26.35	23.12	**48.78**
10	**26.55**	23.2	20	25.72

Appendix B: F-1 Score for the imbalanced datasets

Performances of the four unsupervised learning models in TripAdvisor Hotel Reviews datasets:

	TripAdvisor-score 1.0 total instance:2374 F-1 score				TripAdvisor-score 2.0 total instance:2515 F-1 score			
Instance per class (6class)	Tf-idf	FastText	CNN	LM	Tf-idf	FastText	CNN	LM
Full (max: 28360)	0.71	0.72	0.72	**0.74**	0.32	**0.48**	0.45	**0.48**
2000	0.64	0.67	0.70	**0.71**	0.39	0.45	0.38	**0.50**
1000	0.63	0.65	0.68	**0.72**	0.35	0.44	0.37	**0.50**
500	0.61	0.61	0.67	**0.72**	0.32	0.43	0.38	**0.48**
200	0.58	0.56	0.62	**0.69**	0.30	0.38	0.24	**0.40**
100	0.57	0.46	0.53	**0.69**	0.32	0.31	**0.33**	**0.33**
50	**0.50**	0.34	**0.50**	0.36	0.25	0.16	**0.18**	0.11
10	0.36	0.22	0.14	**0.62**	**0.22**	0.02	0	0.11

Where the datasets consist of unbalanced data where two classes account for the smallest proportion of the entire amount of data. Since accuracy does not represent fair performance in imbalanced data, F-1 score is used for evaluating the four models.