Learning Distributed Document Representations for Multi-label Document Categorization

A Thesis Submitted

in Partial Fulfillment of the Requirements

for the Degree of

B.Tech. - M.Tech. (Dual Degree)

by

Nitish Gupta

to the

DEPARTMENT OF ELECTRICAL ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY,KANPUR

May 2015

CERTIFICATE

It is certified that the work contained in the thesis entitled "*Learning Distributed Document Representations for Multi-label Document Categorization*" by *Nitish Gupta (10327461)* has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

May 2015

Dr. Harish Karnick
Professor,
Department of Computer Science and Engineering
Indian Institute of Technology,
Kanpur-208016.

Dr. Rajesh M. Hegde
Associate Professor,
Department of Electrical Engineering,
Indian Institute of Technology,
Kanpur-208016.

Abstract

Multi-label Document Categorization, the task of automatically assigning a text document into one or more categories has various real-world applications such as categorizing news articles, tagging Web pages, maintaining medical patient records and organizing digital libraries among many others. Statistical Machine Learning approaches to document categorization have focused on multi-label learning algorithms such as Support Vector Machines, k-Nearest Neighbors, Logistic Regression, Neural Networks, Naive Bayes, Generative Probabilistic Models etc. while the input to such algorithms i.e. the vector representation for documents has traditionally been used as the bag-of-words model. Though the usage of simple bag-of-words representation gives surprisingly accurate results, it suffers from sparsity, high-dimensionality, lack of similarity measures along with other drawbacks such as the inability to encode word ordering and contextual information in which the words occur. Encoding contextual information about words in documents is crucial to capture the correct semantic content of the highly complex and ambiguous human language.

Our work is focused on learning continuous distributed vector representations for documents by embedding all the documents in the same low-dimensional space such that documents that are similar in their semantic content have similar vector representations. To tackle the issues in bag-of-words representation model, we present an unsupervised neural network model that uses the document vector to predict words in the document along with using the contextual information in which the word occurs and jointly learns distributed document and word representations. We use a modified version of the logistic regression algorithm to learn similar distributed representations for categories to perform the document categorization task. We show that our model gives state-of-the-art results on the standard *Reuters-21578* dataset, improving the bag-of-words model by 9% and previous state-of-the-art by 3.26% in terms of the F1 Score. We also show the effectiveness of our model in imputing missing categories on the Wikipedia articles against the bag-of-words representations. As we embed documents, categories and words in the same low-dimensional space our model can also estimate semantic similarities between them. We qualitatively demonstrate that the learned representations capture the semantic dependencies between categories and words which is not directly observed in the data.

Dedicated to
My Family

Acknowledgement

I would like to express my sincere gratitude towards my thesis supervisor, Prof. Harish Karnick, for his constant support and encouragement. I am grateful for his patient guidance and advice in giving a proper direction to my efforts. I am also grateful to Prof. Rajesh M. Hegde for giving me the freedom to work on a topic of my liking.

Last, but not the least, I would like to thank my parents and brother for their love and encouragement. Without their support and patience this work would not have been possible.

Nitish Gupta

Contents

Abstract **i**

List of Tables **xi**

List of Figures **xiii**

List of Algorithms **xv**

1 Introduction **1**

1.1 Motivation 3

1.1.1 Inability to preserve word ordering 3

1.1.2 Lack of similarity measures 4

1.1.3 Compositionality of distributed word vectors 4

1.2 Problem Statement 5

1.3 Organization of Thesis 6

2 Background on Document Categorization **7**

2.1 Text Representation 7

2.1.1 Bag of Words 8

2.1.2 Dimensionality Reduction and Feature Selection 9

2.2 Learning Algorithms 11

2.2.1 Document Categorization using Binary Classifiers 12

2.2.2 Document Categorization with Single Joint Classifier 14

3 Distributed Document Representations **17**

3.1 Need for Distributed Word Representations 17

3.2 Background on Word Embeddings 18

3.2.1 Neural Probabilistic Language Model 19

3.2.2 Log-Linear Models 20

3.2.2.1 Continuous Bag-of-Words Model 21

3.2.2.2 Continuous Skip-gram Model 22

3.2.2.3 Dependency-based Word Embeddings 23

3.3 Document Representation 24

3.3.1 Problem Setup 25

3.3.2 Our Model 25

3.3.2.1 Context Representation 26

3.3.2.2 Estimating Prediction Probability 27

3.3.2.3 Learning Objective 28

3.3.2.4 Noise Contrastive Estimation 29

3.3.2.5 Learning Objective using NCE 30

3.3.2.6 Parameter Estimation 31

3.3.2.7 Hyper-parameters of our model 34

4 Multi-Label Document Categorization **37**

4.1 Logistic Regression for Multi-label Document Categorization 37

4.1.1 Training Data 38

4.1.2 Logistic Regression Model 39

4.1.2.1 Learning Objective 39

4.1.2.2 Parameter Estimation 40

4.2 Similarity to Relational Learning 42

4.3 Advantages of Logistic Regression Learning Algorithm 43

5 Performance Evaluation **45**

5.1 Datasets 45

5.1.1 Reuters-21578 46
5.1.2 Wikipedia Datasets 47
5.2 Experimental Setup 48
5.3 Results 51
5.3.1 Document Categorization 51
5.3.1.1 Reuters - 21578 52
5.3.1.2 Physics - Wikipedia 54
5.3.1.3 Biology - Wikipedia 56
5.3.1.4 Mathematics - Wikipedia 58
5.3.1.5 Sports - Wikipedia 60
5.3.2 Imputing Missing Categories 62
5.3.3 Estimating Similarity between Categories and Words 64

6 Conclusions and Future Work **67**
6.1 Future Work 68
6.1.1 Improving Compositionality of Word Vectors 68
6.1.2 Multi-view Supervised Representation Learning 68

Bibliography **71**

List of Tables

1.1 Example Training Data for Multi-label Document Categorization . . 5

5.1 Statistics of the Reuters-21578 (*ModApte*) Dataset 46

5.2 Statistics of the Wikipedia Datasets 48

5.3 Model performance on Reuters development data for different embedding dimensionality 52

5.4 Model performance on Reuters development data for different number of epochs 52

5.5 Model performance on Reuters development data for different window sizes 53

5.6 Model performance on Reuters development data for different number of negative samples 53

5.7 Precision/Recall/F1 for Document Categorization on Reuters-21578 (*ModApte*) dataset 53

5.8 Model performance on Physics(Wikipedia) development data for different embedding dimensionality 54

5.9 Model performance on Physics(Wikipedia) development data for different number of epochs 55

5.10 Model performance on Physics(Wikipedia) development data for different window sizes 55

5.11 Model performance on Physics(Wikipedia) development data for different number of negative samples 55

5.12 Precision/Recall/F1 for Document Categorization on Physics(Wikipedia) dataset 55

5.13 Model performance on Biology(Wikipedia) development data for different embedding dimensionality 56

5.14 Model performance on Biology(Wikipedia) development data for different number of epochs 57

5.15 Model performance on Biology(Wikipedia) development data for different window sizes 57

5.16 Model performance on Biology(Wikipedia) development data for different number of negative samples 57

5.17 Precision/Recall/F1 for Document Categorization on Biology(Wikipedia) dataset 57

5.18 Model performance on Mathematics(Wikipedia) development data for different embedding dimensionality 58

5.19 Model performance on Mathematics(Wikipedia) development data for different number of epochs 59

5.20 Model performance on Mathematics(Wikipedia) development data for different window sizes 59

5.21 Model performance on Mathematics(Wikipedia) development data for different number of negative samples 59

5.22 Precision/Recall/F1 for Document Categorization on Mathematics(Wikipedia) dataset 59

5.23 Model performance on Sports(Wikipedia) development data for different embedding dimensionality 60

5.24 Model performance on Sports(Wikipedia) development data for different number of epochs 61

5.25 Model performance on Sports(Wikipedia) development data for different window sizes 61

5.26 Model performance on Sports(Wikipedia) development data for different number of negative samples 61
5.27 Precision/Recall/F1 for Document Categorization on Sports(Wikipedia) dataset 61
5.28 Precision/Recall/F1 for Imputing Missing Categories in the Wikipedia Datasets 63
5.29 Estimating Similarity between Categories and Words 65

List of Figures

3.1 Neural Network Architecture for Neural Probabilistic Language Model 20

3.2 Continuous Bag-of-Words Model (CBOW) 21

3.3 Continuous Skip-gram Model 22

3.4 Example for Dependency-based context extraction 23

3.5 Neural network Architecture for our model 27

5.1 Precision/Recall for Document Categorization on Reuters-21578 (*ModApte*) dataset 54

5.2 Precision/Recall for Document Categorization on Physics(Wikipedia) dataset 56

5.3 Precision/Recall for Document Categorization on Biology(Wikipedia) dataset 58

5.4 Precision/Recall for Document Categorization on Mathematics(Wikipedia) dataset 60

5.5 Precision/Recall for Document Categorization on Sports(Wikipedia) dataset 62

List of Algorithms

1 Learning Document and Word Vector Representations 34

2 Learning Category Vector Representations 42

Chapter 1

Introduction

Text documents usually belong to more than one conceptual class. For example, a document on music piracy can be simultaneously classified into *Arts/Music*, *Internet/Security*, *Laws/Cyber*. Multi-Label Document Categorization *(also known as Text Categorization or Classification)* is the task of assigning a text document to one or more predefined categories to describe the semantic content of a document and provide a conceptual view of the document collection. With the growth of online information, Document Categorization has found its use in many important real world applications ranging from document organization to information retrieval. It can be used to organize news stories by categories (topics), classify academic papers by the technical domains and sub-domains they belong to, cluster documents based on their semantic content for easy retrieval and recommendation etc. With the advent of crowd-sourced databases such as Wikipedia[1] which contains over 4 million documents that are manually categorized from a category set containing over 500 thousand categories, automatic document categorization is utmost necessary and useful to assign categories to new articles that are added on a daily basis and also assign missing categories to older documents.

The task of multi-label classification belongs to a general family of supervised learning where the training instances along with the labels they belong to are used to learn a multi-label classifier that assigns appropriate labels to new test instances.

[1] www.wikipedia.org

Another supervised learning task that is very relevant to multi-label classification is that of *ranking*. In the ranking task, the learning algorithm learns a ranking function from the training examples that ranks the set of labels for a new instance such that the more relevant labels are the topmost in the ranked list. To generate the proper output of the multi-label classifier, i.e. the set of relevant labels for the test instance, post-processing of the ranked list of categories is required.

Supervised machine learning techniques that learn classifiers to perform the document categorization task can be broken down into two main components, namely, text representation and learning algorithm. Text representation involves converting the documents, that are usually strings of characters, into numerical vectors that are suitable inputs to the learning algorithm while the learning algorithm uses pairs of labeled input text representations and the categories it belongs to, to learn a model so as to assign relevant categories to new documents.

Over the years, documents have been represented as a *bag-of-words* feature vector, which contains information about the presence and absence of words in the document. Given a corpus of documents, each document d_i in the corpus is represented as a vector $v_{d_i} \in \mathbb{R}^{|V|}$ whose size is equal to the size of the vocabulary. Each element in the vector belongs to $\{0, 1\}$ and denotes whether the particular word is present in the document or not. Though bag-of-words document representation has been widely used for document categorization due to its simplicity, efficiency and ability to capture topical content of the documents necessary for categorization, it suffers from various drawbacks. Bag-of-words representation ignores word ordering and the context in which the words appear in the document, that is vital for encoding the semantic content of text. It also lacks in the ability to encode the semantic similarity between words and documents to estimate distances between them. Disadvantages of such sought have necessitated the need for a more robust and efficient document representation model.

Models to learn fixed-length continuous distributed word vector representations from huge corpora of unlabeled text have shown promising results in tasks of language modeling [2], sentiment analysis [41], machine translation [53] etc. by supplementing the labeled data to overcome the inherent data sparsity and improving generalization accuracies in the high-dimensional domain of Natural Language Processing. Such models learn low-dimensional (generally of the order 100 - 500) vector representations of words that encode the semantic similarity between them [25].

Though these word embeddings try to overcome disadvantages of the bag-of-words model, it is unclear how thy can be composed to represent continuous text, namely documents. In this work we present a model to learn such low-dimensional distributed vector representations for documents to aid in the task of document categorization.

1.1 Motivation

1.1.1 Inability to preserve word ordering

The prime drawback faced by the bag-of-words representation is their inability to preserve word ordering information in the text. Language is a complex phenomenon that often changes meaning when the word ordering in sentences changes, even though they may contain exactly the same words. For example, even though the sentences

Jim can only ride bicycles. and *Only Jim can ride bicycles.*

contain exactly the same words and hence have the same bag-of-words representation, they completely differ in their meaning. While the first sentence points to the fact that *Jim* only rides bicycles among other vehicles, the second sentence suggests that no one apart from *Jim* knows how to ride a bicycle. Similarly, the phrases

"a good book" and *"book a good"*

that have the same bag-of-words representations though they contain different topical content. A document containing the first phrase would like by categorized under *Literature* while containing the second under *Trade*. As shown in the examples above, document representations that preserve word ordering and information about the context the words occur in are more likely to perform better at the task of document categorization.

1.1.2 Lack of similarity measures

Distance or similarity between two documents is commonly computed by taking the dot product of their corresponding representation vectors. In the case of bag-of-words representation, this amounts to counting the number of words co-occurring in the two documents. For example, consider the case when all the words in a document d_1 are replaced by synonyms to form another document d_2, in one case, and replaced by random words to form d_3 in another. The distance between vectors of d_1 and d_2 will be exactly the same as the distance between vectors of d_1 and d_3 even though d_1 and d_2 are much closer to each other than to d_3. Hence, the other major issue faced by the bag-of-words representations is the lack of ability to encode semantic similarity between words and documents. This problem can be partially tackled with the aid of an external Lexical Knowledge Database, such as WordNet for the English Language though an ideal representation should internally encode semantic similarity.

Along with the above stated problems, bag-of-words representations also suffer from high-dimensionality and sparsity issues due to huge vocabulary size in large-scale document corpora that may contain upto million unique words.

1.1.3 Compositionality of distributed word vectors

Though word vectors have shown their efficacy in lot of different NLP tasks, they are limited in their ability to express the meaning of longer phrases and sentences. Document Categorization as a task requires the document representation to encode all the semantic topics present in the document for accurate categorization. There has been progress towards learning distributed representations of documents but it limited to simple weighted average of word vectors. Though it deals with the problem of sparsity and high-dimensionality present in the bag-of-words representation, the problem of preserving word order and contextual information still stands.

In this work, we present an unsupervised model for learning distributed vector representations of documents that along with encoding semantic content of the document also tries to incorporate the contextual information surrounding the words in the document.

1.2 Problem Statement

In the task of multi-label document categorization, we are given a set of documents $D = \{d_1, \ldots, d_{|D|}\}$, set of categories $C = \{c_1, \ldots, c_{|C|}\}$ and a training set $\mathcal{T} = \{l_{d_1}, \ldots, l_{d_n}\}$ which contains category information about n ($n < |D|$) documents. Each label vector l_{d_i} can be thought of as a binary bit-vector of size $|C|$ where the m^{th} element of l_{d_i} being one suggests that the document d_i belongs to category c_m.

Given data about documents, categories and their relationship for a subset of training documents, our task is to learn an appropriate document representation matrix $\mathrm{D} \in \mathbb{R}^{k \times |D|}$, where the i-th column of the matrix is a k-dimensional representation vector for d_i and a multi-label classifier $\mathcal{H}$ such that for a new document d_j we would be able to assign the appropriate categories it belongs to, hence the label vector l_{d_j} using $\mathcal{H}$.

Documents	**Sports**	**Music**	**Arts**	**Technology**	**Literature**	**Politics**
d_1	0	0	1	0	1	0
d_2	0	1	1	0	0	1
d_3	1	0	0	1	0	1
d_4	x	x	x	x	x	x
d_5	x	x	x	x	x	x

Table 1.1: Example Training Data for Multi-label Document Categorization

For example, in Table 1.1 we see that the document set D contains 5 documents and the category set C contains 6 categories. The first 3 documents d_1, d_2 and d_3

contain their label vectors and the rest of the documents d_4 and d_5 are test instances whose category assignments are to be made using the learned classifier $\mathcal{H}$.

1.3 Organization of Thesis

In this chapter we introduced the problem of multi-class document categorization and showed why better document representation models are needed to substitute the bag-of-words representation model. The rest of this thesis is organized as follows. Chapter 2 reviews the different text representation models, dimensionality reduction techniques employed on the bag-of-words representation and various learning algorithms developed for multi-label document categorization. Chapter 3 presents our model for learning distributed document representation vectors along with the details of the model. Chapter 4 presents our approach to multi-label document categorization using a modified logistic regression algorithm. Chapter 5 discusses details about the datasets we use, experimental setup and presents performance evaluation for the document categorization task. The conclusions and future work are a part of Chapter 6.

Chapter 2

Background on Document Categorization

The task of document categorization, i.e. classification of documents into a fixed number of predefined categories has been long studied in-depth for many years now. This multi-class classification problem has further evolved into a multi-label document categorization task where each document can belong to multiple, exactly one or no category at all.

Supervised machine learning techniques that learn classifiers to perform this category assignment task can be broken down into two main components, namely, text representation and learning algorithm. Text representation involves converting the documents, that are usually strings of characters, into numerical vectors that are suitable inputs to the learning algorithm while the learning algorithm uses pairs of labeled input text representations and the categories it is belongs in, to learn a model so as to classify new documents into categories.

2.1 Text Representation

Any text-based classification system requires the documents to be represented in an appropriate manner dictated by the task being performed [20]. Moreover, Quinlan [36] showed that the accuracy of the classification task depends as much on the document representation as on the learning algorithm being employed. Different from the data mining task, which deals with structured documents, text classification deals with unstructured documents that need to be appropriately transformed into numerical vectors, i.e. the need for text representation. In this section we introduce the most effective and widely-used techniques to represent documents as vectors for document categorization.

2.1.1 Bag of Words

It is found in information retrieval research that word stems work well as representations units for documents and that their ordering in a document is of minor importance for many tasks. This is attributed by the fact that the most widely-used used model to represent documents for the classification task is the *Vector Space Model (VSM)* [40].

In the Vector Space Model, a document d is represented as a vector in the term/word space, $d = (w_1, w_2, \ldots, w_{|V|})$ where $|V|$ is the size of the vocabulary. Each of the $w_i \in [0, 1]$, represents the weightage of the term i in the document d. This is called the *bag-of-words* model as it ignores word ordering and each document is reduced to a bag of words that it contains or not.

An important requirement of such a representation is that, the terms that help in defining the semantic content of the document and play an important role in classification be given higher weightage than the others. Over the years, there has been much research in the information retrieval field on term weighting schemes. The most important term-weighting techniques are as follows.

Uniform Weighting : This is the most trivial representation, where each document is represented by a binary vector that is size of the vocabulary. Each element in the vector belongs to $\{0, 1\}$ denoting the absence or presence of a specific term in the document.

Term Frequency : The term frequency representation (*tf*) weighs the terms present in the document relative to their occurrence frequency in the document.

Hence a document d is represented as, $d = (w_1, w_2, \ldots, w_{|V|})$, where, w_k is the number of times the term k appears in the document d.

Inverse Document Frequency : Though using *tf* as a term weighting scheme is a good starting point, it faces a challenge when high frequency terms are not concentrated in a few particular documents but are prevalent in the whole collection. Those terms then stop being characteristic of the semantic content of a few documents and need not be given high weightage. To overcome this problem, Salton and Buckley [39] suggested a new term weighting called the inverse document frequency (*idf*). The *idf* weight of a term varies inversely with the number of documents n it belongs to in a collection of total N documents. A typical *idf* vector can be computed as

$$w_k = \log \frac{N}{n} \tag{2.1}$$

Term Frequency Inverse Document Frequency : Given the above two term weighing schemes, it is clear that an important term in a document should have high *tf* but a low overall collection frequency (*idf*). This suggests that a reasonable measure for term importance may be then obtained by product of *tf* and *idf* (*tf*$\times$*idf*). The bag-of-words representation model using the term frequency - inverse document frequency (*tf-idf*) weighing scheme is the most widely used document representation model. It has shown to give state-of-the-art results with various learning algorithms in the multi-label document categorization task.

2.1.2 Dimensionality Reduction and Feature Selection

The bag-of-words representation scheme has several drawbacks but the most important drawback it suffers from is that document vectors are very sparse and high dimensional. Typical vocabulary sizes of a moderate-sized document collection ranges from tens to hundreds of thousands of terms which is prohibitively high for many learning algorithms. To overcome this issue of high-dimensional bag-of-words document representations, automatic feature selection is performed that removes uninformative terms according to corpus statistics and constructs new orthogonal features by combining several lower level features (terms/words). Several techniques used in practice are discussed below.

Information Gain : Information Gain is widely used as a term-goodness criterion in the field of machine learning, mainly in decision trees [37] and also in text classification [21], [32]. It is a feature space pruning technique that measures the number of bits of information obtained(entropy) for category prediction by knowing the presence or absence of a term in a document. For terms where the information gain was below some predefined threshold are not considered in the document vector representation. The information gain of a term t is defined as

$$G(t) = -\sum_{i=1}^{|C|} P(c_i) \log P(c_i) + P(t) \sum_{i=1}^{|C|} P(c_i|t) \log P(c_i|t) + P(\sim t) \sum_{i=1}^{|C|} P(c_i|\, t) \log P(c_i|\, t) \tag{2.2}$$

Mutual Information : Similar to the Information Gain scheme, Mutual Information estimates the information shared between a term and a category and prunes terms that are below a specific threshold. The mutual information between a term t and a category c is estimated in the following fashion,

$$I(t, c) = \log \frac{P(t \wedge c)}{P(t) \times P(c)} \tag{2.3}$$

To measure the goodness of a term in global feature selection, the category specific scores of a term are combined using,

$$I_{avg}(t) = \sum_{i=1}^{|C|} P(c_i) I(t, c_i) \tag{2.4}$$

χ^2 **Statistic** : The χ^2 statistic measures the lack of independence a term t and a category c and can be compared to the χ^2 distribution with one degree of freedom. The term-goodness factor is calculated for each term-category pair and is averaged as in Eq. 2.4. The major difference between Mutual Information and χ^2 statistic is that the later is a normalized value and the goodness factors across terms are comparable for the same category.

Latent Semantic Indexing (LSI) : LSI first introduced by Deerwester et al. [9], is a popular linear algebraic dimensionality reduction technique that uses the term co-occurrence statistics to capture the latent semantic structure of the documents and represent them using low-dimensional vectors. It is an efficient technique to deal with synonymy and polysemy. LSI aims to find the best subspace approximation to the original document bag-of-word vector space using Singular Value Decomposition. Given a term-document matrix $X = \left[x_1, x_2, \ldots, x_{|D|}\right] \in \mathbb{R}^{|V| \times |D|}$, its k-rank approximation as found using SVD, can be expressed as,

$$X = TSD^T \tag{2.5}$$

where, $T \in \mathbb{R}^{|V| \times k}$ and $D \in \mathbb{R}^{|D| \times k}$ are orthonormal matrices called the left and right singular vectors respectively. The matrix $S \in \mathbb{R}^{k \times k}$ is a diagonal matrix of singular values arranged in descending order. The k-dimensional rows of the matrix D contain the dimensionality reduced representations of the $|D|$ documents in the collection. The representations obtained using LSI alleviate the issue of data sparsity and high-dimensionality in bag-of-words representations and also helps unfold the latent semantic structure of the documents.

2.2 Learning Algorithms

Multi-label document categorization has seen growing number of statistical learning methods being applied to it. Over the years, various learning algorithms like, Regression models [6, 11], Conditional Random Field [12], Nearest Neighbor techniques [47, 51, 52], Bayesian classifier and topic modeling [21, 24, 35, 38, 34, 44], SVM [17, 10], Neural Networks [46, 33], Decision Trees [42], Online learning algorithms [22, 8], Non-negative Matrix Factorization [23] etc. have been used and developed for Multi-label document categorization.

Earlier learning algorithms reduced the problem of multi-label classification into multiple binary classification problems and independently learned binary classifiers for each category. While these algorithms performed well, their drawback of considering correlation among categories led to the development of algorithms that learn a single classifier and jointly classify each document.

The other supervised learning task that has the most similarity with multi-label classification is *ranking*. While the former assigns each test instance a $|L|$-sized binary label vector indicating the presence and absence of labels, the ranking algorithm outputs the list of labels arranged in the increasing order of a ranking score which can then be thresholded at an optimum to consider the top labels as appropriate label assignments for test instances.

Below we describe some of the widely learning algorithms for multi-label document categorization.

2.2.1 Document Categorization using Binary Classifiers

The most common approach of multi-label document categorization treats each label independently and learns multiple binary classifiers, one for each category and then assigns to a test document all the categories for which the corresponding classifier says *'yes'*. Algorithms that learn multiple independent binary classifiers for multi-label classification are explained below in the context of document categorization.

Logistic Regression : Introduced by Hosmer and Lemeshow [16], Logistic Regression (LR) is a probabilistic binary classification regression model, that, for binary text classification learns a category weight vector and estimates the probability of a document belonging to the category using dot-product and the logistic link function. LR can be extended for multi-label document categorization by learning multiple category vectors, specifically, one for each category. At test time, one would need to query all category vectors for each document to make the category assignments. In our work, we use logistic regression for multi-label document categorization, the details for which are given in Sec 4.1.

Support Vector Machines : Support Vector Machines (SVM) ([7], [45]) based on the *Structural Risk Minimization* principle, are universal learners. In their basic form, SVMs learn linear threshold functions to find linear hyperplanes in the input data space to separate data of the two different classes. In the case, where data is not linearly separable, SVMs can be plugged-in with appropriate kernel functions to learn polynomial classifiers, radial basic functions etc. For multi-label document categorization, training data is treated separately for each category and maximum margin separating hyperplanes are found for each category independently [17].

Elisseeff and Weston [10] study a ranking based variant of SVM, where the positive/negative distance from the separating hyperplane of a specific category is the score assigned to the particular instance for that category. Their formulation then aims to maximize the margin between the score of a category that belongs to the document and a category that does not belong to do the document. This is also called the Rank-SVM.

Neural Networks : Classification-based, Neural Network approaches to multi-label document categorization were mainly studied by Wiener et al. [46], developed at Xerox PARC and called NNet.PARC and Ng et al. [33], called CLASSI. Both neural networks are examples of multiple-classifier based approaches where a separate neural network was trained for each category to make binary classifications. While CLASSI used a linear perceptron approach to classify text into categories, NNet.PARC built a three-layered nonlinear neural network that extends logistic regression by modeling higher order term interactions and hence finding non-linear decision boundaries.

Naive Bayes : Naive-Bayes (NB) as studied in Lewis [19] and Lewis and Ringuette [21], is one of the most effective and simple statistical model for text classification. For multi-label classification, classifiers are learned so as to estimate $P(C_j = 1|D)$, i.e., the probability that the document, D belongs to the category C_j, for each category. This probability is estimated by estimating the probability $P(W_i = 1|C_j = 1)$, i.e. probability that a particular word appears in the document when it belongs to a particular category. Though this approach makes the assumption of word independence, experiments show that this fast-learning algorithm can yield excellent results.

Although, approaches to multi-label classification discussed above give competitive accuracies in the task, they suffer from inefficiencies due to the following reasons.

Such algorithms make assumptions of category independence and learn 1-vs-All binary classifiers. It is realized that such assumption would not hold true in most real-life situations. Fine-grained categorization of texts usually involve strongly correlated category classes and information about the presence of one gives information about the presence/absence of many others. For eg.

> *Chicago Board of trade grain traders and analysts voiced a lot of interest in how farmers planned to handle their upcoming spring plantings prompting sales of new crop months of corn and oats and purchases in new crop soybeans in the futures markets*

In the sentence above, information from words about the presence of categories like *oats*, *corn* etc. can also aid the prediction of the *agriculture* category which can be boosted using joint classification.

In the case of millions of labels in the dataset, learning millions of high-dimensional classifiers is computationally expensive. Also, the cost of prediction for each test instance would be high as all the classifiers need to be evaluated to make a prediction for a single data-point (document).

2.2.2 Document Categorization with Single Joint Classifier

To overcome the drawbacks in learning multiple binary classifiers, learning algorithms that jointly assign all the categories each document belongs to, have been developed. Outputs of such algorithms are $|L|$-dimensional binary label vectors $\boldsymbol{y} \in \{0,1\}^L$, with $\boldsymbol{y}_l = 1$ if label l is relevant for the particular document. Below we describe algorithms of such kind.

k-Nearest Neighbor : k-nearest neighbor (kNN) classification is one of the most effective lazy learning approaches to classification. Given an arbitrary text document as input, the algorithm first ranks the nearest neighbors among the training documents using some similarity measure. It then uses the category information of the top-k ranked nearest neighbors to predict the categories of the input test document. One simple approach is to take a weighted average of the label vector of the k-nearest neighbors, weights being the similarity score computed while estimating document distances. This yields a category ranking for the test input which can be thresholded to yield binary classifications.

Other approach as devised by Zhang and Zhou [52] is based on the k-NN and the maximum a posteriori(MAP) principle. Given a test instance, their model first identifies its k-nearest neighbors and then based on the statistical information gained from the label sets of the neighboring instances, uses the MAP principle to determine the label set of the given input. The prior probability of label occurrences and the posterior probability, $P(C_l = n|l = 1)$ i.e. given a document belongs to label l, exactly n of its k neighbors also belong to the label l is determined from the training instances to utilize the MAP principle.

Linear Least Squares Fit : Linear Least Squares Fit (LLSF) [48] learns a multivariate regression model automatically from a training set of documents and their categories. Documents are input as vectors in the desired representation and the corresponding output is a $|L|$-dimensional binary label vector. By solving a linear least squares fit on the training pairs of vectors a matrix of word-category regression coefficients is learned, which defines the mapping from an arbitrary document to a weighted category label vector. This weighted vector can be sorted to yield a ranked list of categories for the input document.

Probabilistic Models : Generative probabilistic models described in McCallum [24], Nigam et al. [34], Ueda and Saito [44] etc. argue that the words in a document that belongs to multiple categories can be regarded as a mixture of characteristic words related to each of the categories. Therefore, they represent the multi-label nature of the document by specifying each document with a set of mixture weights, one for each class and also indicate that each document is generated by a mixture of word distributions, one distribution for each label. Once the word distributions are learned using the training data, classification is performed using the Bayes Rule which selects the labels that are most likely to generate the given test document. Hence, along with giving the information on the labels responsible for generating the document, such models also fill the missing information of which labels were responsible for generating each word.

McCallum [24] and Ueda and Saito [44] define a multinomial distribution $\boldsymbol{\theta}_l = \{\theta_{l1}, \theta_{l2}, \ldots, \theta_{l|V|}\}$ over the vocabulary for each label l. The word distribution for a document for a given label vector $\boldsymbol{y}$, is computed by taking a weighted average of the word distributions of the labels that are present in the document. Therefore, if $\boldsymbol{\phi}(\boldsymbol{y}) = \{\phi_1(\boldsymbol{y}), \phi_2(\boldsymbol{y}), \ldots, \phi_{|V|}(\boldsymbol{y})\}$ is the required word distribution, it can be represented by,

$$\boldsymbol{\phi}(\boldsymbol{y}) = \sum_{l=1}^{|L|} h_l(\boldsymbol{y})\boldsymbol{\theta}_l \tag{2.6}$$

where $h_l(\boldsymbol{y})$'s are the mixing proportion that add to 1. The word distributions for each label are found by maximizing the posterior [44] and employing the Expectation-Maximization algorithm [24].

Chapter 3

Distributed Document Representations

In this chapter we describe the concept of distributed word and document representations and why distributed representations of words and documents are better than one-hot or bag-of-words representations as described in 2.1. We then give a background on different models for learning distributed word representations in a fully unsupervised manner. We then finally describe in detail our proposed model for learning distributed document representations that can be used for multi-label document categorization.

3.1 Need for Distributed Word Representations

Words are regarded as atomic symbols in most rule-based and statistical natural language processing (NLP) tasks and hence need appropriate representation to solve the NLP tasks with greater ease and accuracy. Words are traditionally expressed as one-hot vectors, i.e. as vectors of the size of the vocabulary where exactly one element is 1 and the rest all are zero. Though these representations have been widely used, one-hot representations have a plethora of drawbacks that pose problems and limit the ability of systems to perform better.

1. **Curse of Dimensionality** : One-hot representations lead word vectors to be

the size of the vocabulary which often consists of tens to hundreds of thousands of words. Due to this curse of dimensionality, language modeling becomes almost impossible where the number of parameters would grow exponentially with the size of the vocabulary if the words are represented as one-hot vectors.

2. **No Word Similarity** : As words are represented by sparse orthogonal vectors, there is no notion of word similarity that can be introduced. In one-hot representation, the word "symphony" is equally close to the words "bark" and "guitar". We would want word representations such that they capture the semantic or topical similarity between words.

Due to the problems discussed above there is a need for continuous, low-dimensional, non-sparse vector representations for words that capture their semantic content and can be used to model language with continuous distributions and can be used as inputs for various other NLP tasks.

3.2 Background on Word Embeddings

Distributed word representations are dense fixed-sized feature vectors learned for words in an unsupervised manner from large text corpus that capture the semantic content of words. Each word w_i in the corpus is represented by a vector, $v_{w_i} \in \mathbb{R}^m$, where m usually ranges from $50 - 300$ dimensions. These dense representations help deal with sparsity and high-dimensionality issues in one-hot representations and also provide provision for estimating similarities between words; which is as simple as taking the dot-product or calculating the cosine-distance between the vectors.

All of the word vector learning models make use of neural networks [2, 29, 26, 5, 4, 43, 18] but differ in their learning objectives.

Below we describe few models to show how different learning objectives and architecture can lead to learning high-quality word vectors with different properties.

3.2.1 Neural Probabilistic Language Model

Neural Probabilistic Language Model (NPLM), introduced by Bengio et al. [2], aims to learn distributed word vectors and a probability function that uses these vectors to learn a statistical model of language. In their model, the probability of a word sequence is expressed as the product of conditional probabilities of the next word given the previous ones.

$$P(w_1^T) = \prod_{t=1}^{T} P(w_t | w_1^{t-1}) \tag{3.1}$$

And making the n-gram assumption,

$$P(w_t | w_1^{t-1}) \approx P(w_t | w_{t-n+1}^{t-1}) \tag{3.2}$$

i.e. the probability of the next word in the sequence is mostly affected by the local context, in this the previous n-words and not the whole past sequence.

Their model maps each word to a m-dimensional vector in a matrix $C \in \mathbb{R}^{|V| \times m}$ and estimates the probability $P(w_t = i | w_{t-n+1}^{t-1})$ i.e. the probability that the t^{th} word in the sequence is w_i. The neural network that is used to estimate this probability using the word vectors is shown in Figure 3.1. For each input sequence, the neural network outputs a vector $y \in \mathbb{R}^{|V|}$, where y_i is the unnormalized log-probability that the t^{th} word in the sequence is w_i. The output y is computed as,

$$y = b + Wx + Utanh(d + Hx) \tag{3.3}$$

where $tanh$ is the hyperbolic tangent applied to introduce non-linearity and x is the word feature layer activation vector constructed by the concatenation of the context word vectors,

$$x = (C(w_{t-1}), C(w_{t-2}), \ldots, C(w_{t-n+1})) \tag{3.4}$$

The unnormalized log probabilities in y are converted to positive probabilities sum-

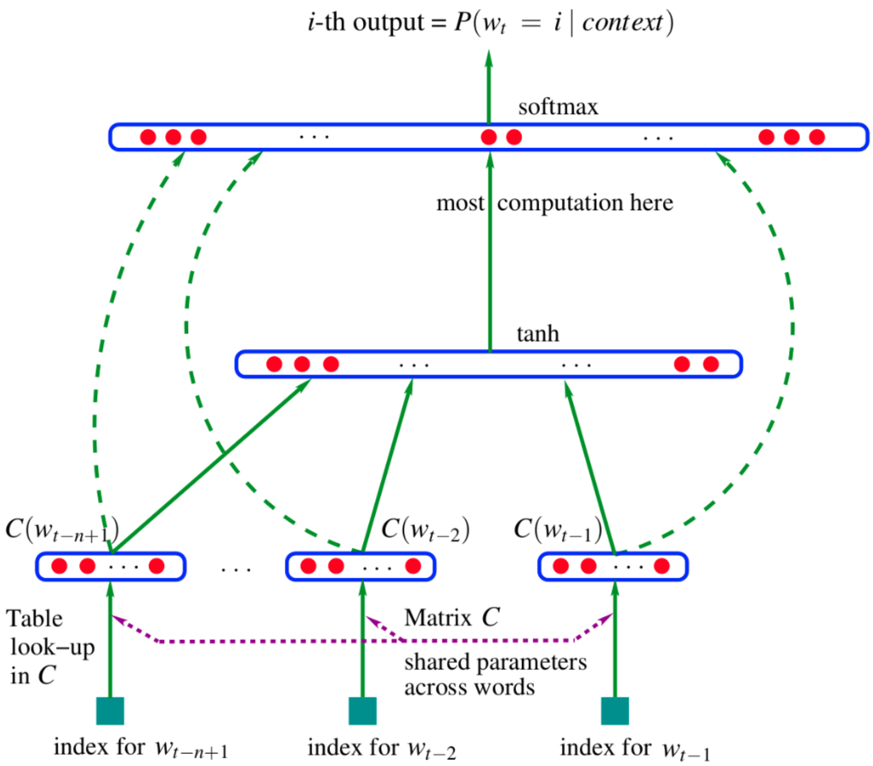


Figure 3.1: Neural Network Architecture for Neural Probabilistic Language Model

ming to 1 by using a *softmax* output layer that computes,

$$P(w_t = i|w_{t-1}, \ldots, w_{t-n+1}) = \frac{e^{y_{w_t}}}{\sum_i e^{y_i}} \tag{3.5}$$

The parameters of the model (b, d, W, U, H) and the word vectors C are estimated by maximizing the log-likelihood of the training corpus.

3.2.2 Log-Linear Models

Simple log-linear models are proposed in Mikolov et al. [25] as opposed to the non-linear NPLM model to bring down the training time complexity without sacrificing with the quality of the word vectors. It achieves so by not having a non-linear layer and matrix weighting of the input vectors, that are the costliest operations in NPLM. The two models proposed in Mikolov et al. [25] (word2vec) are Continuous Bag-of-Words model and Continuous Skip-Gram model, described below.

3.2.2.1 Continuous Bag-of-Words Model

The Continuous Bag-of-Words (CBOW) model is different from the NPLM in that the projection layer is shared for all words; i.e. all words get projected into the same hidden layer vector (their vectors are averaged). This architecture hence neglects the ordering of the words as opposed to NPLM that uses the concatenation of input vectors for the projection layer. The training criteria in this model is to predict the current (middle) word given its context. It also uses word sequence from the future to aid this task with the relaxation that the aim is not to learn a language model. The model architecture is given in Figure 3.2. The model first computes the hidden

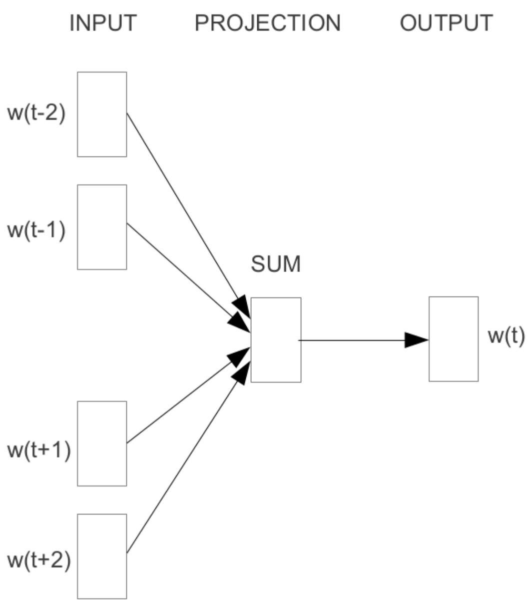


Figure 3.2: Continuous Bag-of-Words Model (CBOW)

layer vector h,

$$h(w_{t-k}, \ldots, w_{t+k}) = \frac{w_{t-k} + \ldots + w_{t-1} + w_{t+1} + \cdots + w_{t+k}}{2k} \tag{3.6}$$

where, w_{t-i} is the i-th previous word in the context of the middle word w_t and k is the window length. The neural network then computes a unnormalized log-probability vector y similar to Sec.3.2.1, and uses the *softmax*-classifier to estimate $P(w_t|w_{t-k}, \ldots, w_{t+k})$,

$$y = b + Uh(w_{t-k}, \ldots, w_{t+k}) \tag{3.7}$$

$$P(w_t|w_{t-k},\ldots,w_{t+k}) = \frac{e^{y_{w_t}}}{\sum_i e^{y_i}} \tag{3.8}$$

The parameters of the CBOW model, (b, U) and the word vectors (w_i) are learned by maximizing the average log probability (Eq. 3.8) of the training corpus.

3.2.2.2 Continuous Skip-gram Model

This model is similar to the CBOW model, but instead of predicting the middle word based on the context, it tries to maximize the classification of a word based on another word in the context. More precisely, given each word, the skip-gram model tries to predict words within a certain range before and after the current word. The model architecture is given in Figure 3.3. Formally, given a sequence

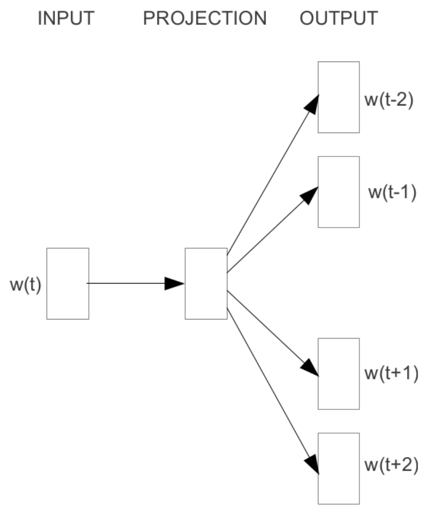


Figure 3.3: Continuous Skip-gram Model

of words in a context $w_{t-k},\ldots,w_{t+k}$, the skip-gram model defines $P(w_{t+j}|w_t)$ using the *softmax*-classifier in the following manner,

$$P(w_{t+j}|w_t) = \frac{e^{(v_{w_t}\cdot v_{w_{t+j}})}}{\sum_i e^{(v_{w_t}\cdot v_{w_i})}} \tag{3.9}$$

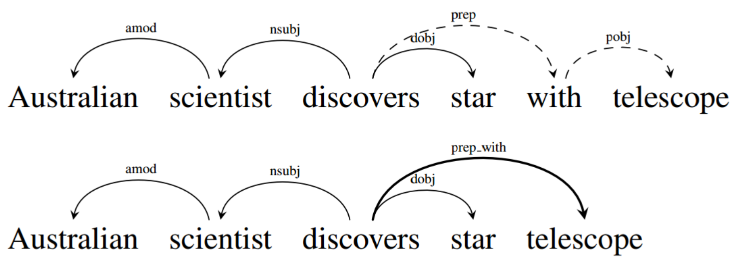
The only parameters of the Skip-gram model are the word vectors (v_{w_i}) that are learned by maximizing the average log probability (Eq. 3.9) of predicting all the context words for all the words in the training corpus.

The CBOW and the Skip-gram models use the *hierarchical softmax* [31] instead of the full softmax to speed-up the learning process.

The quality of the word vectors is tested using the *Semantic-Syntactic Word Relationship test* that evaluates the model performance on retrieving semantically and syntactically similar words to the given test words. The word vectors learned using the skip-gram model are also shown to encode many linear linguistic regularities and pattern [27] and show additive compositionality using simple vector arithmetics. For example, the result of the vector calculation $vec(Madrid) - vec(Spain) + vec(France)$ is closest to $vec(Paris)$ than any other word vectors.

3.2.2.3 Dependency-based Word Embeddings

Instead of using bag-of-words based context as used in *NPLM* and *word2vec*, Levy and Goldberg [18] use arbitrary contexts to investigate its effects on the word vectors and the properties they encode. The most important of their techniques is to derive the contexts based on the syntactic relations that the word participates in. For each word w and its modifiers $m_1, \ldots, m_k$ found using the parse tree of the sentence, contexts $(m_1, lbl_1, \ldots, m_k, lbl_k)$ are extracted, where *lbl* is the type of the dependency relation between word and the modifier and lbl^{-1} is used to mark the inverse-relation. An example of the contexts extracted for a sentence is given in Figure 3.4. After



WORD	CONTEXTS
australian	scientist/amod^{-1}
scientist	australian/amod, discovers/nsubj^{-1}
discovers	scientist/nsubj, star/dobj, telescope/prep_with
star	discovers/dobj^{-1}
telescope	discovers/prep_with^{-1}

Figure 3.4: Example for Dependency-based context extraction

context extraction, their model uses the neural network architecture and the learning objective of the skip-gram model to learn word vectors. On comparison to the vectors learned from the skip-gram model on the tasks of *topical similarity* and *functional similarity* estimation, it is found that the vectors learned from this model perform better on the *functional similarity* task that expects word vectors to encode syntactic relationships better. The task of *topical similarity* estimation showed that vectors from the skip-gram model encode semantic content better because of the bag-of-words context used during training.

3.3 Document Representation

Though the semantic word spaces as described above are very useful for a lot of NLP tasks, their ability to capture the complexity and compositionality of human language is limited. Word embeddings cannot be directly used to represent longer phrases, sentences and documents to express their meaning. Tasks such as word sense disambiguation, sentiment analysis, text categorization etc. all require the text representation to capture the semantic content of the text for better inputs to learning algorithms as compared to a simple bag-of-words model.

Progress towards learning distributed representations for longer pieces of text, such as phrase-level or sentence-level representations [28, 50, 49, 13, 26] that capture semantic compositionality has been promising, but most models do not go beyond simple weighted average of word vectors to represent longer texts. Socher et al. [41] proposes a more sophisticated approach using recursive tensor neural network where the dependency parse-tree of the sentence is used to compose word vectors in a bottom-up approach to represent sentences for sentiment classification of phrases and sentences. Both the techniques have weaknesses for learning document representations. The first approach is analogous to a bag-of-words approach and neglects word order while representing documents whereas the second approach considers syntactic dependencies but cannot go beyond sentences as it relies on parsing.

Below we present our model on learning universal distributed representations for documents and words in the corpus such that,

1. The learned document representations encode different semantic and topical content of the documents.

2. Documents and words are embedded in the same k-dimensional space such that semantically similar entities have similar vector representations.

To learn vectors that satisfy 1. and 2., we hypothesize that document representations should be learned such that they can aid in the prediction of words in a given word sequence from the document. In the sections below we formally introduce the problem and present our model to learn document and word vector representations.

3.3.1 Problem Setup

Given a set of documents, $\mathbf{D} = \{d_1, \ldots, d_{|\mathbf{D}|}\}$ and a vocabulary of words, $\mathbf{V}$ constructed using the given documents, we wish to embed each document $d_i \in \mathbf{D}$ and each word in the vocabulary onto the same k-dimensional space such that the learned vectors encode semantic content of the entities.

For every sequence of words $w_{t-c}, \ldots, w_{t+c}$ in, say document d_i, we wish to estimate the probability $p(w_t | d_i, w_{t-c}, \ldots, w_{t-1}, w_{t+1}, \ldots, w_{t+c})$ of predicting the middle word in the sequence using the information about the document and the words in the context. We will estimate this probability using the vector representations for documents and words and learn vectors such that the probability of predicting the middle word correctly in the context is maximized.

3.3.2 Our Model

A document $d_i \in \mathbf{D}$, indexed by 'i', in our model is represented by a vector $\mathrm{v}_i^D \in \mathbb{R}^k$, which is also the i-th column of the matrix $\mathrm{D} = \left[\mathrm{v}_1^D, \ldots, \mathrm{v}_{|\mathbf{D}|}^D\right] \in \mathbb{R}^{k \times |\mathbf{D}|}$. Similarly, a word indexed by 'i' in the vocabulary $\mathbf{V}$ is represented by the vector $\mathrm{v}_i^W \in \mathbb{R}^k$, which is also the i-th column of the matrix $\mathrm{W} = \left[\mathrm{v}_1^W, \ldots, \mathrm{v}_{|\mathbf{V}|}^W\right] \in \mathbb{R}^{k \times |\mathbf{V}|}$.

Given a sequence $(w_{t-c}, \ldots, w_{t+c})$ of $2c + 1$ words and the document it occurs in, our training objective is to maximize the probability of correctly predicting the middle word, $p(w_t|d_i, w_{t-c}, \ldots, w_{t-1}, w_{t+1}, \ldots, w_{t+c})$, using the surrounding context words and the information about the document in terms of their distributed vector representations.

To learn the distributed document and word representations, we present a neural network model using which we,

1. Represent each document and word in the corpus by a k-dimensional vector stored in the matrices D and W, respectively.

2. Estimate the probability of predicting the middle word in a given word sequence using the vector representation of the the words in the context and the document it occurs in.

3. Learn the word and document vectors along with the parameters of the model simultaneously to estimate the probability.

The architecture for the proposed neural network is given in Fig. 3.5. Also note that the word vector representations learned and stored in the matrix W are universal representations and shared across all documents and contexts.

3.3.2.1 Context Representation

We represent the context(words surrounding the middle word to be predicted) and the document together in the same projection layer, denoted by $h_c \in \mathbb{R}^k$, by taking a weighted sum of the corresponding vector representations. The weights for the context words $\Lambda = \{\lambda_i | i = \{t-c, \ldots, t-1, t+1, \ldots, t+c\}\}$ are kept universal for different sequences across the corpus as we expect the weights to learn some kind of syntactic quality of the language to better represent the context. Also the weight corresponding to the document vector is kept constant at 1. We also (unsuccessfully) experimented by taking matrix weights instead of scalar weights(λ_i) to learn better syntactic qualities of the language. The projection layer representation is computed

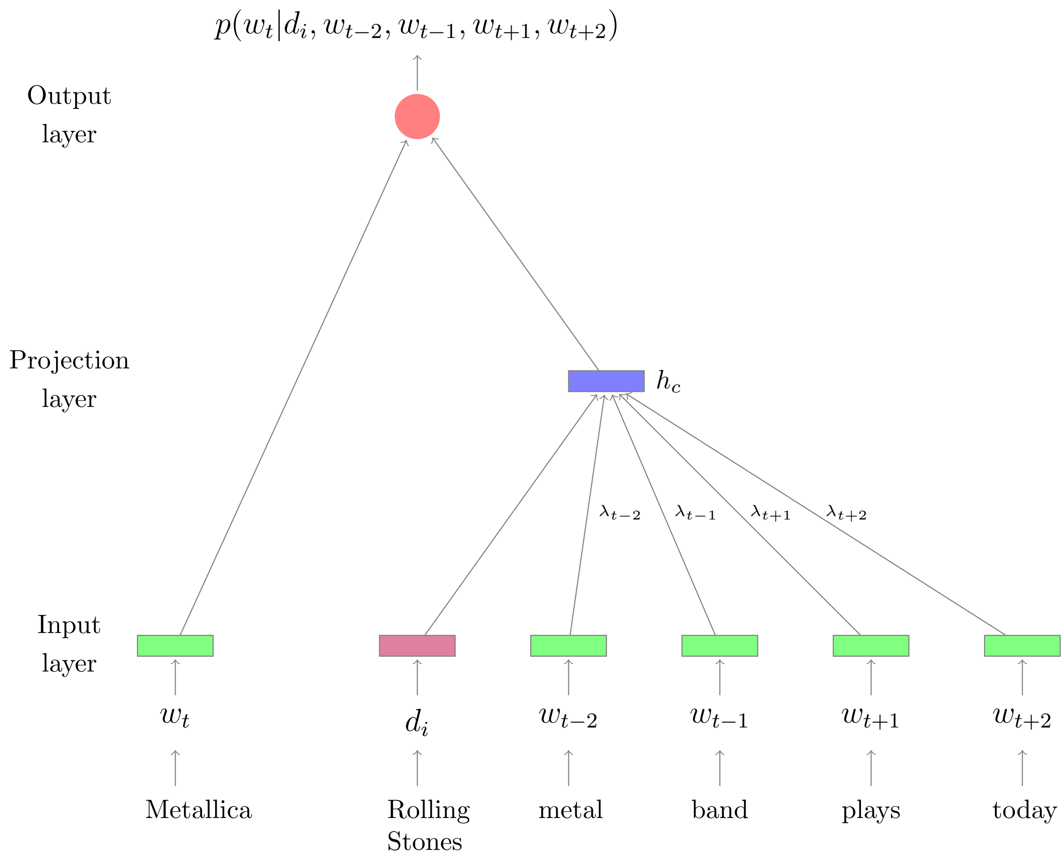


Figure 3.5: Neural network Architecture for our model

using,

$$h_c = \mathrm{v}_i^D + \lambda_{t-c}\mathrm{v}_{t-c}^W + \ldots + \lambda_{t-1}\mathrm{v}_{t-1}^W + \lambda_{t+1}\mathrm{v}_{t+1}^W + \ldots + \lambda_{t+c}\mathrm{v}_{t+c}^W \tag{3.10}$$

3.3.2.2 Estimating Prediction Probability

We expect in absence of any non-linearity that the projection layer vector should be aligned to the correct middle word of the sequence. Hence we estimate the probability of predicting the word w_t as the middle word in the following manner.

1. An output score $s_{w_i} \in \mathbb{R}$ for every w_i in the vocabulary is estimated by,

$$s_{w_i} = \sigma(\mathrm{v}_{w_i}^W \cdot h_c) \tag{3.11}$$

   where $\sigma(z) = \frac{1}{1+e^{-z}}$ is the standard sigmoid function.

2. After calculating the score for each of the word in the vocabulary, we use the

softmax classifier to estimate the probability of predicting the actual correct word w_t as the middle word in the sequence,

$$p(w_t|d_i, w_{t-c}, \dots, w_{t-1}, w_{t+1}, \dots, w_{t+c}) = \frac{e^{s_{w_t}}}{\sum_{i \in \mathbf{V}} e^{s_i}} \tag{3.12}$$

3.3.2.3 Learning Objective

The training data $\mathcal{T}$, is composed of M training sequences each of which consists a $2c+1$ length sequence of words and the document index it belongs to. For example, $t = \{d_i^{(m)}, w_{t-c}^{(m)}, \dots, w_{t+c}^{(m)}\}$ represents the m^{th} training sequence in $\mathcal{T}$.

Given the training data $\mathcal{T}$, our objective is to learn an optimum parameter set $\Theta = (\mathrm{D}, \mathrm{W}, \Lambda)$ consisting of the document and word vector matrices and the projection layer weights for the context words, by maximizing the average log probability of estimating the middle word correctly in all the training word sequences where the probability of estimating the middle word as w_i is given by Eq. 3.12. Therefore,

$$\hat{\Theta} = \arg\max_{\Theta} \; l(\mathcal{T}, \Theta) \tag{3.13}$$

$$l(\mathcal{T}, \Theta) = \frac{1}{M} \sum_{m=1}^{M} \log \left[p(w_t^{(m)}|d_i^{(m)}, w_{t-c}^{(m)}, \dots, w_{t-1}^{(m)}, w_{t+1}^{(m)}, \dots, w_{t+c}^{(m)}) \right] \tag{3.14}$$

To optimize the training objective in Eq. 3.14, we can use the Stochastic Gradient Descent(SGD) algorithm to find gradient of the objective function (Eq. 3.14) w.r.t. to the individual parameters θ_i and apply the update rule as follows,

$$\theta_i^{(x)} = \theta_i^{(x-1)} + \gamma \frac{\partial l(\mathcal{T}, \Theta)}{\partial \theta_i} \tag{3.15}$$

where x is the current iteration number and γ is the learning rate. Note that we add the gradient to $\theta_i^{(x-1)}$ because we wish to maximize the training objective. Updating the parameters for sufficient number of iterations would yield the optimum document and word vectors along with the weights for the neural network.

3.3.2.4 Noise Contrastive Estimation

As we see in Eq 3.12, estimating the probability for each training word sequence requires a sweep through the whole vocabulary of size $|\mathbf{V}|$ which can be a very expensive computation given that typical vocabulary sizes range from a few tens to a few hundreds of thousand words for large datasets. Approaches to reduce this training time, such as, use of hierarchical soft-max [31] and use of importance sampling to approximate the likelihood gradient [3, 1] have been proposed. Hierarchical softmax reduces the training time from linear to logarithmic in vocabulary size but is considerably more involved and finding well-performing trees is not trivial. Also, though importance sampling provides substantial speedups, it suffers from stability problems.

Noise Contrastive Estimation (NCE) [15] is method for fitting unnormalized probabilities by reducing the problem of *probability density estimation* to *probabilistic binary classification*. It has also been adapted to NPLM (Sec. 3.2.1) [30] and learning word embeddings. Mnih and Kavukcuoglu [29] and shows significant improvements in training time with no considerable degradation in the quality of word vectors learned.

The basic idea of NCE as incorporated to our model is to learn a logistic classifier to distinguish between the correct middle word in the given word sequence and corrupt samples from some "noise" distribution. Therefore, given a training sequence of the form $t = \{d_i^{(m)}, w_{t-c}^{(m)}, \ldots, w_{t+c}^{(m)}\}$, our training objective now is to train a classifier such that it can distinguish between positive training sample $w_t^{(m)}$ as positive example and negative training samples w_x drawn from a noise distribution $P_n(w)$ as negative examples for the middle word given the surrounding words (context) and the document the sequence belongs to.

Our training data $\mathcal{T}$ is now converted to a set of labeled sequences of the form $\{d_i^{(m)}, w_{t-c}^{(m)}, \ldots, w_{t+c}^{(m)}, Y^{(m)} = 1\}_{m=1}^{m=M}$ where $Y = 1$ denotes that the sequence is a positive sample occurring in the corpus. For every positive training sequence t we also have n corrupt training sequences where, in each of them only the middle word

w_t has been replaced by a corrupt word sampled from the noise distribution $P_n(w)$ and the value of the label $Y = 0$. Therefore, for every positive training example there exists n negative training examples and the total number of training samples in $\mathcal{T}$ is $M + nM$ now. We now need to train a binary classifier such that, given a sequence of words and the document it belongs to, it can predict correctly whether the sequence is legitimate (correct value of the label indicator Y).

Given a training sequence we estimate the probability that the given sequence is positive using,

$$P(Y = 1|d_i, w_{t-c}, \ldots, w_{t+c}, \Theta) = \sigma(\mathrm{v}^W_{w_t} \cdot h_c) \tag{3.16}$$

where h_c is the projection layer vector calculated using Eq. 3.10. Similarly, the probability of estimating that a given sequence is corrupt is given by,

$$P(Y = 0|d_i, w_{t-c}, \ldots, w_{t+c}, \Theta) = 1 - \sigma(\mathrm{v}^W_{w_t} \cdot h_c) \tag{3.17}$$

From Eq. 3.16 and Eq. 3.17 we get,

$$P(Y|d_i, w_{t-c}, \ldots, w_{t+c}, \Theta) = [\sigma(\mathrm{v}^W_{w_t} \cdot h_c)]^Y [1 - \sigma(\mathrm{v}^W_{w_t} \cdot h_c)]^{1-Y} \tag{3.18}$$

As a shorthand notation, we would express the probability estimation in Eq. 3.18 as $P_\Theta(Y)$.

3.3.2.5 Learning Objective using NCE

Our new training objective involves maximizing the log-likelihood of observing the modified training data $\mathcal{T}$ that includes the negative examples sampled from the noise distribution $P_n(w)$ along with the original positive training sequences.

$$\hat{\Theta} = \arg\max_{\Theta} \; l(\mathcal{T}, \Theta) \tag{3.19}$$

$$l(\mathcal{T},\Theta) = \sum_{m=1}^{M+nM} \log P_\Theta(Y_m = Y^{(m)}) \tag{3.20}$$

where Y_m is the predicted label for the m-th training sequence and,

$$\log P_\Theta(Y_m = Y^{(m)}) = Y^{(m)} \log \sigma(\mathrm{v}^W_{w_t^{(m)}} \cdot h_c^{(m)}) + (1 - Y^{(m)}) \log(1 - \sigma(\mathrm{v}^W_{w_t^{(m)}} \cdot h_c^{(m)})) \tag{3.21}$$

3.3.2.6 Parameter Estimation

To learn the optimum parameters $\Theta = (\mathrm{D}, \mathrm{W}, \Lambda)$, we maximize the log-likelihood of observing the training data given in Eq 3.19 using the Stochastic Gradient Descent(SGD) algorithm described below.

Firstly, for the SGD algorithm we need to calculate the gradient of the log probability estimate (Eq 3.21) with respect to individual parameters $\theta \in \Theta$. The derivative of the log probability estimate w.r.t. to a parameter $\theta \in \Theta$ is given by,

$$\frac{\partial \log P_\Theta(Y_m = Y^{(m)})}{\partial \theta} = \left[Y^{(m)} \frac{1}{\sigma(d^{(m)})} - (1 - Y^{(m)}) \frac{1}{(1 - \sigma(d^{(m)}))} \right] \frac{\partial \sigma(d^{(m)})}{\partial \theta} \tag{3.22}$$

$$= \left[Y^{(m)} \frac{1}{\sigma(d^{(m)})} - (1 - Y^{(m)}) \frac{1}{(1 - \sigma(d^{(m)}))} \right] \left[\sigma(d^{(m)})(1 - \sigma(d^{(m)})) \right] \frac{\partial d^{(m)}}{\partial \theta} \tag{3.23}$$

$$\frac{\partial \log P_\Theta(Y_m = Y^{(m)})}{\partial \theta} = \left[Y^{(m)} - \sigma(d^{(m)}) \right] \frac{\partial d^{(m)}}{\partial \theta} \tag{3.24}$$

where $d^{(m)} = (\mathrm{v}^W_{w_t^{(m)}} \cdot h_c^{(m)})$ is the dot-product of the projection layer vector with the word vector for the middle word. Therefore,

$$\frac{\partial \log P_\Theta(Y_m = Y^{(m)})}{\partial \theta} = \left[Y^{(m)} - \sigma(\mathrm{v}^W_{w_t^{(m)}} \cdot h_c^{(m)}) \right] \frac{\partial (\mathrm{v}^W_{w_t^{(m)}} \cdot h_c^{(m)})}{\partial \theta} \tag{3.25}$$

For any training sequence m, there are four types of parameters θ that need to be updated. Firstly, the document vector for $d_i^{(m)}$, the middle word vector for word $w_t^{(m)}$, word vectors for context words $w_{t+j}^{(m)}$ and the neural network weights λ_i. The derivate $\frac{\partial(\mathrm{v}^W_{w_t^{(m)}} \cdot h_c^{(m)})}{\partial\theta}$ w.r.t. each of them is given by,

$$\frac{\partial(\mathrm{v}^W_{w_t^{(m)}} \cdot h_c^{(m)})}{\partial \mathrm{v}^D_{d_i^{(m)}}} = \mathrm{v}^W_{w_t^{(m)}} \tag{3.26}$$

$$\frac{\partial(\mathrm{v}^W_{w_t^{(m)}} \cdot h_c^{(m)})}{\partial \mathrm{v}^W_{w_t^{(m)}}} = h_c^{(m)} \tag{3.27}$$

$$\frac{\partial(\mathrm{v}^W_{w_t^{(m)}} \cdot h_c^{(m)})}{\partial \mathrm{v}^W_{w_{t+j}^{(m)}}} = \lambda_{t+j} * \mathrm{v}^W_{w_t^{(m)}} \tag{3.28}$$

$$\frac{\partial(\mathrm{v}^W_{w_t^{(m)}} \cdot h_c^{(m)})}{\partial \lambda_{t+j}} = \mathrm{v}^W_{w_t^{(m)}} \cdot \mathrm{v}^W_{w_{t+j}^{(m)}} \tag{3.29}$$

Therefore the derivative of the log-probability estimate w.r.t. the

1. Document Vector is given by,

$$\frac{\partial \log P_\Theta(Y_m = Y^{(m)})}{\partial \mathrm{v}^D_{d_i^{(m)}}} = \left[Y^{(m)} - \sigma(\mathrm{v}^W_{w_t^{(m)}} \cdot h_c^{(m)})\right] \mathrm{v}^W_{w_t^{(m)}} \tag{3.30}$$

2. Middle word is given by,

$$\frac{\partial \log P_\Theta(Y_m = Y^{(m)})}{\partial \mathrm{v}^W_{w_t^{(m)}}} = \left[Y^{(m)} - \sigma(\mathrm{v}^W_{w_t^{(m)}} \cdot h_c^{(m)})\right] h_c^{(m)} \tag{3.31}$$

3. Context words is given by,

$$\frac{\partial \log P_\Theta(Y_m = Y^{(m)})}{\partial \mathrm{v}^W_{w_{t+j}^{(m)}}} = \left[Y^{(m)} - \sigma(\mathrm{v}^W_{w_t^{(m)}} \cdot h_c^{(m)})\right] \lambda_{t+j} \mathrm{v}^W_{w_t^{(m)}} \tag{3.32}$$

4. Neural Network weights is given by

$$\frac{\partial \log P_\Theta(Y_m = Y^{(m)})}{\partial \lambda_{t+j}} = \left[Y^{(m)} - \sigma(\mathrm{v}^W_{w_t^{(m)}} \cdot h_c^{(m)})\right] (\mathrm{v}^W_{w_t^{(m)}} \cdot \mathrm{v}^W_{w_{t+j}^{(m)}}) \tag{3.33}$$

According to the SGD algorithm, the update to be made to a parameter $\theta \in \Theta$ on observing a training sequence m is therefore given in Eq. 3.34. We also include L_2 regularization for the parameters as it helps in avoiding overfitting and restricts the parameters to blow up in value.

$$\theta^{(i+1)} \leftarrow \theta^{(i)} + \gamma \left[\frac{\partial \log P_{\Theta}(Y_m = Y^{(m)})}{\partial \theta^{(i)}} - \beta \theta^{(i)} \right] \tag{3.34}$$

here $\theta^{(i)}$ denotes the value of the parameter in the i-th iteration, θ^{i+1} is the value after the update, γ is the learning rate and β is the regularization constant. The update rules for the different parameters are given below.

1. Update rule for the Document Vector,

$$(\mathrm{v}^D_{d_i^{(m)}})^{(i+1)} = (\mathrm{v}^D_{d_i^{(m)}})^{(i)} + \gamma \left[(Y^{(m)} - \sigma(\mathrm{v}^W_{w_t^{(m)}} \cdot h_c^{(m)})\mathrm{v}^W_{w_t^{(m)}} - \beta \mathrm{v}^D_{d_i^{(m)}} \right] \tag{3.35}$$

2. Update rule for the Middle Word Vector,

$$(\mathrm{v}^W_{w_t^{(m)}})^{(i+1)} = (\mathrm{v}^W_{w_t^{(m)}})^{(i)} + \gamma \left[(Y^{(m)} - \sigma(\mathrm{v}^W_{w_t^{(m)}} \cdot h_c^{(m)}))h_c^{(m)} - \beta \mathrm{v}^W_{w_t^{(m)}} \right] \tag{3.36}$$

3. Update rule for the Context Word Vectors,

$$(\mathrm{v}^W_{w_{t+j}^{(m)}})^{(i+1)} = (\mathrm{v}^W_{w_{t+j}^{(m)}})^{(i)} + \gamma \left[(Y^{(m)} - \sigma(\mathrm{v}^W_{w_t^{(m)}} \cdot h_c^{(m)}))\lambda_{t+j} \mathrm{v}^W_{w_t^{(m)}} - \beta \mathrm{v}^W_{w_{t+j}^{(m)}} \right] \tag{3.37}$$

4. Update rule for the Neural Network Weights,

$$\lambda_{t+j}^{(i+1)} = \lambda_{t+j}^{(i)} + \gamma \left[(Y^{(m)} - \sigma(\mathrm{v}^W_{w_t^{(m)}} \cdot h_c^{(m)}))(\mathrm{v}^W_{w_t^{(m)}} \cdot \mathrm{v}^W_{w_{t+j}^{(m)}}) - \beta \lambda_{t+j} \right] \tag{3.38}$$

To learn the vectors D, W and weights Λ we initialize vectors in D and W to small random vectors the weight vector Λ to all ones. We iterate through the training data for a fixed number of epochs, making the appropriate updates using Eqs. 3.35, 3.36, 3.37 and 3.38. For each training sequence we make one update to the document vector, $2c$ updates for the context word vectors in the sequence and the neural network weights, where c is the window length we consider while training. The complete algorithm for learning document and word vectors using our model is described in Algorithm 1.

Algorithm 1 Learning Document and Word Vector Representations

Input: $\mathbf{D}$, k, c, n, β, γ, *epochs*
Output: Document Vectors D, Word Vectors W
$\mathbf{V} \leftarrow Extractfrom(\mathbf{D})$
$\mathrm{D} \leftarrow random(\mathbb{R}^{k\times|\mathbf{D}|})$
$\mathrm{W} \leftarrow random(\mathbb{R}^{k\times|\mathbf{V}|})$
$\mathcal{T} \leftarrow Extractfrom(\mathbf{D}, c, n)$ ▷ $|\mathcal{T}| = M + nM$
$\Lambda \leftarrow \mathbf{1}^{2c}$ ▷ $2c$-sized vector of 1s
while $epochs \geq 1$ **do**
  for all $\{d_i, w_{t-c}, \ldots, w_{t+c}, Y\} \in \mathcal{T}$ **do**
    $h_c \leftarrow \mathrm{v}^D_{d_i} + \lambda_{t-c}\mathrm{v}^W_{w_{t-c}} + \ldots + \lambda_{t+c}\mathrm{v}^W_{w_{t+c}}$
    $\mathrm{v}^D_{d_i} \leftarrow \mathrm{v}^D_{d_i} + \gamma\left[(Y - \sigma(\mathrm{v}^W_{w_t}\cdot h_c))\mathrm{v}^W_{w_t} - \beta\mathrm{v}^D_{d_i}\right]$
    $\mathrm{v}^W_{w_t} \leftarrow \mathrm{v}^W_{w_t} + \gamma\left[(Y - \sigma(\mathrm{v}^W_{w_t}\cdot h_c))h_c - \beta\mathrm{v}^W_{w_t}\right]$
    for all $j \in \{t-c, \ldots, t-1, t+1, \ldots, t+c\}$ **do**
      $\mathrm{v}^W_{w_{t+j}} \leftarrow \mathrm{v}^W_{w_{t+j}} + \gamma\left[(Y - \sigma(\mathrm{v}^W_{w_t}\cdot h_c))\lambda_{t+j}\mathrm{v}^W_{w_t} - \beta\mathrm{v}^W_{w_{t+j}}\right]$
      $\lambda_{t+j} \leftarrow \lambda_{t+j} + \gamma\left[(Y - \sigma(\mathrm{v}^W_{w_t}\cdot h_c))(\mathrm{v}^W_{w_t}\cdot\mathrm{v}^W_{w_{t+j}}) - \beta\lambda_{t+j}\right]$
    end for
    $epochs \leftarrow epochs - 1$
  end for
end while
return D, W

3.3.2.7 Hyper-parameters of our model

Our model has various hyper-parameters that need to be tuned for optimum model performance and learning high quality document and word vectors. Below we describe the hyper-parameters in our model and the effect they have on learning document representations.

1. **Embedding Dimensionality (k)** : The most important hyper-parameter in our model is the size of the document and word embedding vectors, k. The embedding dimensionality needs to be large enough such that the document vectors can encode the different semantic topics across the corpus but shouldn't be very large so that it introduces noise in the vectors.

2. **Window Size (c)** : The length of the sequence or the window size c, that we consider as context surrounding a word is crucial for the quality of representations learned. While a smaller window could result in the negligence of important words that surround the middle word, a large window could introduce noise in the context that can deteriorate the performance of the model.

3. **Number of Negative Samples (n)** : In NCE, tuning the number of negative samples introduced in the training data per positive example is important for deciding the trade-off between learning better word density distribution, with larger n and lower training complexity, with smaller n.

4. **Number of Epochs ($epochs$))** : The number of times we loop through the training data to learn the representations needs to be optimum to prevent overfitting, that would occur with large epochs, while at the same time learn high quality representations.

5. **Learning Rate (γ)** : The convergence of a non-convex objective being optimized using the SGD algorithm depends largely on the learning rate. While a smaller learning rate could harm the training time and convergence, a larger learning rate may lead to the case of non-convergence and poor parameters.

6. **Regularization Constant (β)** : Regularization is introduced in the training objective to avoid overfitting of parameters by reducing model complexity. While a small β may not penalize complexity enough, larger constants may inhibit the growing of parameters with a negative effect which requires careful selection of the regularization constant dependent on the dataset.

We explain the technique in which we tune these hyper-parameters later in Sec. 5.2.

Chapter 4

Multi-Label Document Categorization

In this chapter, we present the multinomial logistic regression algorithm in the context of multi-label document categorization and give an overview of the training data required for the task. We discuss the algorithm's similarity to relational learning and matrix factorization and also present its advantages over other learning algorithms.

4.1 Logistic Regression for Multi-label Document Categorization

Introduced by Hosmer and Lemeshow [16], Logistic Regression (LR) is a probabilistic binary classification regression model that, given binary labeled data, performs regression over the data and learns weight vectors to predict whether a given data point belongs to the positive or the negative class. The probability of the data point to belong in a class is estimated using the *logistic (sigmoid) function*, hence the name logistic regression.

Logistic Regression, though is a technique to discriminate between two categories, can be easily extended to classification between multiple categories which is then referred to as Multinomial Logistic Regression. Though we use use multinomial logistic regression for our task of multi-label text classification, for the sake of brevity we refer to our algorithm as logistic regression.

In the sections below we describe the logistic regression model as modified for the task of multi-label document categorization.

4.1.1 Training Data

The training data $\mathcal{T}$ is composed of a set of documents $\mathbf{D}$, set of categories $\mathbf{C}$, data about which documents belong to each of the categories and the document representation vectors.

Document-Category Data : Each document d_i in $\mathbf{D}$ belongs to atleast one category from $\mathbf{C}$. To store this relational data between the documents and the categories, we create a database $\mathcal{D}$ in which for the m-th training instance we store a tuple of the form $\{d_i^{(m)}, c_j^{(m)}, y^{(m)}\}_{m=1}^{m=T}$ where $y^{(m)} \in \{0, 1\}$ denotes whether the document $d_i^{(m)}$ belongs the category $c_j^{(m)}$ or not.

Mostly the data about document categories is given such that it is known what categories do the documents belong to, without conclusive information about whether the document necessarily does not belong to the rest of the categories. In such cases, if we assume the given data to be complete, then along with positive data examples of the form, $\{d_i, c_j, 1\}$, we introduce negative samples, $\{d_i, c_k, 0\}$ for every category c_k each document d_i does not belong to. If the document-category data is viewed as a matrix with documents as rows and categories as columns, then in such case, we would only observe positive examples (1) in matrix but at sparse locations. To make the training data complete in such cases, we would fill the matrix with negative examples (0) at every empty location.

Document Representations : Along with the document-category data, the training data also composes of the contents of the documents in terms of the appropriate document representation vectors. For every document $d_i \in \mathbf{D}$ indexed by i, the training data comprises of a vector representation v_{d_i} for the document.

4.1.2 Logistic Regression Model

For multi-label document categorization, we extend the standard binary classification logistic regression model to, learn a probabilistic function from the given multi-labeled document category data such that for any given test document-category pair $\{d_i, c_j\}$, our function can estimate the probability of the document d_i belonging to the category c_j, $p(c_j = 1|d_i)$.

As we explained in Sec. 3.3, we learn low-rank distributed vector representation for every document in the corpus. Similarly, for multi-label logistic regression, we represent each category $c_i \in \mathbf{C}$ using a low-rank embedding $\mathrm{v}^C_{c_i} \in \mathbb{R}^k$ of the same dimensionality as the document embeddings, k. Similar to D, we stack these category embeddings as columns in the matrix $\mathrm{C} \in \mathbb{R}^{k \times |\mathbf{C}|}$.

Given a document-category tuple of the form $\{d_i, c_j\}$, we estimate the probability of the document belonging to the category $(y = 1)$ using the logistic function as,

$$P(y = 1|d_i, c_j, \mathrm{D}, \mathrm{C}) = \sigma(\mathrm{v}^D_{d_i} \cdot \mathrm{v}^C_{c_j}) \tag{4.1}$$

This model is similar to the standard logistic regression (LR) as in standard LR for binary classification, we learn a universal weight vector $\mathbf{w}$ that is used to estimate the probability in Eq. 4.1 instead of $\mathrm{v}^C_{c_j}$. Here, because we have multiple categories, we learn multiple weight vectors (category embeddings) for each category separately and hence perform multiple binary classifications.

4.1.2.1 Learning Objective

As explained in Sec. 4.1.1, the training data $\mathcal{T}$, is composed of T tuples of the form $\{d^{(m)}_i, c^{(m)}_j, y^{(m)}\}$. Our training objective involves learning optimum category embeddings such that for any unobserved document $d_x \notin \mathbf{D}$, we should be able to predict the categories it belongs to.

For the m-th training instance $\{d^{(m)}_i, c^{(m)}_j, y^{(m)}\}$, we denote the prediction that whether the document $d^{(m)}_i$ belongs the category $c^{(m)}_j$ by y_m. Therefore, if we predict

$d_i^{(m)}$ belongs $c_j^{(m)}$ i.e. $y_m = 1$ using Eq. 4.1,

$$P(y_m = 1|d_i^{(m)}, c_j^{(m)}, \mathrm{D}, \mathrm{C}) = \sigma(\mathrm{v}^D_{d_i^{(m)}} \cdot \mathrm{v}^C_{c_j^{(m)}}) \tag{4.2}$$

We denote the above probability estimate as $P_{\mathrm{D,C}}(y_m = 1)$ for brevity. Therefore,

$$P_{\mathrm{D,C}}(y_m = 0) = 1 - \sigma(\mathrm{v}^D_{d_i^{(m)}} \cdot \mathrm{v}^C_{c_j^{(m)}}) \tag{4.3}$$

$$P_{\mathrm{D,C}}(y_m) = \sigma(\mathrm{v}^D_{d_i^{(m)}} \cdot \mathrm{v}^C_{c_j^{(m)}})^{y_m}(1 - \sigma(\mathrm{v}^D_{d_i^{(m)}} \cdot \mathrm{v}^C_{c_j^{(m)}}))^{1-y_m} \tag{4.4}$$

To learn the optimum parameter set $\Theta = (\mathrm{C})$ consisting of the set of category embeddings, we would maximize the log-likelihood of observing the training data,

$$\hat{\Theta} = \arg\max_{\Theta} \ l(\mathcal{T}, \Theta) \tag{4.5}$$

$$l(\mathcal{T}, \Theta) = \sum_{m=1}^{T} \log P_{\mathrm{D,C}}(y_m = y^{(m)}) \tag{4.6}$$

where,

$$\log P_{\mathrm{D,C}}(y_m = y^{(m)}) = y^{(m)} \log \sigma(\mathrm{v}^D_{d_i^{(m)}} \cdot \mathrm{v}^C_{c_j^{(m)}}) + (1 - y^{(m)}) \log(1 - \sigma(\mathrm{v}^D_{d_i^{(m)}} \cdot \mathrm{v}^C_{c_j^{(m)}})) \tag{4.7}$$

4.1.2.2 Parameter Estimation

To learn the optimum parameters $\Theta = (\mathrm{C})$ we would maximize the log-likelihood of observing the training data given in Eq 4.6 using the Stochastic Gradient Descent(SGD) algorithm as described earlier in Sec 3.3.2.6. We first need to calculate the gradient of the log probability estimate (Eq. 4.7) with respect to the category embeddings which is given by,

$$\frac{\partial \log P_{\mathrm{D,C}}(y_m = y^{(m)})}{\partial \mathrm{v}^C_{c_j^{(m)}}} = \left[y^{(m)}\frac{1}{\sigma(s^{(m)})} - (1-y^{(m)})\frac{1}{(1-\sigma(s^{(m)}))}\right]\frac{\partial \sigma(s^{(m)})}{\partial \mathrm{v}^C_{c_j^{(m)}}} \tag{4.8}$$

$$= \left[y^{(m)}\frac{1}{\sigma(s^{(m)})} - (1-y^{(m)})\frac{1}{(1-\sigma(s^{(m)}))}\right]\left[\sigma(s^{(m)})(1-\sigma(s^{(m)}))\right]\frac{\partial s^{(m)}}{\partial \mathrm{v}^C_{c_j^{(m)}}} \tag{4.9}$$

$$\frac{\partial \log P_{\mathrm{D,C}}(y_m = y^{(m)})}{\partial \mathrm{v}^C_{c_j^{(m)}}} = \left[y^{(m)} - \sigma(s^{(m)})\right]\frac{\partial s^{(m)}}{\partial \mathrm{v}^C_{c_j^{(m)}}} \tag{4.10}$$

where, $s^{(m)} = (\mathrm{v}^D_{d_i^{(m)}} \cdot \mathrm{v}^C_{c_j^{(m)}})$ is the *pre-sigmoid activation*. Therefore,

$$\frac{\partial \log P_{\mathrm{D,C}}(y_m = y^{(m)})}{\partial \mathrm{v}^C_{c_j^{(m)}}} = \left[y^{(m)} - \sigma(\mathrm{v}^D_{d_i^{(m)}} \cdot \mathrm{v}^C_{c_j^{(m)}})\right]\frac{\partial(\mathrm{v}^D_{d_i^{(m)}} \cdot \mathrm{v}^C_{c_j^{(m)}})}{\partial \mathrm{v}^C_{c_j^{(m)}}} \tag{4.11}$$

$$\frac{\partial \log P_{\mathrm{D,C}}(y_m = y^{(m)})}{\partial \mathrm{v}^C_{c_j^{(m)}}} = \left[y^{(m)} - \sigma(\mathrm{v}^D_{d_i^{(m)}} \cdot \mathrm{v}^C_{c_j^{(m)}})\right]\mathrm{v}^D_{d_i^{(m)}} \tag{4.12}$$

According to the SGD algorithm and Eq. 3.34, the update to be made to the category embedding on observing the m-th training instance is given by Eq. 4.13. We also include L_2 regularization for the category embeddings as it helps in avoiding overfitting and restricts the embeddings to blow up in value.

$$(\mathrm{v}^C_{c_j^{(m)}})^{(i+1)} = (\mathrm{v}^C_{c_j^{(m)}})^{(i)} + \gamma\left[(y^{(m)} - \sigma(\mathrm{v}^D_{d_i^{(m)}} \cdot (\mathrm{v}^C_{c_j^{(m)}})^{(i)})\mathrm{v}^D_{d_i^{(m)}} - \beta(\mathrm{v}^C_{c_j^{(m)}})^{(i)}\right] \tag{4.13}$$

Here $(\mathrm{v}^C_{c_j^{(m)}})^{(i)}$ and $(\mathrm{v}^C_{c_j^{(m)}})^{(i+1)}$ are the category embeddings before and after the update, respectively, γ is the learning rate of the algorithm and β is the regularization constant used for the L_2 regularization.

Similar to Sec. 3.3.2.6, we initialize the category embeddings C to small random vectors and loop through the training data $\mathcal{T}$ until we reach convergence based on the development data. The complete algorithm for employing the logistic regression model for multi-label document categorization is given in Algorithm 2

Algorithm 2 Learning Category Vector Representations

Input: D, $\mathbf{C}$, $\mathcal{T}$, k, β, γ
Output: Category Vectors C
C $\leftarrow random(\mathbb{R}^{k\times|\mathbf{C}|})$
while not converged **do**
  for all $\{d_i, c_j, y\} \in \mathcal{T}$ **do**
    $\mathrm{v}^C_{c_j} \leftarrow \mathrm{v}^C_{c_j} + \gamma\left[(y - \sigma(\mathrm{v}^D_{d_i}\cdot\mathrm{v}^C_{c_j}))\mathrm{v}^D_{d_i} - \beta\mathrm{v}^C_{c_j}\right]$
  end for
end while
return C

4.2 Similarity to Relational Learning

Multi-label document categorization can be viewed as a relational learning problem where the task is to find missing links between documents and categories or for new documents find what categories does it link to. Relational learning has a novel solution in Matrix Factorization which assumes that the relational data matrix between entities has a low-rank structure and tries to factorize it as a product of two matrices representing the row and column entity factors.

Therefore, if R denotes the binary relational matrix, where rows and columns correspond to the entities of two different type and the entries of the matrix, $r(i,j) \in \{0,1\}$ denotes the presence/absence of link between the i-th row and the j-th column entity. Matrix factorization would try to decompose the matrix R into row and column factors, say U and V such that,

$$R = UV^T \tag{4.14}$$

where, if $R \in \mathbb{R}^{m\times n}$ then $U \in \mathbb{R}^{m\times k}$ and $V \in \mathbb{R}^{n\times k}$, where $k < min(m,n)$ is the approximate rank of R. The rows of the matrix U and V store the factors/embeddings for the entities. As we see, such an approach tries to learn low-rank embeddings for the row and column entities and projects them on to the same k-dimensional space.

In our model of logistic regression for multi-label document classification, though we take a similar approach and learn category embeddings to project the categories along with the documents to the same k-dimensional space, our method has the following differences,

1. Instead of factorizing the document-category data matrix, R using a linear matrix factorization approach as in Eq. 4.14, we take a probabilistic approach and think of R has a probabilistic database and estimate the probability of a document belonging to a particular category.

2. As we already learn document representations, D using our model described in Sec. 3.3.2, we only learn category factors, C instead of learning factors for both kind of entities as done in Eq. 4.14 by learning both U and V.

4.3 Advantages of Logistic Regression Learning Algorithm

As shown above, we use a modified version of the Logistic Regression for multi-label document categorization that learns multiple classifiers (in terms of the category embeddings) for multi-labeling task. Even though learning algorithms that learn multiple independent classifiers have drawbacks as elucidated in Sec 2.2.1, our logistic regression style algorithm has a lot of advantages that make it a good choice for the document categorization task. Below we list some of its major advantages.

1. Since we learn distributed representations for the categories in the same k-dimensional space as the documents and words it enables us to estimate the similarity between unrelated entities. Similarity estimation is as simple as taking the dot-product of the corresponding representation vectors. For example, similarity between two categories or between categories and words can be estimated, would not be possible if some other learning technique was used.

2. The major drawback of algorithms that learn multiple independent classifiers is their inability to capture and exploit the correlations among categories from the training data. As we exploit the low-rank structure of the document-

category relational matrix by learning factors for categories we are able to learn correlations among categories in a way similar to collaborative filtering.

3. Generally, document-category data is accompanied by additional relational data about documents and/or categories which is often incomplete. As has been shown in Gupta and Singh [14], joint modeling of relations and entities using collective matrix factorization of multiple relational matrices can provide signification improvements in the relation prediction task. Our model, based on matrix factorization as shown in Sec 4.2, can easily be extended to incorporate additional relational data and aid in improved database completion.

4. The fact that we use Stochastic Gradient Descent to find the optimum parameters (category representations), in which the training examples are presented to the system one at a time makes our document categorization algorithm completely online. It enables us to incorporate streaming data and additional training data at any stage of the learning without any effort.

Chapter 5

Performance Evaluation

In this chapter we first introduce the datasets that we use for evaluating the efficacy of our document representations for the document categorization task. We then explain the details of our experimentation setup and the different document representation techniques that form strong baselines that we compare our model to. In the sections following that we present the results for assigning categories to new documents and also imputing missing categories for documents for which we already have prior category information.

5.1 Datasets

We perform our experimentation on 5 datasets that contain rich data about documents belonging to multiple categories simultaneously. One of the 5 datasets we use is the famous Reuters-21578 collection, which is considered the benchmark for text classification evaluation. Along with the being richly multi-label and large-sized, Reuters-21578 has been used for many years for evaluation which gives us the opportunity to compare our accuracy with the previous state-of-the-art results. The other 4 datasets that we evaluate on are, exclusive subsets of documents, extracted from Wikipedia.

5.1.1 Reuters-21578

Reuters-21578 collection consists of documents published on the Reuters newswire in 1987. As the name suggests it has a total of 21758 documents assigned categories from a set of 135 categories. Though most of the documents in the collection are multi-labeled, many documents are assigned only a single category. The Reuters-21578 dataset has over the years become a standard dataset for evaluating many information retrieval algorithms due to the the multi-label nature of the collection and the large number of categories present in the collection that are overlapping and non-exhaustive. Relationships between the categories also makes it an interesting dataset to evaluate on, as the algorithms that capture the correlations between the categories are bound to perform better. Even though the dataset contains large number of documents and categories, it is very sparse making the learning difficult.

Though there exist many processed versions of the collection, the *ModApte* (Modified Apte) version is the most widely used version of the Reuters-21578 for evaluating multi-label classification algorithms. The *ModApte* version predefines a train/test split by considering all documents published after a specific date for testing purposes and the rest for training. After the split, only categories considered are the ones that have atleast one document in the training and the test set. The number of documents, categories, words and other statistics of the Reuters(*ModApte*) dataset are given in Table 5.1.

	|D|	**|C|**	**|V|**	**Data Points**	**Sparsity**
Train Set	7,767	90	39,853	9,585	0.0137
Test Set	3,019	90	39,853	3,745	0.0138

Table 5.1: Statistics of the Reuters-21578 (*ModApte*) Dataset

5.1.2 Wikipedia Datasets

Wikipedia[1] is a free-access free content Internet encyclopedia that contains articles about virtually anything possible. Along with the humongous amounts of articles, Wikipedia also has a hierarchical cyclic *Wikipedia Category Graph* that is used to label articles with the categories they belong in. Though the category graph is completely connected, it has few major top-level categories within which all subsequent categories fall. For eg. some of the top-level categories are *Culture and Arts*, *Geography*, *Health*, *History*, *Mathematics*, *Natural and Physical Sciences*, *Philosophy* etc. Each of the top-level categories are further divided into deep trees of fine-grained categories that are assigned to the articles.

The categories that are assigned to an article thus ranges from broader categories to much fine-grained that are very difficult to assign via an automated system. For eg. some of the categories assigned to an article on the musician *Jimi Hendrix* are *1942 Births*, *1970 deaths*, *American Rock Guitarists*, *Musicians from Seattle*, *Military Brats*, *Alcohol-related deaths in England.* Automatic categorization of such granularity requires the document representation to capture the different semantic topics in the document with great accuracy.

To test the efficacy of our model on such a diverse, recent and real-life dataset, we extracted documents from 4 top-level categories in Wikipedia, namely *Physics*, *Biology*, *Mathematics* and *Sports*, in the following manner. For each of the top-level category we compiled a list of all its child categories till a 3-level depth. To create the document-category dataset, we considered all the documents and the assigned categories, that belonged to the compiled category list. The number of documents, categories, words and other statistics of the extracted datasets are given in Table 5.2.

[1] www.wikipedia.org

	$\lvert D \rvert$	$\lvert C \rvert$	$\lvert V \rvert$	**Data Points**	**Sparsity**
Physics	4,229	2,999	81,614	14,070	0.0010
Biology	1,604	2,051	63,767	5,908	0.0018
Sports	1,529	2,829	59,058	3,745	0.0008
Mathematics	1,193	1,519	43,398	3,916	0.0013

Table 5.2: Statistics of the Wikipedia Datasets

5.2 Experimental Setup

In this section we describe in detail the data preprocessing steps, how we tune the hyper-parameters for both the document representation learning and the categorization algorithm, our evaluation techniques and the baselines we compare to.

Data Preprocessing : To curate the documents for learning their distributed representations we first split each document at sentence boundaries. We consider different sentences separately because we expect our system to learn syntactic qualities of the language which would not be possible if the sentence boundaries are ignored. All independent numbers in the documents are converted to '$\backslash num$' and numbers that are a part of a word are left as it is. Eg. *TH1RT3EN*, *Se7en*. To remove noise in the documents we only consider those words that occur atleast 5 times in the corpus. We preserve the capitalization of the words as, capitalization sometimes encodes a lot of semantic content that we do not wish to lose. Eg. *Apple* in most cases is used to refer to *Apple Inc.* (Company) rather than the fruit, *apple*. Models that distinguish between the two forms should learn better embeddings.

Learning Document Representations : To learn document representations using our model, we initialize the documents and word embeddings to small random vectors whose elements are drawn uniformly from the range $[-\frac{1}{k}, \frac{1}{k}]$. The value of the hyper-parameters of the representation learning model are chosen based on the performance on development data on the end-task of categorization. We could also choose the optimum hyper-parameters that minimize the development data perplexity on the document embedding learning task but as found in [29], lower perplexity does not ensure better representations but could also mean under-training. To choose the hyper-parameters we first use the default values as $k = 100$, $c = 2$, $n = 10$, *number of epochs = 50*, $\gamma = 0.0025$, $\beta = 0.1$. We found that the learning rate γ and the regularization constant β give best performance across datasets at their default value. To choose the value of the other parameters, we first tune the embedding dimensions k then the *number of epochs* after which we consider different window sizes (c) and finally the number of negative samples n for noise contrastive estimation. The values of the different hyper-parameters depend on the dataset. The noise distribution, $P_n(w)$ for NCE is chosen to be unigram distribution $U(w)$ raised to the $3/4rd$ power (i.e. $U(w)^{\frac{3}{4}}/Z$) where Z is the normalization factor. Other common choices for noise distribution are uniform distribution and unigram distribution but as found in previous works, $P_n(w) \sim U(w)^{\frac{3}{4}}$ works best.

Document Categorization : For the task of document categorization, we split the document category data into training, development and test data by keeping 10% documents for test purposes and 10% for development. The rest 80% are used for training purposes, i.e. learning category representations. We stop the training based on the convergence reached on the development data. The value of the learning rate $\gamma = 0.01$ and regularization constant $\beta = 0.01$ is chosen based on the performance on development data across different datasets. To make final binary predictions from the estimated probability we need to threshold it, value for which was chosen based on the development data across datasets. It was found that the default logistic threshold of 0.5 gave the best performance.

Evaluation Criteria : To evaluate the task of multi-label classification many evaluation criteria such as *Hamming Loss*, *Accuracy* and *F1 score* have been used. We use the *F1* score i.e. the harmonic mean of the precision and recall values to evaluate our model as the F1 score is a much more preferred measure compared to hamming loss or accuracy in the presence of imbalanced label distribution. We compute the F1 score in the micro-averaged fashion by combining prediction values for all the categories together for a particular dataset and then computing the precision and recall values. Such averaging ensures equal weightage to all test instances.

Baselines : To test the efficiency of our document representation learning algorithm, we compare our model's performance to some of the baseline methods that are explained below.

1. **Bag-of-Words** : The most widely used document representation is the bag-of-words representation with tf-idf weighting. It has been used to produce state-of-the-art results with various learning algorithms. We call this model the **BOW** model.

2. **Latent Semantic Indexing** : As explained in Sec 2.1.2, Latent Semantic Indexing is a popular dimensionality reduction technique that uses the term co-occurrence statistics to capture the latent semantic structure of the documents. We use LSI to learn 100-dimensional representation vectors for documents. We call this the **LSI-100** model.

3. **Word Vector Averaging (WordVecAvg)** : With the availability of word vectors learned using the NPLM or the word2vec model, the most simple method to learn document embeddings is to take a weighted average of the word embeddings to represent the document. This method is similar to the bag-of-words technique as it ignores word ordering. We show that the document representations learned using our method perform better than averaged word vector representations. We call this the **WordVecAvg** model. The weighing scheme used to take the weighted average is *tf-idf*.

4. **Probabilistic Matrix Factorization (PMF)** : The most simple technique for document categorization is the probabilistic matrix factorization of the document-category data matrix as explained in Sec. 4.2. In this model, instead of learning only category embeddings, we also learn document representations simultaneously. This model does not require any information about the document contents and also uses the document-category co-occurrence data. We call this the **PMF** model.

   Employing **PMF** for predicting categories from new documents is useless as they do not contain any prior information in the training data needed to learn document representations. **PMF** in such case would act as a trivial strawman. It can be used to predict categories for documents that already have category history in training data. Eg. For imputing missing categories in the database.

Apart from the above baselines, many more baselines for the *Reuters-21578* dataset exist in literature that primarily use bag-of-words representation but use different learning algorithms. We also compare our model against them.

5.3 Results

In this section we present our model's performance in categorizing new documents by learning distributed representations for the documents in the corpus and using training document-category data to learn category representations to predict categories for new documents. We compare our model's accuracy with baselines explained in the above section and also explain the process to choose hyper-parameters dependent on the dataset. We also present our model's performance in imputing missing categories for the Wikipedia articles and also present qualitative evaluation in estimating the similarities between categories and words.

5.3.1 Document Categorization

In this section we present our model's efficacy in assigning categories to documents with no prior category information.

For evaluation of a particular dataset, we first divide it into training, development and test sets by keeping 80% of the documents for training purposes and then dividing rest of the documents equally for development and testing. For each of the different hyper-parameter settings for the representation learning phase, we first learn document representations for all the documents. We then use the training document-category data to learn category representations and evaluate our model based on the development data. Using the accuracies obtained on the development data for different hyper-parameter settings we select the "*best*" model (hyper-parameters, document and category representations) which is then used to report results on the test data.

5.3.1.1 Reuters - 21578

The performance of our model on the development data for different values of different hyper-parameters is given in Table 5.3, 5.4, 5.5 and 5.6. Table 5.3 and 5.4 show that with increasing embedding dimensionality and the number of training epochs the accuracy of category prediction improves in terms of the F1 score. To avoid overfitting we use $k = 100$ and $epoch = 200$. In Table 5.5 we see that the default window size of $c = 2$ performs best and Table 5.6 shows that 10 - 15 negative samples per positive sample should be used for NCE.

Tuning	**Hyper-parameters** : $epochs = 50$, $c = 2$, $n = 10$								
	$k = 50$			$k = 100$			$k = 150$		
Reuters (Development)	P	R	**F1**	P	R	**F1**	P	R	**F1**
	88.7	89.9	89.3	90.9	89.8	**90.3**	90.9	89.8	**90.3**

Table 5.3: Model performance on Reuters development data for different embedding dimensionality

Tuning	**Hyper-parameters** : $k = 100$, $c = 2$, $n = 10$														
	$epochs = 50$			$epochs = 100$			$epochs = 150$			$epochs = 200$			$epochs = 250$		
Reuters (Development)	P	R	**F1**	P	R	**F1**	P	R	**F1**	P	R	**F1**	P	R	**F1**
	90.9	89.8	90.3	92.3	92.9	92.6	93.3	93.0	93.2	93.7	93.9	**93.8**	94.0	93.1	93.6

Table 5.4: Model performance on Reuters development data for different number of epochs

Table 5.7 compares our model's accuracy against different document representations and other learning algorithms that use bag-of-words representations. We find that document representations learned using our model performs the best and improves the previous state-of-the-art result, achieving a F1 score of 91.7%. Our representations improves the F1 score of 84.1%, achieved using bag-of-words rep-

Tuning	**Hyper-parameters** : $k = 100$, $epochs = 200$, $n = 10$								
	$c = 2$			$c = 3$			$c = 4$		
Reuters (Development)	P	R	**F1**	P	R	**F1**	P	R	**F1**
	93.7	93.9	**93.8**	93.2	90.9	92.0	91.8	90.8	91.3

Table 5.5: Model performance on Reuters development data for different window sizes

Tuning	**Hyper-parameters** : $k = 100$, $epochs = 200$, $c = 2$								
	$n = 5$			$n = 10$			$n = 15$		
Reuters (Development)	P	R	**F1**	P	R	**F1**	P	R	**F1**
	92.5	89.5	91.0	93.7	93.9	**93.8**	94.7	92.4	93.5

Table 5.6: Model performance on Reuters development data for different number of negative samples

resentation, by 9%. Improvements of 1%-6.6% on the F1 score are found against models such as SVM, CRF, LSI, MFoM. We also see that simple summing of word vectors to compute the projection layer reduces the performance of our model and hence leads to poor quality document representations. This corroborates with our hypothesis that weighted average of surrounding words is necessary to capture syntactic qualities and learn better quality representations. Similar observations are seen in the Precision/Recall curves in Fig. 5.1.

Reuters-21578	P	R	**F1**
BOW	77.8	91.5	84.1
LSI-100	84.8	96.7	90.4
WordVecAvg	94.1	88.1	91.0
CMLF (CRF)	-	-	87.0
SVM (poly)	-	-	86.0
SVM (rbf)	-	-	86.4
Binary-MFoM	-	-	88.4
MC-MFoM	-	-	88.8
Our Model (no weight)	92.1	86.1	89.0
Our Model (with weights)	94.1	89.3	**91.7**

Table 5.7: Precision/Recall/F1 for Document Categorization on Reuters-21578 (*ModApte*) dataset

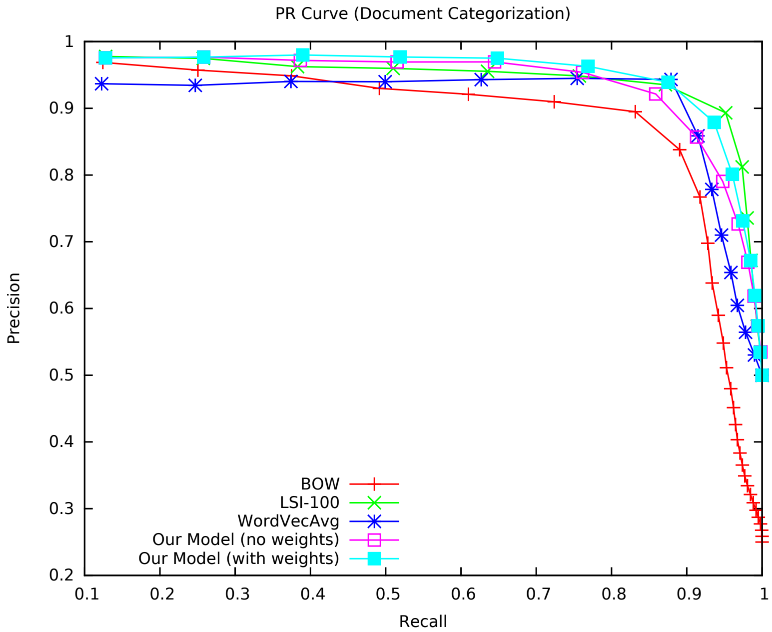


Figure 5.1: Precision/Recall for Document Categorization on Reuters-21578 (*ModApte*) dataset

5.3.1.2 Physics - Wikipedia

Performance of our model on the development data for different hyper-parameters is given in Table 5.8, 5.9, 5.10 and 5.11. We find that our model gives the best performance for $k = 100$ (Table 5.8) when trained for 150 epochs (Table 5.9). Table 5.10 shows that larger context window ($c = 3$) and default number of negative samples ($n = 10$) per positive sample gives the best performance.

	Hyper-parameters : $epochs = 50$, $c = 2$, $n = 10$								
Tuning	$k = 50$			$k = 100$			$k = 150$		
Physics (Development)	P	R	**F1**	P	R	**F1**	P	R	**F1**
	84.4	72.1	77.8	89.1	71.2	**79.2**	89.1	69.7	78.2

Table 5.8: Model performance on Physics(Wikipedia) development data for different embedding dimensionality

Table 5.12 shows that our model outperforms other baseline models with a huge margin. Our model achieves a F1 score of 79.7% while the second best model, **BOW** trails ours by 2.31%. As observed in the evaluation of Reuters-21578, syntactic features are necessary to learn high quality document representations. This is shown by the fact that when we simply sum the word vectors to represent the context, our

Tuning	Hyper-parameters : $k = 100$, $c = 2$, $n = 10$														
	$epochs = 20$			$epochs = 50$			$epochs = 100$			$epochs = 150$			$epochs = 200$		
Physics (Development)	P	R	**F1**	P	R	**F1**	P	R	**F1**	P	R	**F1**	P	R	**F1**
	85.5	70.6	77.3	89.1	71.2	79.2	86.9	73.8	79.9	87.4	73.6	**79.9**	86.9	73.3	79.5

Table 5.9: Model performance on Physics(Wikipedia) development data for different number of epochs

Tuning	Hyper-parameters : $k = 100$, $epochs = 150$, $n = 10$								
	$c = 2$			$c = 3$			$c = 4$		
Physics (Development)	P	R	**F1**	P	R	**F1**	P	R	**F1**
	87.4	73.6	79.9	86.5	74.4	**80.0**	87.0	73.1	79.5

Table 5.10: Model performance on Physics(Wikipedia) development data for different window sizes

Tuning	Hyper-parameters : $k = 100$, $epochs = 150$, $c = 3$								
	$n = 5$			$n = 10$			$n = 15$		
Physics (Development)	P	R	**F1**	P	R	**F1**	P	R	**F1**
	85.5	74.3	79.5	86.5	74.4	**80.0**	87.8	73.3	79.9

Table 5.11: Model performance on Physics(Wikipedia) development data for different number of negative samples

model only achieves an accuracy of 73.8% in terms of the F1 score. Fig. 5.2 also shows that our model performs the best while **BOW** model is the second best performing model.

Physics (Wikipedia)	P	R	**F1**
BOW	87.8	70.1	77.9
LSI-100	83.4	69.5	75.8
WordVecAvg	91.0	59.1	71.7
Our Model (no weights)	86.1	64.6	73.8
Our Model (with weights)	88.6	72.4	**79.7**

Table 5.12: Precision/Recall/F1 for Document Categorization on Physics(Wikipedia) dataset

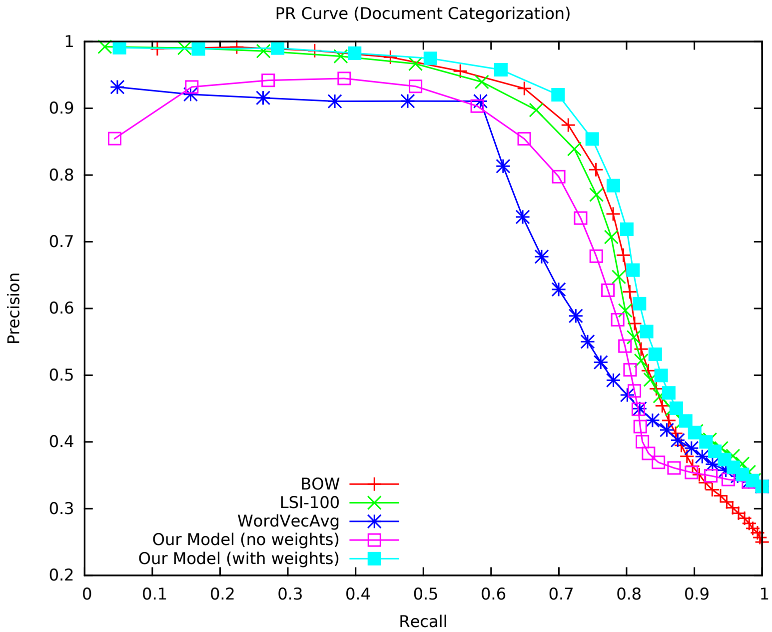


Figure 5.2: Precision/Recall for Document Categorization on Physics(Wikipedia) dataset

5.3.1.3 Biology - Wikipedia

The performance of our model on the development data for different values of hyper-parameters is shown in Table 5.13, 5.14, 5.15 and 5.16. Table 5.13 shows that default embedding size of $k = 100$ performs best and relatively low training epochs ($epochs = 50$) gives the best results (Table 5.14). Larger context window ($c = 3$) (Table 5.15) and default number of negative samples ($n = 10$) per positive sample (Table 5.16) gives the best model performance.

Tuning	**Hyper-parameters** : $epochs = 50$, $c = 2$, $n = 10$								
	$k = 50$			$k = 100$			$k = 150$		
Biology (Development)	P	R	**F1**	P	R	**F1**	P	R	**F1**
	83.9	60.0	70.0	82.4	61.2	**70.2**	83.7	60.2	70.0

Table 5.13: Model performance on Biology(Wikipedia) development data for different embedding dimensionality

Table 5.17 shows that the distributed representations learned by our model outperform some baseline representations but the bag-of-words representation works the best. Our model achieves a F1 score of 67.8% while the **BOW** improves our model by 1.7%. We also find that document representations learned by averaging

Tuning / Biology (Development)	Hyper-parameters : $k = 100$, $c = 2$, $n = 10$														
	$epochs = 20$			$epochs = 50$			$epochs = 100$			$epochs = 150$			$epochs = 200$		
	P	R	**F1**	P	R	**F1**	P	R	**F1**	P	R	**F1**	P	R	**F1**
	75.3	59.8	66.7	82.4	61.2	**70.2**	82.9	60.2	69.7	85.6	58.6	69.6	85.3	58.0	69.0

Table 5.14: Model performance on Biology(Wikipedia) development data for different number of epochs

Tuning / Biology (Development)	Hyper-parameters : $k = 100$, $epochs = 50$, $n = 10$								
	$c = 2$			$c = 3$			$c = 4$		
	P	R	**F1**	P	R	**F1**	P	R	**F1**
	82.4	61.2	**70.2**	80.8	62.0	**70.2**	81.3	61.4	70.0

Table 5.15: Model performance on Biology(Wikipedia) development data for different window sizes

Tuning / Biology (Development)	Hyper-parameters : $k = 100$, $epochs = 50$, $c = 3$								
	$n = 5$			$n = 10$			$n = 15$		
	P	R	**F1**	P	R	**F1**	P	R	**F1**
	81.8	61.0	69.9	80.8	62.0	**70.2**	81.2	60.8	69.6

Table 5.16: Model performance on Biology(Wikipedia) development data for different number of negative samples

word vectors performs the worst. The model accuracy suffers by 4.95% when context word weighting is ignored during training. Fig. 5.3 shows the precision/recall curves for our model and other baseline methods. It also shows that the **BOW** representations perform the closest to representations learned using our model.

Biology (Wikipedia)	P	R	**F1**
BOW	90.3	59.5	**69.0**
LSI-100	82.1	51.6	63.4
WordVecAvg	79.4	50.4	61.6
Our Model (no weights)	80.3	53.8	64.4
Our Model 79.7 R : 59.0 F1 : 67.8 (with weights)	79.7	59.0	67.8

Table 5.17: Precision/Recall/F1 for Document Categorization on Biology(Wikipedia) dataset

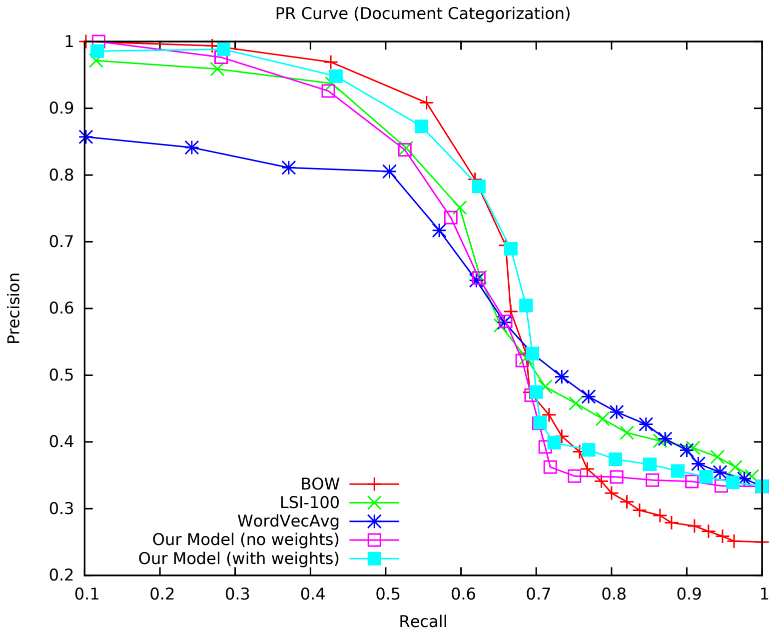


Figure 5.3: Precision/Recall for Document Categorization on Biology(Wikipedia) dataset

5.3.1.4 Mathematics - Wikipedia

Table 5.18, 5.19, 5.20 and 5.21 show that low embedding size of $k = 50$ and moderate training epochs ($epochs = 100$) give the best model performance. We also find that relatively larger context window ($c = 3$) and number of negative samples ($n = 15$) per positive sample are required for optimal model performance.

	Hyper-parameters : $epochs = 50$, $c = 2$, $n = 10$								
Tuning	$k = 50$			$k = 100$			$k = 150$		
Mathematics (Development)	P	R	**F1**	P	R	**F1**	P	R	**F1**
	83.2	57.0	**67.7**	82.3	56.0	66.7	86.1	55.5	67.5

Table 5.18: Model performance on Mathematics(Wikipedia) development data for different embedding dimensionality

Table 5.22 shows that the distributed representations learned by our model outperform other baseline representations. Our model achieves a F1 score of 68.2% while the second best model, **BOW** trails ours by 4.41%. Word vectors averaged by tf-idf weighing scheme to represent document in the Mathematics (Wikipedia) dataset give the worst accuracy with an F1 score of 55.7%. We again find that modeling syntactic qualities of the language is important as weighting word vectors to

Tuning	Hyper-parameters : $k = 50$, $c = 2$, $n = 10$														
	$epochs = 20$			$epochs = 50$			$epochs = 100$			$epochs = 150$			$epochs = 200$		
Mathematics (Development)	P	R	**F1**	P	R	**F1**	P	R	**F1**	P	R	**F1**	P	R	**F1**
	81.2	55.2	65.8	83.2	57.0	67.7	87.0	56.3	**68.3**	86.7	55.2	67.5	82.2	57.8	67.9

Table 5.19: Model performance on Mathematics(Wikipedia) development data for different number of epochs

Tuning	Hyper-parameters : $k = 50$, $epochs = 100$, $n = 10$								
	$c = 2$			$c = 3$			$c = 4$		
Mathematics (Development)	P	R	**F1**	P	R	**F1**	P	R	**F1**
	87.0	56.3	**68.3**	91.0	54.5	68.2	88.1	55.0	67.7

Table 5.20: Model performance on Mathematics(Wikipedia) development data for different window sizes

Tuning	Hyper-parameters : $k = 50$, $epochs = 100$, $c = 2$								
	$n = 5$			$n = 10$			$n = 15$		
Mathematics (Development)	P	R	**F1**	P	R	**F1**	P	R	**F1**
	86.1	57.0	68.6	87.0	56.3	68.3	90.0	55.5	**68.7**

Table 5.21: Model performance on Mathematics(Wikipedia) development data for different number of negative samples

represent the context gives the best performance. Fig. 5.4 shows the precision/recall curves for our model and other baseline methods.

Mathematics (Wikipedia)	P	R	**F1**
BOW	65.6	65.1	65.3
LSI-100	89.7	50.3	64.4
WordVecAvg	90.5	40.3	55.7
Our Model (no weights)	78.4	57.4	66.3
Our Model (with weights)	85.3	56.8	**68.2**

Table 5.22: Precision/Recall/F1 for Document Categorization on Mathematics(Wikipedia) dataset

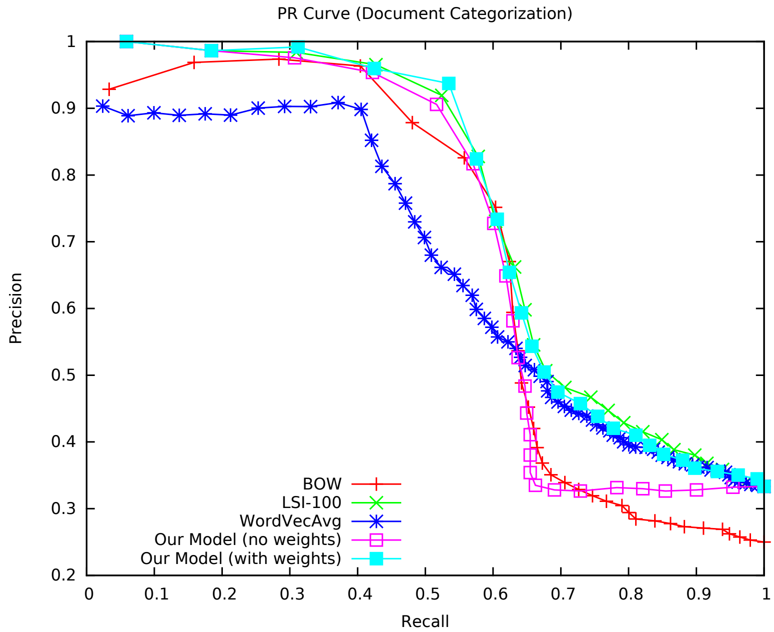


Figure 5.4: Precision/Recall for Document Categorization on Mathematics(Wikipedia) dataset

5.3.1.5 Sports - Wikipedia

The performance of our model on the development data for different values of different hyper-parameters is shown in Table 5.23, 5.24, 5.25 and 5.26. Table 5.23 shows that low vector size of $k = 50$ and Table 5.24 shows moderate training epochs ($epochs = 100$) gives the best model performance. Table 5.25 shows that context window size of $c = 3$ and relatively larger number of negative samples ($n = 15$) per positive sample achieves the best performance.

Tuning	**Hyper-parameters** : $epochs = 50$, $c = 2$, $n = 10$								
	$k = 50$			$k = 100$			$k = 150$		
Sports (Development)	P	R	**F1**	P	R	**F1**	P	R	**F1**
	82.2	45.7	**58.8**	81.2	45.9	58.6	80.9	45.7	58.4

Table 5.23: Model performance on Sports(Wikipedia) development data for different embedding dimensionality

Table 5.27 shows that the distributed representations learned by our model outperform other baseline representations. Our model achieves a F1 score of 57.3% while again the **BOW** model is second best performing. As found in previous evaluations, document representations using bag-of-words style word vector averaging

	Hyper-parameters : $k = 50$, $c = 2$, $n = 10$														
Tuning	$epochs = 20$			$epochs = 50$			$epochs = 100$			$epochs = 150$			$epochs = 200$		
Sports (Development)	P	R	**F1**	P	R	**F1**	P	R	**F1**	P	R	**F1**	P	R	**F1**
	73.9	45.9	56.6	82.2	45.7	58.8	83.9	47.6	**60.7**	83.7	46.8	60.0	67.1	50.2	57.4

Table 5.24: Model performance on Sports(Wikipedia) development data for different number of epochs

	Hyper-parameters : $k = 50$, $epochs = 100$, $n = 10$								
Tuning	$c = 2$			$c = 3$			$c = 4$		
Sports (Development)	P	R	**F1**	P	R	**F1**	P	R	**F1**
	83.9	47.6	60.7	84.5	47.8	**61.0**	86.0	47.0	60.8

Table 5.25: Model performance on Sports(Wikipedia) development data for different window sizes

	Hyper-parameters : $k = 50$, $epochs = 100$, $c = 3$								
Tuning	$n = 5$			$n = 10$			$n = 15$		
Sports (Development)	P	R	**F1**	P	R	**F1**	P	R	**F1**
	84.6	47.0	60.4	84.5	47.8	**61.0**	82.7	47.4	60.3

Table 5.26: Model performance on Sports(Wikipedia) development data for different number of negative samples

perform the worst and modeling syntactic features is important for learning the best quality document representations. Fig. 5.5 shows the precision/recall curves for our model and other baseline methods.

Sports (Wikipedia)	P	R	**F1**
BOW	91.7	41.3	56.9
LSI-100	91.2	40.1	55.7
WordVecAvg	81.8	37.5	51.4
Our Model (no weights)	80.5	40.1	53.6
Our Model (with weights)	82.1	44.0	**57.3**

Table 5.27: Precision/Recall/F1 for Document Categorization on Sports(Wikipedia) dataset

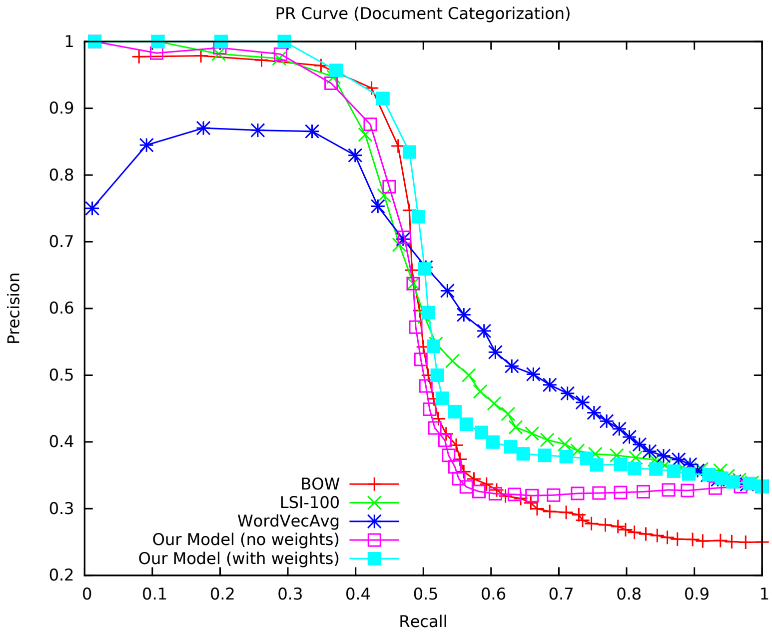


Figure 5.5: Precision/Recall for Document Categorization on Sports(Wikipedia) dataset

5.3.2 Imputing Missing Categories

Most of the real-life databases that contain categorization for documents contain incomplete information. In the case of huge number of categories and when document categorization is carried out by non-experts, as in Wikipedia, one can never be sure that all the relevant categories have been assigned to each document. In such large-scale databases where manual annotation and review is difficult, systems that can automatically impute missing categories for documents in the database are imperative.

In this section we evaluate our system's efficacy in imputing missing categories in the Wikipedia datasets. The task is similar to Relational Learning where the task is to complete the sparse relational matrix, R between two types of entities. In our case, we estimate the probability of every unobserved document-category pair (d_i, c_k) to be true to complete the sparse document-category matrix in which only positive document-category pairs are observed. For training and testing purposes, we first introduce corrupt document-category pairs as negative samples in the training data $\mathcal{T}$ in the following manner. For every positive $\{d_i, c_j, 1\}$ observed in the training data, we introduce two negative samples $\{d_i, c_k, 0\}$ where the negative sampled category c'_k is chosen uniformly for the category set. This increases the tuples in the training data by three times. From the modified training data we randomly choose 80% document-category pairs for training purposes and divide the rest equally for development and test purposes. The document representations are learned in the same manner as in the previous evaluation.

Table 5.28 shows our model's efficacy in imputing missing categories in the different Wikipedia datasets. We also compute a combined F1 score for all the datasets in the micro-averaged fashion, i.e. by combining all the prediction for all the datasets and then computing the precision and the recall. As expected our model of learning distributed document representations outperforms all other baselines by a large margin. Our model achieves an overall F1 score of 74.3% which improves upon the bag-of-words (**BOW**) model by 4.78%. Representing documents by averaging the word vectors using *tf-idf* weighing scheme achieves a F1 score of 65.4%. The worst performing model, as expected is **PMF** which does not use any textual information about the documents. As found in the previous section, in our model, simple summing of context word vectors to represent the context does not provide competitive results which corroborates the hypothesis that encoding syntactic information is necessary.

	Physics			Biology			Mathematics			Sports			**Combined**		
	P	R	**F1**	P	R	**F1**	P	R	**F1**	P	R	**F1**	P	R	**F1**
PMF	73.0	64.3	68.4	72.1	47.5	57.3	41.6	58.2	48.5	51.3	35.6	42.0	63.0	54.8	58.6
LSI-100	59.5	82.3	69.0	49.9	71.6	58.8	47.1	73.0	57.3	43.1	68.2	52.8	52.5	76.3	62.2
BOW	76.1	79.4	77.7	69.7	67.7	68.7	70.9	63.5	67.0	64.8	49.3	56.0	72.5	69.4	70.9
WordVecAvg	88.0	63.5	73.8	80.7	50.3	61.9	71.8	46.7	56.6	87.2	35.4	50.3	84.2	53.4	65.4
Our Model (without weights)	88.6	69.1	77.7	80.5	55.3	65.6	74.3	53.1	61.9	84.7	40.2	54.5	85.4	58.5	69.2
Our Model (with weights)	89.9	74.5	**81.5**	84.9	63.8	**72.9**	79.9	60.7	**69.0**	81.1	45.6	**58.4**	86.3	65.2	**74.3**

Table 5.28: Precision/Recall/F1 for Imputing Missing Categories in the Wikipedia Datasets

5.3.3 Estimating Similarity between Categories and Words

One of the primary drawbacks faced by the bag-of-words representations is the lack of similarity measures due to the discrete nature of the representation vectors. Also, when documents are represented using bag-of-words vectors, there is no representation for the words in the vocabulary. As we show in our model, we embed words and documents in the same k-dimensional space such that semantically similar entities have similar vector representations.

One of the biggest advantages of using the modified logistic regression algorithm is that we learn distributed category representations in the same k-dimensional space as the semantic space of words and documents. Such representations allow us to compute similarity between indirectly related entities such as words and categories. Estimating the similarity or distance between two entities is equivalent to taking the dot-product between their representation vectors.

In Table 5.29 we present a few random categories along with their nearest neighboring words as found using the representations learned using our model. We see that the representations learned using our model can successfully encode the semantic similarity between entities, categories and words in this case. For example, we see that the words closest to the category *Evolutionary Biology* are *Darwin*, *lineage*, *phylogenetics*, closest to category *Thermodynamics* are *convection*, *enthalpy*, *calorimetric* and closest to category *Theoretical Physicists* are world famous physicists such as *Dipankar*, *Uri*, *Aneesur*.

Category	**Nearest Neighbors**
Evolutionary Biology	gene, phylogenetics, speciation, ancestor, Darwin, lineage, evolutionary, interbreeding
Statistical Mechanics	ergodicity, Eigenstate, Universality, DMFT, Markovian, Parisi, Combinatorics
Thermodynamics	Convection, ecosystem, Enthalpy, Joule, calorimetric, compressible, Thermodynamic
Trade	import, Pledges, Tariff, Trade, competitiveness, toll, billion, basket, Ditch, Worldwide
Money-FX	Borrowing, franc, banker, Currency, banks, nervous, sideways, Markets, FORWARD
Virology	nucleoside, ribozyme, adenoviruses, Virology, retroviruses, poliovirus, Viroid
Neurobiology	purinergic, cyclase, vertebral, Ehrlich, nexus, steroid, lean, gendered, reticular
Physical Exercise	Fitness, aerobics, metabolic, workout, Exercise, Stretching, pelvic, Physiology, fibers
Algebra	subalgebra, Algebras, nilpotent, adjoints, octonions, bicommutant, diagonalizable
Theoretical Physicists	Dipankar, DSc, Hubert, Aneesur, Uri, Ignaz, Chia, Stig, Diderot, Dannie
Mathematical Physics	covectors, pseudotensor, spacelike, dyadic, Curl, torque, contractions, wavefunctions
Sports Venues	stadion, decoration, tracks, seating, buildings, parcourse, architectural, arenas, circular
Indian Mathematics	utkrama, ecliptic, Siddhanta, Hellenistic, Brahmi, sexagesimal, scribe, Islamic, Sanskrit

Table 5.29: Estimating Similarity between Categories and Words

Chapter 6

Conclusions and Future Work

We presented an unsupervised neural network model that jointly learns fixed-length low-dimensional distributed vector representations for documents and words that encode the semantic content of the documents and words for multi-label document categorization. We overcome some of the issues in the bag-of-words representations by encoding the contextual information surrounding the words in the documents in our representations to improve their quality and performance on the categorization task. Our neural network architecture is a linear-model that uses Noise Contrastive Estimation (NCE) to approximate the word probability distribution making parameter learning computationally inexpensive. We use the Stochastic Gradient Descent (SGD) to minimize the training objective that allows parallelization of the learning task further decreasing training time many folds.

We use a modified version of the logistic regression algorithm that learns distributed category representations by embedding categories in the same low-dimensional space as the documents and words for the multi-label document categorization task. On the standard *Reuters-21578* dataset we show that using representations learned using model we achieve a F1 score of 91.7% improving the bag-of-words representation accuracy by 9.03% and the previous state-of-the-art, Multi-Class Maximum Figure-of-Merit (MC-MFoM) by 3.26% in terms of the F1 score. We also present evaluations on the Wikipedia datasets showing that our representations perform better than the bag-of-words representations. We also show our model performs better than the bag-of-words representation at the task of imputing missing categories for existing articles on Wikipedia. Using continuous vector representations we embed documents, words and categories in the same semantic space allowing us to estimate similarities between indirectly related entities such as words and categories. We qualitatively show that representations learned using our model capture the semantic similarity between categories and words effectively.

6.1 Future Work

6.1.1 Improving Compositionality of Word Vectors

Human language is complex and change in word ordering and syntax can completely change the semantic meaning of a sentence and hence longer pieces of text. As we see in our performance evaluation, encoding syntactic nature of the language in terms of weighting the context words leads to better document representations than weighing all context words equally. In our model, we do not preserve word ordering and do not consider the parse tree of the sentences in documents. Recursive Neural Tensor Networks [41] have shown to be effective in composing word vectors to learn sentence level representations for sentiment analysis. We would like to extend our model to incorporate syntactic dependencies in sentences for more effective compositionality of word vectors to the document level.

6.1.2 Multi-view Supervised Representation Learning

In our model, we learn universal document representations in an unsupervised manner for the task of multi-label document categorization. Though such document representations are general purpose and can be used as inputs for any document level tasks, such as sentiment analysis, better performance accuracies on the required task can be achieved by making the document representation learning model supervised with the help of training data for document categorization. This would lead to joint learning of document and category vectors though the drawback being document representations will lose they generality.

As discussed in Sec. 4.3, one of the advantages of the learning distributed representations is that additional relational information about documents can be incorporated to the task of document categorization to improve the quality of learned representations. Also, as shown in Gupta and Singh [14], joint modeling of relational data about entities can boost the performance of predicting relations while learning high quality distributed representations that can encode dependencies between unrelated entities.

Bibliography

[1] Y. Bengio and J.-S. Senecal. Adaptive importance sampling to accelerate training of a neural probabilistic language model. *Neural Networks, IEEE Transactions on*, 19(4):713–722, 2008.

[2] Y. Bengio, R. Ducharme, P. Vincent, and C. Janvin. A neural probabilistic language model. *The Journal of Machine Learning Research*, 3:1137–1155, 2003.

[3] Y. Bengio, J.-S. Senécal, et al. Quick training of probabilistic neural nets by importance sampling. In *AISTATS Conference*, 2003.

[4] L. Bottou. From machine learning to machine reasoning. *Machine learning*, 94 (2):133–149, 2014.

[5] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa. Natural language processing (almost) from scratch. *The Journal of Machine Learning Research*, 12:2493–2537, 2011.

[6] W. Cooper, A. Chen, and F. Gey. Full text retrieval based on probabilistic equations with coefficients fitted by logistic regression. *NIST SPECIAL PUBLICATION SP*, pages 57–57, 1994.

[7] C. Cortes and V. Vapnik. Support-vector networks. *Machine learning*, 20(3): 273–297, 1995.

[8] K. Crammer and Y. Singer. A new family of online algorithms for category ranking. In *Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 151–158. ACM, 2002.

[9] S. C. Deerwester, S. T. Dumais, T. K. Landauer, G. W. Furnas, and R. A. Harshman. Indexing by latent semantic analysis. *JAsIs*, 41(6):391–407, 1990.

[10] A. Elisseeff and J. Weston. A kernel method for multi-labelled classification. In *Advances in neural information processing systems*, pages 681–687, 2001.

[11] N. Fuhr, S. Hartmann, G. Lustig, M. Schwantner, K. Tzeras, and G. Knorz. *AIR, X: a rule based multistage indexing system for large subject fields*. Citeseer, 1991.

[12] N. Ghamrawi and A. McCallum. Collective multi-label classification. In *Proceedings of the 14th ACM international conference on Information and knowledge management*, pages 195–200. ACM, 2005.

[13] E. Grefenstette, G. Dinu, Y.-Z. Zhang, M. Sadrzadeh, and M. Baroni. Multi-step regression learning for compositional distributional semantics. *arXiv preprint arXiv:1301.6939*, 2013.

[14] N. Gupta and S. Singh. Collectively embedding multi-relational data for predicting user preferences. *arXiv preprint arXiv:1504.06165*, 2015.

[15] M. U. Gutmann and A. Hyvärinen. Noise-contrastive estimation of unnormalized statistical models, with applications to natural image statistics. *The Journal of Machine Learning Research*, 13(1):307–361, 2012.

[16] D. W. Hosmer and S. Lemeshow. Applied logistic regression. 1989. *New York: Johns Wiley & Sons*, 1989.

[17] T. Joachims. *Text categorization with support vector machines: Learning with many relevant features*. Springer, 1998.

[18] O. Levy and Y. Goldberg. Dependencybased word embeddings. *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*, 2: 302–308, 2014.

[19] D. D. Lewis. *Representation and learning in information retrieval*. PhD thesis, University of Massachusetts, 1992.

[20] D. D. Lewis. Text representation for intelligent text retrieval: a classification-oriented view. *Text-based intelligent systems: current research and practice in information extraction and retrieval*, pages 179–197, 1992.

[21] D. D. Lewis and M. Ringuette. A comparison of two learning algorithms for text categorization. In *Third annual symposium on document analysis and information retrieval*, volume 33, pages 81–93, 1994.

[22] D. D. Lewis, R. E. Schapire, J. P. Callan, and R. Papka. Training algorithms for linear text classifiers. In *Proceedings of the 19th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 298–306. ACM, 1996.

[23] Y. Liu, R. Jin, and L. Yang. Semi-supervised multi-label learning by constrained non-negative matrix factorization. In *Proceedings of the national conference on artificial intelligence*, volume 21, page 421. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2006.

[24] A. McCallum. Multi-label text classification with a mixture model trained by em. 1999.

[25] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[26] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems*, pages 3111–3119, 2013.

[27] T. Mikolov, W.-t. Yih, and G. Zweig. Linguistic regularities in continuous space word representations. In *HLT-NAACL*, pages 746–751, 2013.

[28] J. Mitchell and M. Lapata. Composition in distributional models of semantics. *Cognitive science*, 34(8):1388–1429, 2010.

[29] A. Mnih and K. Kavukcuoglu. Learning word embeddings efficiently with noise-contrastive estimation. In *Advances in Neural Information Processing Systems*, pages 2265–2273, 2013.

[30] A. Mnih and Y. W. Teh. A fast and simple algorithm for training neural probabilistic language models. *arXiv preprint arXiv:1206.6426*, 2012.

[31] F. Morin and Y. Bengio. Hierarchical probabilistic neural network language model. In *Proceedings of the international workshop on artificial intelligence and statistics*, pages 246–252. Citeseer, 2005.

[32] I. Moulinier, G. Raskinis, and J. Ganascia. Text categorization: a symbolic approach. In *proceedings of the fifth annual symposium on document analysis and information retrieval*, pages 87–99, 1996.

[33] H. T. Ng, W. B. Goh, and K. L. Low. Feature selection, perceptron learning, and a usability case study for text categorization. In *ACM SIGIR Forum*, volume 31, pages 67–73. ACM, 1997.

[34] K. Nigam, J. Lafferty, and A. McCallum. Using maximum entropy for text classification. In *IJCAI-99 workshop on machine learning for information filtering*, volume 1, pages 61–67, 1999.

[35] K. Nigam, A. K. McCallum, S. Thrun, and T. Mitchell. Text classification from labeled and unlabeled documents using em. *Machine learning*, 39(2-3):103–134, 2000.

[36] J. R. Quinlan. Learning efficient classification procedures and their application to chess end games. In *Machine learning*, pages 463–482. Springer, 1983.

[37] J. R. Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986.

[38] T. N. Rubin, A. Chambers, P. Smyth, and M. Steyvers. Statistical topic models for multi-label document classification. *Machine Learning*, 88(1-2):157–208, 2012.

[39] G. Salton and C. Buckley. Term-weighting approaches in automatic text retrieval. *Information processing & management*, 24(5):513–523, 1988.

[40] G. Salton and C.-S. Yang. On the specification of term values in automatic indexing. *Journal of documentation*, 29(4):351–372, 1973.

[41] R. Socher, A. Perelygin, J. Y. Wu, J. Chuang, C. D. Manning, A. Y. Ng, and C. Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the conference on empirical methods in natural language processing (EMNLP)*, volume 1631, page 1642. Citeseer, 2013.

[42] R. M. Tong and L. A. Appelbaum. Machine learning for knowledge-based document routing (a report on the trec-2 experiment). *NIST SPECIAL PUBLICATION SP*, pages 253–253, 1994.

[43] J. Turian, L. Ratinov, and Y. Bengio. Word representations: a simple and general method for semi-supervised learning. In *Proceedings of the 48th annual meeting of the association for computational linguistics*, pages 384–394. Association for Computational Linguistics, 2010.

[44] N. Ueda and K. Saito. Parametric mixture models for multi-labeled text. In *Advances in neural information processing systems*, pages 721–728, 2002.

[45] V. Vapnik. *The nature of statistical learning theory.* Springer Science & Business Media, 2000.

[46] E. Wiener, J. O. Pedersen, A. S. Weigend, et al. A neural network approach to topic spotting. In *Proceedings of SDAIR-95, 4th annual symposium on document analysis and information retrieval*, pages 317–332. Citeseer, 1995.

[47] Y. Yang. Expert network: Effective and efficient learning from human decisions in text categorization and retrieval. In *Proceedings of the 17th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 13–22. Springer-Verlag New York, Inc., 1994.

[48] Y. Yang and C. G. Chute. A linear least squares fit mapping method for information retrieval from natural language texts. In *Proceedings of the 14th conference on Computational linguistics-Volume 2*, pages 447–453. Association for Computational Linguistics, 1992.

[49] A. Yessenalina and C. Cardie. Compositional matrix-space models for sentiment analysis. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 172–182. Association for Computational Linguistics, 2011.

[50] F. M. Zanzotto, I. Korkontzelos, F. Fallucchi, and S. Manandhar. Estimating linear models for compositional distributional semantics. In *Proceedings of the 23rd International Conference on Computational Linguistics*, pages 1263–1271. Association for Computational Linguistics, 2010.

[51] M.-L. Zhang and Z.-H. Zhou. A k-nearest neighbor based algorithm for multi-label classification. In *Granular Computing, 2005 IEEE International Conference on*, volume 2, pages 718–721. IEEE, 2005.

[52] M.-L. Zhang and Z.-H. Zhou. Ml-knn: A lazy learning approach to multi-label learning. *Pattern recognition*, 40(7):2038–2048, 2007.

[53] W. Y. Zou, R. Socher, D. M. Cer, and C. D. Manning. Bilingual word embeddings for phrase-based machine translation. In *EMNLP*, pages 1393–1398, 2013.