Learning Distributed Document Representations for Multi-Label Document Categorization

A Thesis Submitted
in Partial Fulfilment of the Requirements
for the Degree of

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

by
Nitish Gupta
Roll No. : 10327461

under the guidance of
Prof. Harish Karnick
Prof. Rajesh M. Hegde

Department of Electrical Engineering
Indian Institute of Technology Kanpur
April, 2015

CERTIFICATE

It is certified that the work contained in this thesis entitled "*Merging Word Senses*", by *Sumit Bhagwani(Roll No. Y8127515)*, has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

(Prof. Harish Karnick)
Department of Computer Science and Engineering,
Indian Institute of Technology Kanpur
Kanpur-208016

June, 2013

Abstract

Abstract goes here

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

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

Chapter 2

Related Work

The task of text classification, 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 text classification 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 [Lewis, 1992a]. Moreover, [Quinlan, 1983] 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 trans-

formed 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 for text classification.

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)* [Salton and Yang, 1973].

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 described below :

1. **One Hot Representation** : This is the most trivial representation, where each document is represented by a vector that is size of the vocabulary. Each element in the vector is either a 0 or a 1 to denote the absence or presence of a specific term in the document.

2. **Term Frequency (tf))** : The term frequency representation 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.

3. **Inverse Document Frequency (idf)** : 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 [1988] 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

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

4. **Term Frequency Inverse Document Frequency (tf-idf)** : 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 the *tf* and the *idf* (*tf*$\times$*idf*). As we will see in the results section, the *tf-idf* weighed bag-of-words document representation gives one of the best accuracies in the multi-label text classification 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,

1. **Information Gain** : Information Gain is widely used as a term-goodness criterion in the field of machine learning, mainly in decision trees [Quinlan, 1986] and also in text classification [Lewis and Ringuette, 1994], [Moulinier et al., 1996]. 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

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

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,

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

3. 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 above. 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.

4. **Latent Semantic Indexing (LSI)** : LSI first introduced by Deerwester et al. [1990], 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 $X = \left[x_1, x_2, \ldots, x_{|D|}\right] \in \mathbb{R}^{|V|}$, 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 text classification has seen growing number of statistical learning methods being applied to it. Over the years, various learning algorithms like, Regression models ([Cooper et al., 1994], [Fuhr et al., 1991]), Conditional Random Field ([Ghamrawi and McCallum, 2005]), Nearest Neighbor techniques ([Yang, 1994], [Zhang and Zhou, 2005], [Zhang and Zhou, 2007]), Bayesian classifier and topic modeling ([Lewis and Ringuette, 1994], [McCallum, 1999], [Nigam et al., 2000], [Rubin et al., 2012], [Nigam et al., 1999], [Ueda and Saito, 2002]), SVM ([Joachims, 1998], [Elisseeff and Weston, 2001]), Neural Networks ([Wiener et al., 1995], [Ng et al.,

1997]), Decision Trees ([Tong and Appelbaum, 1994]), Online learning algorithms ([Lewis et al., 1996], [Crammer and Singer, 2002]), Non-negative Matrix Factorization ([Liu et al., 2006]) etc. have been used or 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.

Multi-label classification problems can be also be classified into classification-based and ranking-based approaches, where the former assigns each test instance a $|L|$-sized label vector of ones and zeros indicating the presence and absence of labels. In the case of a ranking-based approach, the ranking system outputs the list of labels arranged in the increasing order of a ranking score which is then thresholded at an optimum and the top labels are considered appropriate label assignments for test instances.

Below we describe some of the famous learning algorithms for multi-label text classification,

2.2.1 Text Classification with Multiple Binary Classifiers

The most common approach of multi-label text classification 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'*. Below we describe some of the algorithms, in the context of multi-label text classification, that learn multiple independent binary classifiers.

1. **Logistic Regression (LR)** : Introduced by [Hosmer and Lemeshow, 1989], 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 classification 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 text classification, the details for which are given in Sec 4.1.

2. **Support Vector Machines (SVM)** : Support Vector Machines ([Cortes and Vapnik, 1995], [Vapnik, 2000]) 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 text classification, training data is treated separately for each category and maximum margin separating hyperplanes are found for each category independently [Joachims, 1998].

   Elisseeff and Weston [2001] 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.

3. **Neural Networks (NNet)** : Classification-based, Neural Network approaches to multi-label text classification were mainly studied by Wiener et al. [1995], developed at Xerox PARC and called NNet.PARC and Ng et al. [1997], 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.

4. **Naive Bayes (NB)** : Naive-Bayes as studied in Lewis [1992b] and Lewis and Ringuette [1994], 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. in the sentence,

  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

  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.

- Apart from inefficiencies induced by ignoring category correlations, learning independent classifiers poses other drawbacks, such as, in case of millions of

labels, learning millions of high-dimensional classifiers is a computationally expensive. Secondly, 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 Text Classification with Single Joint Classifier

To overcome the difficulties and drawback of learning multiple binary classifiers, researchers have since developed learning algorithms that jointly classify each document into categories it belongs to. Outputs of such algorithms are $|L|$-dimensional 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 for multi-label text classification that learn a single classifier for assigning all relevant labels to a document jointly.

1. **k-Nearest Neighbor (kNN)** : k-nearest neighbor classification is one of the most effective lazy learning approaches to classification. Given an arbitrary text document 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 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 [2007] is based on the k-NN and the maximum a posteriori(MAP) principle. Their approach is, given a test instance, to first identify its k-nearest neighbors and then based on the statistical information gained from the label sets of the neighboring instances, use 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.

2. **Linear Least Squares Fit (LLSF)** : LLSF[Yang and Chute, 1992] 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.

3. **Probabilistic Models** : Generative probabilistic models described in McCallum [1999], Nigam et al. [1999], Ueda and Saito [2002] etc. argue that the words in a document belonging to a multi-category class 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 [1999] and Ueda and Saito [2002] define a multinomial distribution $\boldsymbol{\theta}_l = \{\theta_{l1}, \theta_{l2}, \ldots, \theta_{l|V|}\}$ over the vocabulary for each label, and 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 $\boldsymbol{\phi}(\boldsymbol{y}) = \{\phi_1(\boldsymbol{y}), \phi_2(\boldsymbol{y}), \ldots, \phi_2(\boldsymbol{y})\}$ 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 in [Ueda and Saito, 2002] and by employing the Expectation-Maximization algorithm in [McCallum, 1999].

Chapter 3

Distributed Document Embeddings

In this chapter we describe the concept of distributed word and document embeddings 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 that learn distributed representations for words in a fully unsupervised manner and finally describe in detail our proposed model for learning distributed embeddings for documents that can be used for multi-label text classification.

3.1 Motivation

TODO: Get in tune to document representations. Say words and documents suffer in the same manner with one-hot or bow representations. Express problems in docs with changing words. Give example of sentence

TODO: Can be tackled with distributed repr. Similarity measures as simple as cos-distance can be introduced in documents. Lets model joint distributions of words with continuous distributions. Words have distributed representaions but not docs.

Words are regarded as atomic symbols in most rule-based and statistical natu-

ral language processing(NLP) tasks and hence need the 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 more robust, low-dimensional, non-sparse vector representations for words that capture the semantic similarity between them, 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 similarity between 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 $50 - 300$. 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 ([Bengio et al., 2003a], [Mnih and Kavukcuoglu, 2013], [Mikolov et al., 2013b], [Collobert et al., 2011], [Bottou, 2014], [Turian et al., 2010], [Levy and Goldberg, 2014]) but differ in their training objectives.

Below we describe in detail two models to show how models with very different learning objectives and architecture can lead to learning high-quality word vectors.

3.2.1 Neural Probabilistic Language Model

Neural Probabilistic Language Model (NPLM), introduced by Bengio et al. [2003a], 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

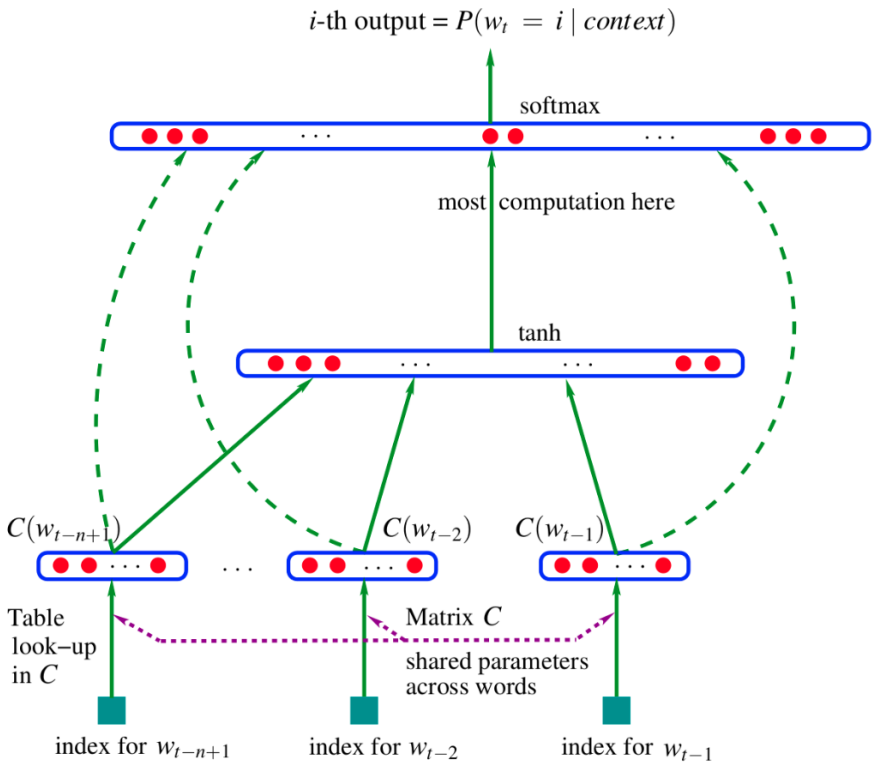


Figure 3.1: Bengio's Neural Network Architecture for Neural Probabilistic Language Model

the t^{th} word in the sequence is w_i.

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

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 summing 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. [2013a] as opposed to the non-linear NPLM model to bring down the training time complexity without sacrificing with the quality of the word vectors. **TODO:** Also these models are based on the Distributional Hypothesis The models bring down the complexity of learning vectors 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. [2013a] (word2vec) are Continuous Bag-of-Words and Continuous Skip-Gram model, described below.

3.2.2.1 Continuous Bag-of-Words

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 NNLM that uses the concatenation of input vectors for the projection layer. The training criteria in this model is to to classify 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 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 $P(w_t|w_{t-k}, \ldots, w_{t+k})$,

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

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

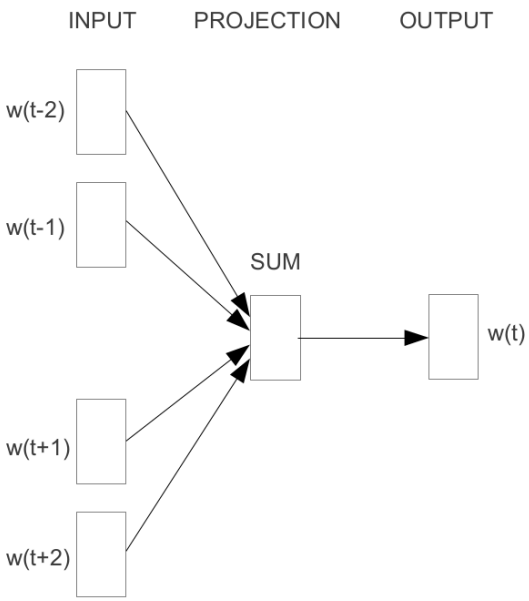


Figure 3.2: Continuous Bag-of-Words Model (CBOW) **TODO:** Add ref?

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

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 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* [Morin and

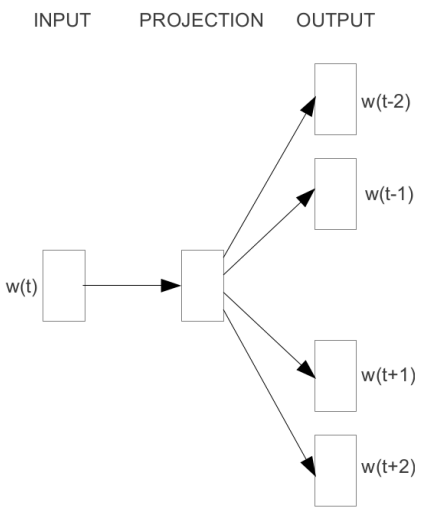


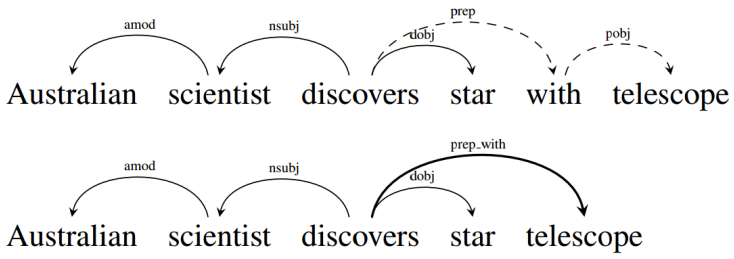
Figure 3.3: Continuous Skip-gram Model **TODO:** Add ref?

Bengio, 2005] 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 linguistic regularities and pattern [Mikolov et al., 2013c] 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 [2014] 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 is. 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 extracting the contexts, their model uses the neural network



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: Dependency-based context extraction example **TODO:** Add ref?

architecture and the training 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. In the task of *topical similarity* estimation, the vectors from the skip-gram model performed better as they encode semantic similarity between words because of the bag-of-words context used during training.

3.3 Document Embeddings

In the previous section we saw how distributed word embeddings that encode semantic similarity can be learned from text. Though these semantic word spaces are very useful for a lot of 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 [Mitchell and Lapata [2010], Zanzotto et al. [2010], Yessenalina and Cardie [2011], Grefenstette et al. [2013], Mikolov et al. [2013b]] 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. [2013] 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 vector representations for documents and words in the corpus such that,

1. The learned vectors encode semantic and topical content of the documents and words.

2. Semantically similar documents/words have similar vector representations.

To learn vectors that satisfy 1. and 2. above, 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 set of 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 in the context correctly 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 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 w_t using the surrounding context words $(w_{t-c}, \ldots, w_{t-1}, w_{t+1}, \ldots, w_{t+c})$, which we now denote as *context* as a shorthand, and the information about the document in terms of their distributed vector representations. Therefore, our training objective is to maximize the probability $p(w_t|d_i, \mathit{context})$ of correctly predicting the middle word using the information about the surrounding words and the document the sequence occurs in.

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

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

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

3. Learn the word and document vectors simultaneously with the parameters of the function 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.

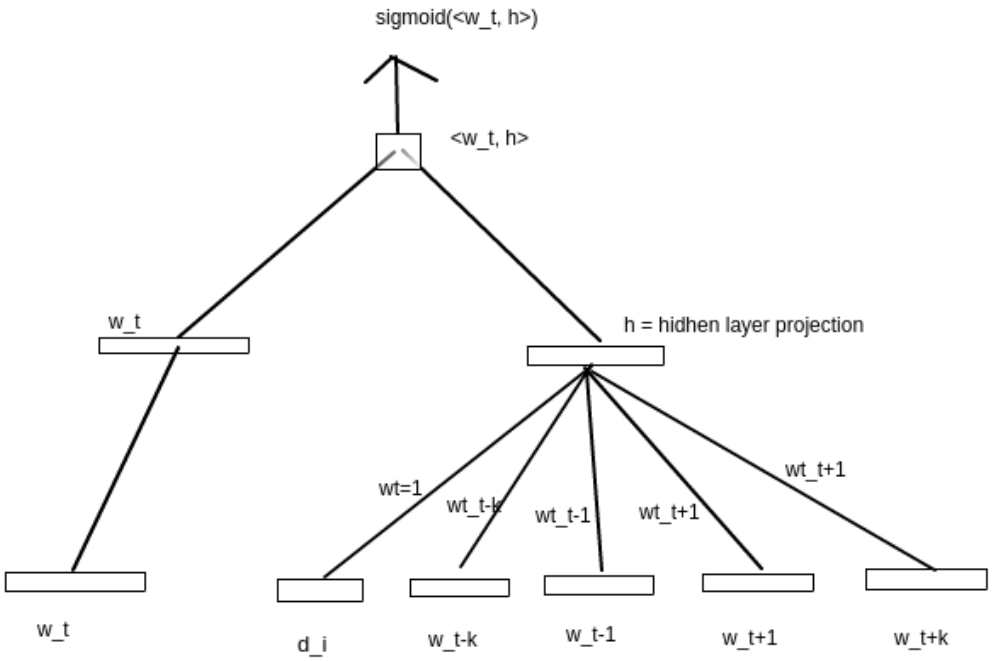


Figure 3.5: GloDETC : Neural Network Architecture **TODO:** Change figure

3.3.2.1 Projection Layer (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 as we expected the document to have equal contribution to all sequences. This also gave the best

results. We also (unsuccessfully) experimented by taking matrix weights instead of scalar weights(λ_i) to learn better syntactic qualities of the language.

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

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,

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

3.3.2.3 Training 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, $t = \{d_i^{(m)}, w_{t-c}^{(m)}, \ldots, w_{t+c}^{(m)}\}$ 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,

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

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

To learn the optimize the above training objective, 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. Also note that we add the gradient to $\theta_i^{(x)}$ because we wish to maximize the training objective. Updating the parameters for sufficient number of iterations should 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 [Morin and Bengio, 2005] and use of importance sampling to approximate the likelihood gradient [Bengio et al., 2003b], [Bengio and Senecal, 2008] have been proposed. Using hierarchical softmax reduces the training time from linear to logarithmic in vocabulary size but is considerable 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) [Gutmann and Hyvärinen, 2012] is

method for fitting unnormalized probabilities by reducing the problem of *density estimation* to *probabilistic binary classification*. It has also been adapted to NPLM (Sec. 3.2.1) [Mnih and Teh, 2012] and learning word embeddings Mnih and Kavukcuoglu [2013] and shows significant improvements in training time with no considerable degradation in the quality of word vectors learned.

The basic idea of NCE is to train 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 now in $\mathcal{T}$ is $M + nM$. 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,

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

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,

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

From Eq. 3.16 and Eq. 3.17 we get,

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

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

3.3.2.5 New Training Objective

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,

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

3.3.2.6 Parameter Estimation

To learn the optimum parameters $\Theta = (\mathrm{D}, \mathrm{W}, \Lambda)$ we would 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 would 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,

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

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

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

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,

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

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,

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

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

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

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

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

1. **Document Vector** :

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

2. **Middle Word** :

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

3. **Context Word** :

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

4. **Neural Network Weight** :

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

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 document vector, word vectors and the neural network weights are hence given by,

1. **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. **Middle Word Vector** :

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

3. **Context Word Vectors** :

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

4. **Neural Network Weights** :

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

To learn the vectors D, W and weights Λ we initialize them to small random vectors and scalars respectively and using Eqs. 3.35, 3.36, 3.37 and 3.38 we iterate through the training data, making the appropriate updates for a fixed number of epochs that is learned using the development data. For each training sequence we make one update to the document vector, $2c$ updates for the 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.

3.3.2.7 Hyper-parameters

Our model, like any other, has hyper-parameters that need to be tuned for optimum model performance and learning high quality document and word vectors and the model parameters for achieving the the best accuracies in the task. 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

Algorithm 1 Learning Document and Word Vector Representations

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

embedding dimensionality needs to be big 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 plays an important role in the vectors that are learned. While a smaller window could result in the negligence of important/similar 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, the number of negative samples introduced in the training data per positive example plays an important role in deciding the trade-off between learning better word density distribution with larger n while smaller n leads to lesser training times. Hence, the number of negative samples introduced needs to be tuned using the development data.

4. **Number of Epochs** : The number of times we need to loop through the training data to learn the vectors needs to be optimized to prevent overfitting with large epochs while at the same time learn high quality representations.

5. **Learning Rate (γ)** : The learning rate used in the SGD algorithm plays an important role whose optimum value is dependent on the dataset and the objective function. While a smaller learning rate could harm the training time and convergence, a larger learning rate may lead to a 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 value may not penalize complexity by enough, larger constants may inhibit the growing of parameters with a negative effect which required careful selection of the regularization constant dependent on the dataset.

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

Chapter 4

Multi-Label Text Categorization

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

4.1 Logistic Regression for Multi-label Document Categorization

Introduced by [Hosmer and Lemeshow, 1989], Logistic Regression (LR) is a probabilistic binary classification regression model that, given labeled binary 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 would refer to our algorithm as logistic regression.

In the sections below we describe the training data required for the task of multi-label document classification, the logistic regression model as modified for the task and also its similarity to relational learning.

4.1.1 Training Data

The training data $\mathcal{T}$ is composed of a set of documents $\mathbf{D}$, set of categories $\mathbf{C}$ and data about in what categories do each of the documents belong to.

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 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 a document necessarily does not belong to a particular category. 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 document representations in the form of either bag-of-words representations or distributed document embeddings as learnt in Sec. 3.3. Therefore for every document $d_i \in \mathbf{D}$ indexed by i, we have a vector representation $\mathrm{v}_i^D \in \mathbb{R}^k$ of the document.

4.1.2 Logistic Regression Model

For multi-label document categorization we extend the standard logistic regression (which is a binary classification algorithm) to,

1. Train from multi-labeled document-category data data

2. Given a document-category pair $\{d_i, c_j\}$, estimate the probability of the document d_i belonging to the category c_j.

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 $\mathrm{v}_{c_i}^C \in \mathbb{R}^k$ 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,

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

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 $\mathrm{v}_{c_j}^C$. 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 Training Objective

As explained in Sec. 4.1.1, the training data $\mathcal{T}$, is composed of T tuples of the form $\{d_i^{(m)}, c_j^{(m)}, y^{(m)}\}$. 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 $\{d_i^{(m)}, c_j^{(m)}, y^{(m)}\}$, we denote the prediction that

whether the document $d_i^{(m)}$ belongs the category $c_j^{(m)}$ by y_m. Therefore, if we estimate $d_i^{(m)}$ belongs $c_j^{(m)}$, $y_m = 1$ otherwise $y_m = 0$. 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 L2 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.

As explained in 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, *epochs*
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**
    $\mathrm{v}_{c_j}^C \leftarrow \mathrm{v}_{c_j}^C + \gamma \left[(y - \sigma(\mathrm{v}_{d_i}^D \cdot \mathrm{v}_{c_j}^C)) \mathrm{v}_{d_i}^D - \beta \mathrm{v}_{c_j}^C \right]$
  end for
end while
return C

4.2 Similarity to Relational Learning

The 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 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 $r(i, j) \in \{0, 1\}$ of the matrix denotes the presence/absence of link between the i-th row and the j-th column entity. Employing 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 embeddings D using our model described in Sec. 3.3.2, we only learn the category embeddings matrix C instead of learning embeddings 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 the major advantages of the algorithm.

1. Since we learn distributed embeddings for the categories in the same k-dimensional space as the documents and words it enables us to estimate the similarity within entities and between unrelated entities. It is as simple as taking a dot-product of the corresponding embeddings. For example, we can estimate the similarity between two or more categories or between categories and words that wouldn't 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 as done by 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 [2015], 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. **TODO:** Find missing categories??

Chapter 5

Datasets and Evaluations

TODO: Section 3 : Results - Document Categorization Section 4 : Results - Imputing Missing Categories. Only done on Wikipedia datasets Section 5 : Estimating Similarity between cats, docs, words

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 for the task. In the sections following that we present the results for assigning categories to new documents and also recommending categories for documents which we already have some 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 and contains a total 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 large number of documents and categories present, the dataset is very sparse that makes 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 Roch 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 in the following manner. **TODO:** rephraseFor each of the top-level category we compiled a list of all its child categories till a 3-level depth. Then, to create the document-category dataset, we considered all the documents and the categories assigned to them, that belonged to these categories. The four top-level categories we consider, hence, the datasets that we extract from Wikipedia are from the *Physics*, *Biology*, *Mathematics* and *Sports* categories. The number of documents, categories, words and other statistics of the extracted datasets are given in Table 5.2.

[1] www.wikipedia.org

	|D|	|C|	|V|	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 criteria and techniques and also the baselines that 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 keep the capitalization of the words as it is because capitalization sometimes encodes a lot of semantic content that we do not wish to lose. Eg. *Apple* in most cases is used to refer *Apple Inc.* (Company) rather than the fruit. Similarly, *apple* is most likely to refer to the fruit. Models that distinguish between the two forms will be able to learn better embeddings.

Learning Document Embeddings : 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 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 [Mnih and Kavukcuoglu, 2013], 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 embeddings. We stop training based on the convergence reached on the development data. The value of the hyper-parameters 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 predictions that whether the document should be assigned a particular category, we need to threshold the estimated probability, threshold 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 to evaluate our model i.e. the harmonic mean of the precision and recall values. We compute the F1 score in the micro-averaged fashion i.e. we combine prediction values for all the categories together for a particular dataset and

then compute the precision and recall values. F1 score is a much more preferred measure compared to hamming loss or accuracy in the presence of imbalanced label distribution.

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

1. **Bag-of-Words (BOW)** : The most famous document representation that has produced state-of-the-art results is the bag-of-words model with tf-idf weighting. We compare our document representations against the bag-of-words representations. We call this model the **BOW** model.

2. **Latent Semantic Indexing (LSI)** : 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 and compare our representations against them. 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 representations learned using our method perform better than averaged 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. Note that 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.

Also note that employing PMF for predicting categories from new documents is useless as they do not contain any category history in the training data and hence the document vectors would not be learned. PMF in such case would act as a trivial strawman. PMF can be used to predict categories for documents that already have category history in training data.

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, details for which are given in the next section.

5.3 Results

In this section we present our model's accuracy in categorizing new documents by learning distributed representations for the documents in the corpus and using historical data to learn category embeddings to predict categories for new documents. We compare our model's performance 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 categories for known documents in case the data has missing values.

5.3.1 Document Categorization

As we see in **TODO:** some ref, the task of categorizing documents with no prior categorization is of immense importance. In this section we present our model's efficacy in assigning categories to documents with no prior categorization.

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

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

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 embeddings 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-embeddings) 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 shows 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.

Table 5.7 compares our model's accuracy against different document represen-

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

tations and other learning algorithms that use bag-of-words representations. We find that document representations learned using our model performs the best. Our representations improves the F1 score of 84.1%, achieved when using bag-of-words representation, by 9%. Improvements of 1%-6.6% on the F1 score are found against models such as SVM, CRF, LSI, MFoM and representations learned using tf-idf weighted average of word vectors. 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.

5.3.1.2 Physics - Wikipedia

The performance of our model on the development data for different values of different hyper-parameters is given in Table 5.8, 5.9, 5.10 and 5.11. From Table 5.8 and 5.9 we find that our model performs the best for $k = 100$ when trained for 150 epochs. 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.

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: **Document Categorization on Reuters-21578(*ModApte*) dataset:** Precision/Recall/F1 on document categorization of our model compared against different document representations and complex learning algorithms on the Reuters-21578 dataset.

	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, Bag-of-Words (**BOW**) trails our by 4.18%. 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 model only achieves an accuracy of **TODO:** add no weight in terms of the F1 score. Fig. 5.2 also shows that our model performs the best while **BOW**

	Hyper-parameters : $k = 100$, $c = 2$, $n = 10$														
Tuning	$epochs = 20$			$epochs = 50$			$epochs = 100$			$epochs = 150$			$epochs = 200$		
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

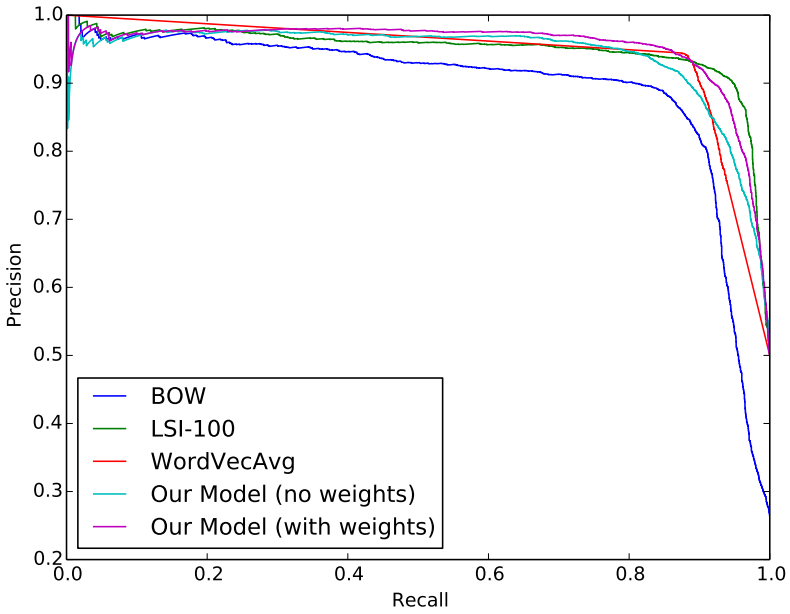


Figure 5.1: Precision/Recall for Document Categorization on Reuters-21578 (*ModApte*) dataset

	Hyper-parameters : $k = 100$, $epochs = 150$, $n = 10$								
Tuning	$c = 2$			$c = 3$			$c = 4$		
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

	Hyper-parameters : $k = 100$, $epochs = 150$, $c = 3$								
Tuning	$n = 5$			$n = 10$			$n = 15$		
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 is the second best performing model.

5.3.1.3 Biology - Wikipedia

The performance of our model on the development data for different values of different hyper-parameters is shown in Table 5.13, 5.14, 5.15 and 5.16. From Table 5.13 shows that default embedding size of $k = 100$ performs the best and 5.14 shows

Physics (Wikipedia)	P	R	**F1**
BOW	75.5	77.5	76.5
LSI-100	57.4	80.5	67.0
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: **Document Categorization on Physics(Wikipedia) dataset:** Precision/Recall/F1 of our model for document categorization compared against different document representations on the Physics(Wikipedia) dataset.

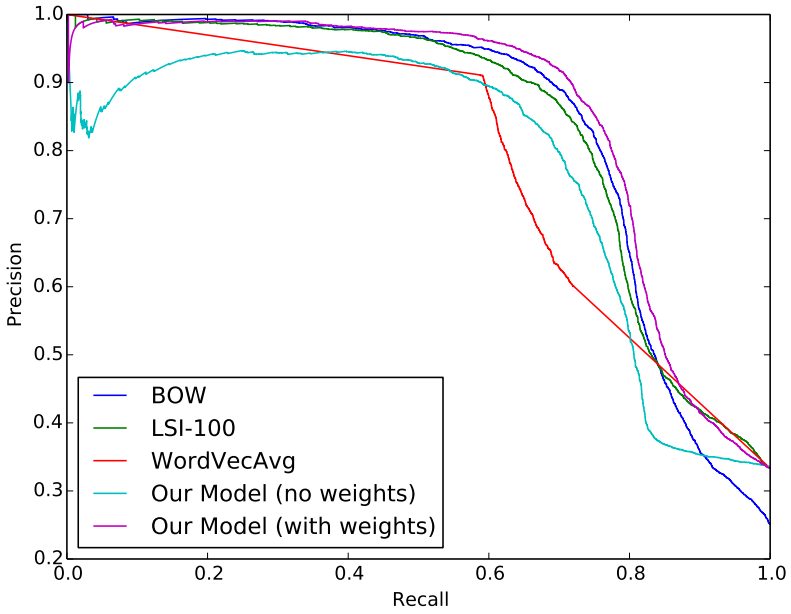


Figure 5.2: Precision/Recall for Document Categorization on Physics(Wikipedia) dataset

that relatively low training epochs ($epochs = 50$) gives the best results. Table 5.15 shows that larger context window ($c = 3$) and default number of negative samples ($n = 10$) per positive sample 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

	Hyper-parameters : $k = 100$, $c = 2$, $n = 10$														
Tuning	*epochs* = 20			*epochs* = 50			*epochs* = 100			*epochs* = 150			*epochs* = 200		
Biology (Development)	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

	Hyper-parameters : $k = 100$, *epochs* = 50, $n = 10$								
Tuning	$c = 2$			$c = 3$			$c = 4$		
Biology (Development)	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

	Hyper-parameters : $k = 100$, *epochs* = 50, $c = 3$								
Tuning	$n = 5$			$n = 10$			$n = 15$		
Biology (Development)	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

Table 5.17 shows that the distributed representations learned by our model outperform other baseline representations. Our model achieves a F1 score of 67.8% while the second best model, again the Bag-of-Words (**BOW**) trails ours by 2.41%. We also find that document representations learned by averaging word vectors perform worse by 9.1% and the model accuracy suffers by 4.95% when word ordering 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 representations learned using our model.

5.3.1.4 Mathematics - Wikipedia

The performance of our model on the development data for different values of different hyper-parameters is shown in Table 5.18, 5.19, 5.20 and 5.21. Table 5.18

Biology (Wikipedia)	P	R	**F1**
BOW	67.0	65.3	66.2
LSI-100	51.8	69.6	59.4
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: **Document Categorization on Biology(Wikipedia) dataset:** Precision/Recall/F1 of our model for document categorization compared against different document representations on the Biology(Wikipedia) dataset.

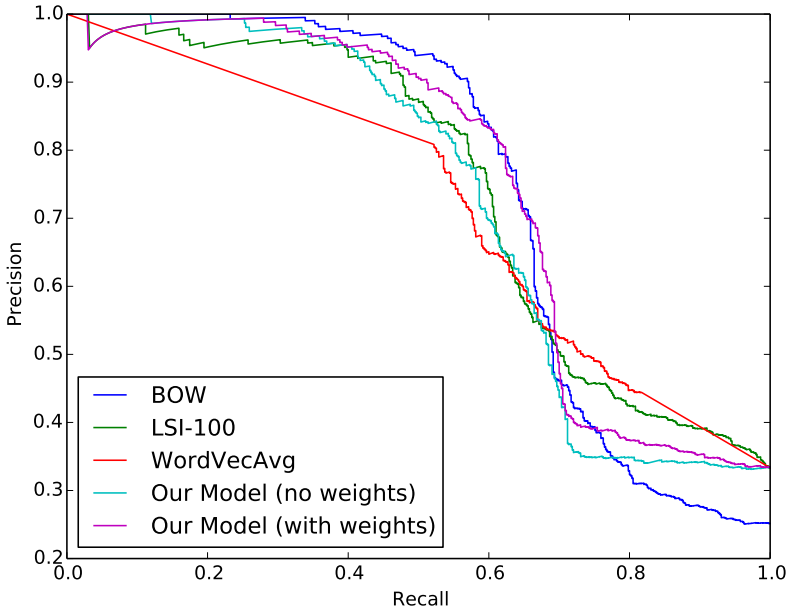


Figure 5.3: Precision/Recall for Document Categorization on Biology(Wikipedia) dataset

and 5.19 show that low embedding size of $k = 50$ and moderate training epochs ($epochs = 100$) give the best model performance. Table 5.20 shows that small context window ($c = 3$) and relatively larger number of negative samples ($n = 15$) per positive sample achieves the best performance.

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, Bag-of-Words (**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

Tuning	**Hyper-parameters** : $epochs = 50$, $c = 2$, $n = 10$								
	$k = 50$			$k = 100$			$k = 150$		
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

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

again find that modeling syntactic qualities of the language is important as weighting word vectors to represent the context gives the best performance. Fig. 5.4 shows the precision/recall curves for our model and other baseline methods. It also shows that the **BOW** representations perform the closest representations learned using our model.

Mathematics (Wikipedia)	P	R	F1
BOW	65.6	65.1	65.3
LSI-100	47.0	75.0	57.8
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: **Document Categorization on Mathematics(Wikipedia) dataset:** Precision/Recall/F1 of our model for document categorization compared against different document representations on the Mathematics(Wikipedia) dataset.

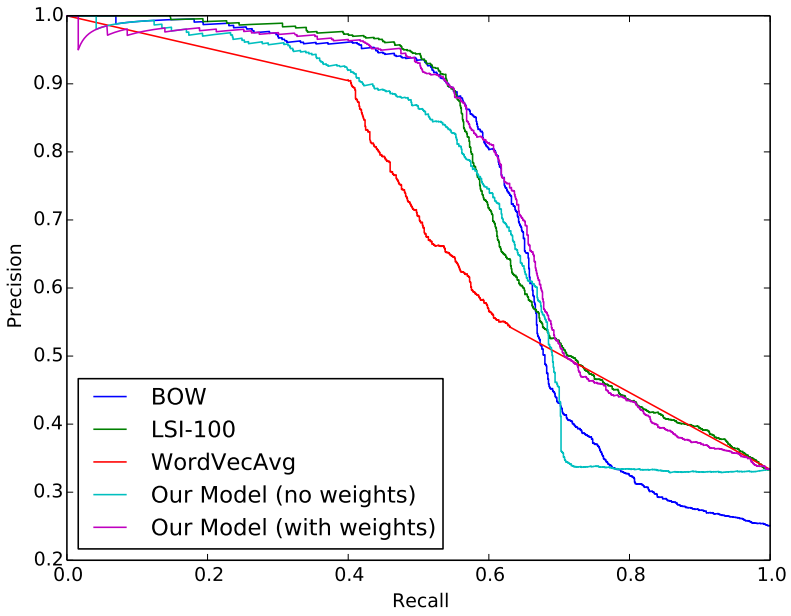


Figure 5.4: Precision/Recall for Document Categorization on Mathematics(Wikipedia) dataset

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 small context window ($c = 3$) and relatively larger number of negative samples ($n = 15$) per positive sample achieves the best performance.

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%

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

Tuning	**Hyper-parameters** : $k = 50$, $c = 2$, $n = 10$														
	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

Tuning	**Hyper-parameters** : $k = 50$, *epochs* = 100, $n = 10$								
	$c = 2$			$c = 3$			$c = 4$		
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

Tuning	**Hyper-parameters** : $k = 50$, *epochs* = 100, $c = 3$								
	$n = 5$			$n = 10$			$n = 15$		
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

while the second best performing representations are learned using Latent Semantic Indexing (**LSI-100**). **LSI-100** achieves a F1 score of 54.1% and trails our model by 6.1%. As found in previous evaluations, document representations using bag-of-words style word vector averaging do not give the best performance and also 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	58.5	48.3	52.9
LSI-100	43.5	71.4	54.1
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: **Document Categorization on Sports(Wikipedia) dataset:** Precision/Recall/F1 of our model for document categorization compared against different document representations on the Sports(Wikipedia) dataset.

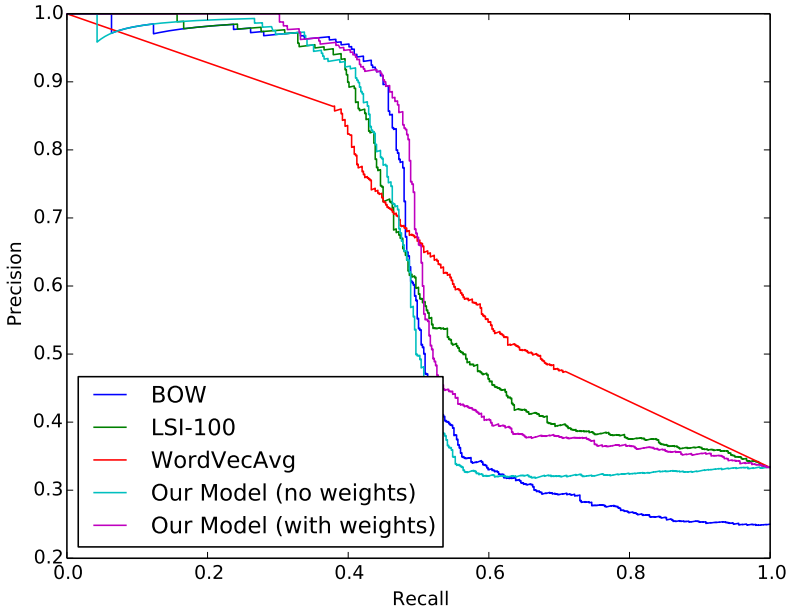


Figure 5.5: Precision/Recall for Document Categorization on Sports(Wikipedia) dataset

5.3.2 Imputing Missing Categories

Most of the real-life databases that contain categorization for documents are incomplete. 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 a 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. Here 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 weighted averaging the word vectors 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 surrounding word vectors to represent the context does not provide competitive results which corroborated with the hypothesis that encoding syntactic information is necessary. Figure 5.6 shows the precision/recall curves for the different document representation models along with the performance of our models.

	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: **Imputing Missing Categories in Wikipedia Datasets:** Performance of different document representation models against our distributed representations model in imputing missing categories in the Wikipedia document-category datasets.

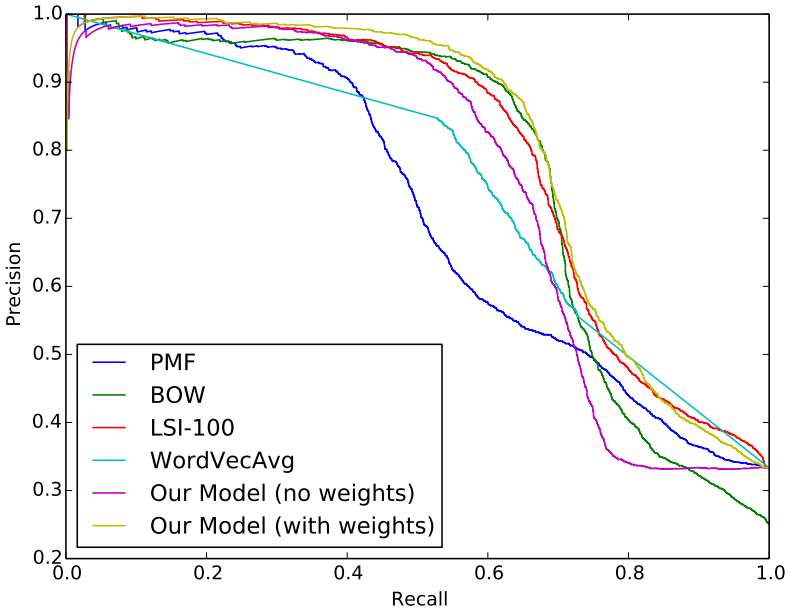


Figure 5.6: Precision/Recall for Imputing Missing Categories on the Wikipedia datasets

Bibliography

Yoshua 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.

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. A neural probabilistic language model. *The Journal of Machine Learning Research*, 3:1137–1155, 2003a.

Yoshua Bengio, Jean-Sébastien Senécal, et al. Quick training of probabilistic neural nets by importance sampling. In *AISTATS Conference*, 2003b.

Léon Bottou. From machine learning to machine reasoning. *Machine learning*, 94 (2):133–149, 2014.

Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural language processing (almost) from scratch. *The Journal of Machine Learning Research*, 12:2493–2537, 2011.

W Cooper, Aitao 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.

Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

Koby Crammer and Yoram 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.

Scott C. Deerwester, Susan T Dumais, Thomas K. Landauer, George W. Furnas, and Richard A. Harshman. Indexing by latent semantic analysis. *JAsIs*, 41(6): 391–407, 1990.

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

Norbert Fuhr, Stephan Hartmann, Gerhard Lustig, Michael Schwantner, Kostas Tzeras, and Gerhard Knorz. *AIR, X: a rule based multistage indexing system for large subject fields*. Citeseer, 1991.

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

Edward Grefenstette, Georgiana Dinu, Yao-Zhong Zhang, Mehrnoosh Sadrzadeh, and Marco Baroni. Multi-step regression learning for compositional distributional semantics. *arXiv preprint arXiv:1301.6939*, 2013.

Nitish Gupta and Sameer Singh. Collectively embedding multi-relational data for predicting user preferences. *arXiv preprint arXiv:1504.06165*, 2015.

Michael U Gutmann and Aapo 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.

David W Hosmer and Stanley Lemeshow. Applied logistic regression. 1989. *New York: Johns Wiley & Sons*, 1989.

Thorsten Joachims. *Text categorization with support vector machines: Learning with many relevant features*. Springer, 1998.

Omer Levy and Yoav Goldberg. Dependencybased word embeddings. *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*, 2: 302–308, 2014.

David 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, 1992a.

David D Lewis and Marc 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.

David D Lewis, Robert E Schapire, James P Callan, and Ron 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.

David Dolan Lewis. *Representation and learning in information retrieval*. PhD thesis, University of Massachusetts, 1992b.

Yi Liu, Rong Jin, and Liu 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.

Andrew McCallum. Multi-label text classification with a mixture model trained by em. 1999.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013a.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems*, pages 3111–3119, 2013b.

Tomas Mikolov, Wen-tau Yih, and Geoffrey Zweig. Linguistic regularities in continuous space word representations. In *HLT-NAACL*, pages 746–751, 2013c.

Jeff Mitchell and Mirella Lapata. Composition in distributional models of semantics. *Cognitive science*, 34(8):1388–1429, 2010.

Andriy Mnih and Koray Kavukcuoglu. Learning word embeddings efficiently with noise-contrastive estimation. In *Advances in Neural Information Processing Systems*, pages 2265–2273, 2013.

Andriy Mnih and Yee Whye Teh. A fast and simple algorithm for training neural probabilistic language models. *arXiv preprint arXiv:1206.6426*, 2012.

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

Isabelle Moulinier, Gailius 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.

Hwee Tou Ng, Wei Boon Goh, and Kok Leong Low. Feature selection, perceptron learning, and a usability case study for text categorization. In *ACM SIGIR Forum*, volume 31, pages 67–73. ACM, 1997.

Kamal Nigam, John Lafferty, and Andrew McCallum. Using maximum entropy for text classification. In *IJCAI-99 workshop on machine learning for information filtering*, volume 1, pages 61–67, 1999.

Kamal Nigam, Andrew Kachites McCallum, Sebastian Thrun, and Tom Mitchell. Text classification from labeled and unlabeled documents using em. *Machine learning*, 39(2-3):103–134, 2000.

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

J. Ross Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986.

Timothy N Rubin, America Chambers, Padhraic Smyth, and Mark Steyvers. Statistical topic models for multi-label document classification. *Machine Learning*, 88(1-2):157–208, 2012.

Gerard Salton and Christopher Buckley. Term-weighting approaches in automatic text retrieval. *Information processing & management*, 24(5):513–523, 1988.

Gerard Salton and Chung-Shu Yang. On the specification of term values in automatic indexing. *Journal of documentation*, 29(4):351–372, 1973.

Richard Socher, Alex Perelygin, Jean Y Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher 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.

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

Joseph Turian, Lev Ratinov, and Yoshua 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.

Naonori Ueda and Kazumi Saito. Parametric mixture models for multi-labeled text. In *Advances in neural information processing systems*, pages 721–728, 2002.

Vladimir Vapnik. *The nature of statistical learning theory*. Springer Science & Business Media, 2000.

Erik Wiener, Jan O Pedersen, Andreas 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.

Yiming 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.

Yiming Yang and Christopher 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.

Ainur Yessenalina and Claire 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.

Fabio Massimo Zanzotto, Ioannis Korkontzelos, Francesca Fallucchi, and Suresh 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.

Min-Ling Zhang and Zhi-Hua 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.

Min-Ling Zhang and Zhi-Hua Zhou. Ml-knn: A lazy learning approach to multi-label learning. *Pattern recognition*, 40(7):2038–2048, 2007.