USING MULTI-TASK LEARNING FOR LARGE-SCALE DOCUMENT CLASSIFICATION

by

Azad Naik
A Thesis
Submitted to the
Graduate Faculty
of
George Mason University
In Partial fulfillment of
The Requirements for the Degree
of
Master of Science
Computer Science

Committee:

Dr. Huzefa Rangwala, Thesis Director

Dr. Daniel Barbará, Committee Member

Dr. Carlotta Domeniconi, Committee Member

Dr. Sanjeev Setia, Chairman, Department of Computer Science

Dr. Kenneth S. Ball, Dean, Volgenau School of Engineering

Date: 03rd May 2013

Spring Semester 2013
George Mason University
Fairfax, VA

Using Multi-Task Learning For Large-Scale Document Classification

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science at George Mason University

By

Azad Naik
Bachelor of Technology
Indian School of Mines, 2009

Director: Dr. Huzefa Rangwala, Professor
Department of Computer Science

Spring Semester 2013
George Mason University
Fairfax, VA

Dedication

I dedicate this thesis to my family members for being there for me throughout my life. I would also like to dedicate this thesis to many of my friends who have supported me throughout the process.

Acknowledgments

I would like to thank my thesis director, Dr. Huzefa Rangwala for the immense help throughout the project, advice given by Dr. Rangwala has been of great help in solving the challenging problem. I am grateful for his countless hours of reading, encouraging and patience throughout the entire process.

I would also like to extend my thanks to the committee member, Dr. Daniel Barbará and Dr. Carlotta Domeniconi for the valuable guidance and precious time in reviewing. I am also thankful to Anveshi Charuvaka for the fruitful discussion and suggestion during the course of my thesis work.

Finally, I would like to thank my parents, friends, relatives, and supporters who have made this happen.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

μAF_1	Micro Average F_1
$AMCC$	Average Matthews Correlation Coefficient
bMH	b-bit Minwise Hashing
$DMOZ$	Directory Mozilla
GPU	Graphics Processing Unit
HD	High Distribution
INA	Individual Neighborhood Approach
kNN	k-Nearest Neighbor
LD	Low Distribution
LSH	Locality Sensitive Hashing
MAF_1	Macro Average F_1
MAP	Macro Average Precision
MAR	Macro Average Recall
MCC	Matthews Correlation Coefficient
MD	Medium Distribution
MTL	Multi-Task Learning
nB	Number of Bins
nP	Number of Permutatiosn
NPA	Neighborhood Pooling Approach
OPH	One Permutation Hashing
sd	Standard Deviation
SE	Standard Error
SSL	Semi-Supervised Learning
STL	Single-Task Learning
TL	Transfer Learning

Abstract

USING MULTI-TASK LEARNING FOR LARGE-SCALE DOCUMENT CLASSIFICATION

Azad Naik

George Mason University, 2013

Thesis Director: Dr. Huzefa Rangwala

Multi-Task Learning (MTL) involves learning of multiple tasks, jointly. It seeks to improve the generalization performance of each task by leveraging the relationships among the different tasks. It is an advanced concept of Single-Task Learning (STL), most widely used in classification. In STL, each task is considered to be independent and learnt independently whereas in MTL, multiple tasks are learnt simultaneously by utilizing task relatedness. The main intuition is that the training signal present in related tasks can help each of the tasks learn better models. It also allows for learning of better models with fewer labeled examples.

In this thesis our focus is on improving the classification performance for a database categorized as a hierarchy and archiving large number of documents. We focus on improving the classification performance of this database (source) by developing a MTL based model. In this model we use an external database to facilitate the classification process for the source database. We have used the logistic regression model for multiple classification tasks and k-nearest neighbor approach for finding the similarities between the classes in two hierarchical databases. The kNN allows us to define task relationships. Experiment on sampled DMOZ dataset has been done to evaluate the performance of MTL with STL,

Semi-Supervised Learning (SSL) and Transfer Learning (TL). We have also used random projections for achieving better runtime performance at a minimal effect on classification accuracy.

Chapter 1: Introduction

Large volume of document data is generated by users every day. With increasing data, it has become an essential need to automatically categorize the related documents into groups/classes. Classifying this document/text data can be useful in several ways. One of the most common use being - information retrieval of related document in response to user query in search engine. Many algorithms [1][2] have been proposed for classifying documents. Recently, the hierarchical structure for document categorization at multiple granularity levels has been explored in [3][4][5]. The main idea behind hierarchical classification models is to capture the parent-child relationship and achieve better classification performance. Although taking hierarchy into account gives better results, the time taken to learn the model is very high.

Multi-Task Learning (MTL) is used to capture the intrinsic relatedness between the task and hence achieve better generalization performance, especially when the number of training examples is less. MTL has been successfully applied in the field of medical informatics [6], structural classification [7], sequence analysis [8], web image and video search [9] and many more applications [10][11][12]. Overall, MTL helps to learn the better predictive models for each of the multiple related task. In text classification, for each of the class labels we define a binary one-versus-rest classification a task. We find the related tasks corresponding to each task using k-nearest neighbor, which is then learned together to find the better parameters for the model corresponding to each task.

1.1 Problem Statement

Automatically categorizing document data into relevant group, accurately and fastly, is a challenging problem. In this thesis, our task is to develop the best model(s) which can

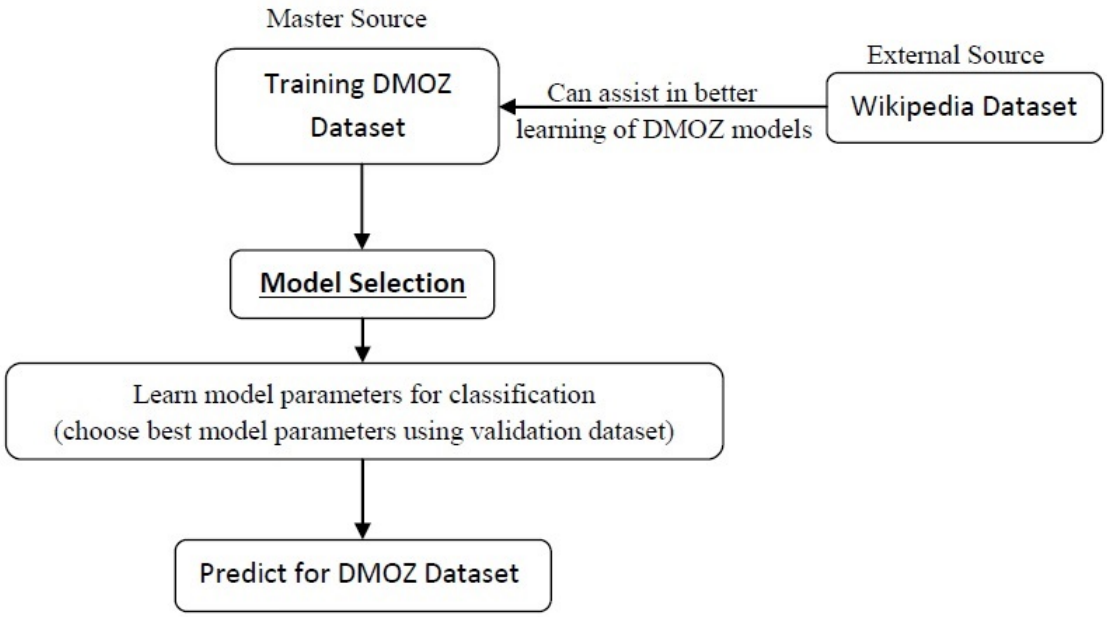


Figure 1.1: Task: Testing various models and selecting the best one based on accuracy/runtime

classify a given master source database (DMOZ) with accuracy and efficiency (refer to Figure 1.1). To improve the model accuracy, we use an external source in the form of the wikipedia dataset in conjunction with DMOZ dataset. Using external source helps in learning the better model parameters for source classes, by increasing the number of positive examples, especially when the number of training examples for a given class is less. We have evaluated various models by dividing the DMOZ dataset into three different categories, namely, low distribution (containing less positive examples per class), medium distribution (containing average positive examples per class) and high distribution (containing huge positive examples per class). We have also evaluated the models performance by using various random projections (hashing) technique, which leads to better runtime for learning model parameters but at the cost of reduced accuracy.

1.2 Contributions

We have contributed towards developing new MTL based models for integrating two large document archieves i.e. DMOZ and wikipedia. We treat the DMOZ database as the master source database and wikipedia as the external source database. Depending upon how we make use of the external source database, we can categorize our contribution as,

1) MTL model based on Neighborhood Pooling Approach (MTL-NPA),

2) MTL model based on Individual Neighborhood Approach (MTL-INA).

In MTL-NPA we pool all the k-nearest external source neighbors for each class of main database as one neighbor and then learn the model parameters whereas in MTL-INA we consider each of the k-nearest neighbors from the external source database for each class of master database as individual task and then learn the model parameters.

1.3 Thesis Outline

The rest of the thesis is organized as follows. Chapter 2 provides a thorough review of various methods for learning model parameters along with various hashing techniques. Chapter 3 covers the various methods we have used for designing the models. Chapter 4 provides the result of all the models discussed in Chapter 3, in terms of accuracy and runtime. This chapter also provides the hashing result, in terms of accuracy and runtime. Chapter 5 describes several directions for improving the models accuracy and runtime.

Chapter 2: Background

2.1 Multi-Task Learning

Multi-Task Learning (MTL) [13] is designed to simultaneously learn models for multiple tasks (refer to Figure 2.1), as opposed to Single-Task Learning (STL), where for each task, the model parameter are learned independently. The main motivation behind using MTL is that training signal present in related tasks can help each of the tasks learn better model parameters[13][14]. It has been shown [15][16] that MTL improves the performance of the model, especially when the number of training examples is less and when the tasks are related. There has been huge research exploration in the MTL research and the literature about the MTL survey can be found in Zhou et. al [17].

Given a training set, with n input-output pairs (x_1, y_1), (x_2, y_2), ... (x_n, y_n), the goal is to learn a mapping function $f : X \rightarrow Y$ between the input domain $x_i \in X$ and output domain $y_i \in Y$. X and Y are input and output domain, respectively. The objective is to learn a model that minimizes the loss function on the training data while constraining the model complexity with a regularization penalty. The learning objective for the regularized STL can be given as,

$$\min_{\theta} \sum_{i=1}^{n} \underbrace{L(\theta, x_i, y_i)}_{loss} + \lambda \underbrace{R(\theta)}_{Regularization} \tag{2.1}$$

where θ represents the model parameters. The regularization term controls the model complexity, thus safeguarding against the model over-fitting. Extension of STL is MTL which learns the model parameters of the related task together. In MTL we are given T tasks with training set defined for each of the $t = 1...T$ tasks, given by $(x_{it}, y_{it}) : i = 1...n_t$,

and the combined learning objective is given by,

$$\sum_{t=1}^{T}\sum_{i=1}^{n_t}\underbrace{L(\theta_t, x_{it}, y_{it})}_{loss} + \lambda \underbrace{R(\Theta)}_{Regularization} \tag{2.2}$$

where n_t is the number of training instances for the t^{th} task, θ_t denotes the model parameters for the t^{th} task, (x_{it}, y_{it}) represents the i^{th} input and output pair for t^{th} task, and $\Theta = \{\theta_t\}_{t=1}^{T}$ is the combined set of model parameters for all the related tasks. Various multi-task learning methods take this general approach to build combined models for many related tasks. In Evgeniou et. al [18] the model for each task is constrained to be close to the average of all the tasks. In multi-task feature learning and feature selection methods [19][20][21][22], sparse learning based on lasso [23], is performed to select or learn a common set of features across many related tasks. However, a common assumption made by many methods [18][24][25] is that all tasks are equally related. This assumption does not hold in all situations.

Therefore, it is sensible to take the task relationships into account in multi-task learning. Kato et. al [26] and Evgeniou et. al [27] propose formulations which use an externally provided task network or graph structure. However, these relationships might not be available and may need to be determined from the data. Clustered multi-task learning approaches assume that tasks exhibit a group structure, which is not known a-priori and seeks to learn the clusters of tasks that are learned together [28][29][30]. Another set of approaches, mostly based on Gaussian Process models, learn the task co-variance structure [31][32] and are able to take advantage of both positive and negative correlations between the tasks.

In this thesis we have focused on the MTL based models for the purpose of multi-class text classification. We use nearest neighbor based approach to find the related tasks within a different domain (database).

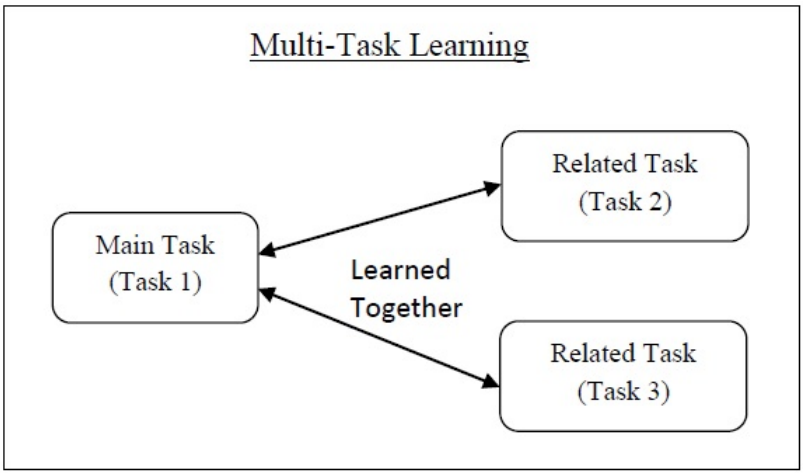


Figure 2.1: Information flow in Multi-Task Learning

2.2 Transfer Learning

Transfer Learning (TL) is designed to learn the parameters of the main task (also know as parent task) based on the trasferred parameters from the related task(s) (also known as children tasks) (refer to figure 2.2). Main intuition behind using TL is that the information contained in the children task can help in learning the better predictive models for the main task. When transferred parameters from the child task assist in better learning the predictive models of the parent task than it is referred to as positive transfer. However, in some cases if related task(s) are not found correctly than TL may lead to the predictive models which is worse than the original predictive models without transfer and this type of transfer is known as negative transfer. It has been shown in the work of Pan et. al [33] that TL improves the generalization performance of the predictive models provided the related task(s) are simliar to each other.

Goal of TL is to learn the mapping function $f : X \rightarrow Y$ from n input-output pairs (x_1, y_1), (x_2, y_2), ... (x_n, y_n) in such a way so as to minimize the objective function given by,

$$\min_{\theta} \sum_{i=1}^{n} \underbrace{L(\theta, x_i, y_i)}_{loss} + \lambda \underbrace{R(\theta)}_{Regularization} + \lambda^* \underbrace{R(\theta^*)}_{Regularization} \tag{2.3}$$

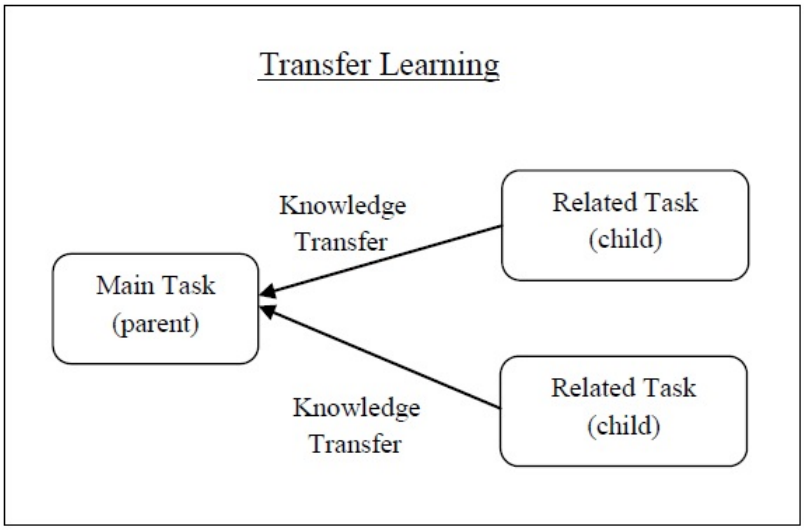


Figure 2.2: Knowledge transfer in Transfer Learning

where θ is the model parameters, θ^* is the learned parameter that is transferred from the children task, regularization term $R(\theta)$ control the models from over-fitting, and regularization term $R(\theta^*)$ take into account the parameters of the related task to capture the information present in the related task.

TL differ from MTL in terms of parameters learning behavior. In MTL, all related task parameters are learned simultaneously whereas in TL related task parameters are learned first which is then trasferred to the main task of interest. TL has also been referred to as Assymetric Multi-Task Learning because of it's focus on one of the tasks, referred to as the parent/main task.

2.3 Semi-Supervised Learning

Semi-Supervised Learning (SSL) involves use of both labeled and unlabeled data for predicting the parameters of the model. SSL approaches between unsupervised (no labeled training data) and supervised learning (completely labeled training data) [34]. SSL works on the principle that more the training examples, better the generality. However, the result of SSL is largely dependent on how accurately we group the unlabeled data with the labeled data. More accurate the grouping of unlabeled data with labeled data better the result.

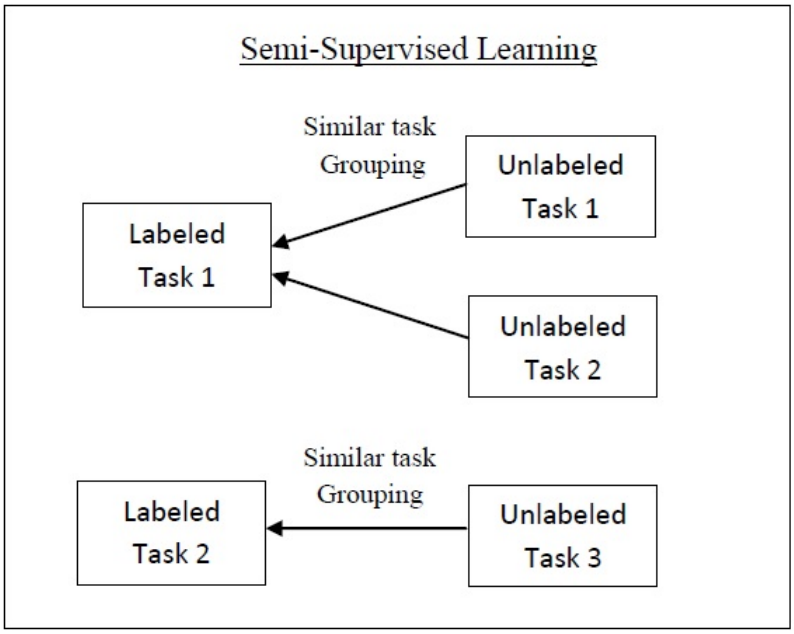


Figure 2.3: Semi Supervised Learning Method

For developing models for DMOZ class dataset classification using SSL method we have used kNN with tanimoto similarity to find the groupings of unlabeled wikipedia dataset with each of the DMOZ class. We have tested SSL models performance by using k value as 2 and 5.

2.4 Random Projections (Hashing)

Random projections are gaining popularity for reducing the dimension of a huge dataset. It is used for improving the runtime performance of the model and also helps in reducing the memory required for learning the model parameters. Many literature related to different hashing algorithm can be found in [35][36][37][38]. In this thesis we focus on three hashing algorithms namely, Locality Sensitive Hashing (LSH) [35], b-bit Minwise Hashing (bMH) [36], and One Permutation Hashing (OPH) [37].

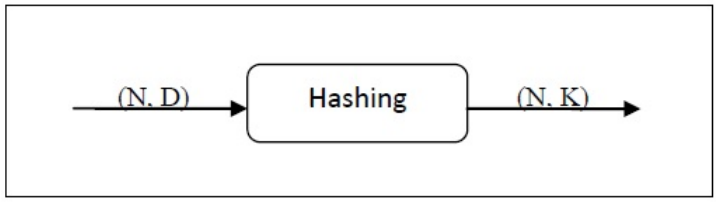


Figure 2.4: Hashing Method: N = No. of Instance, D = No. of features before hashing, K = No. of features after hashing, $D > K$

2.4.1 Locality Sensitive Hashing

Locality Sensitive Hashing (LSH) works by randomly choosing K feature indices $i_1, i_2, ..., i_K$ from the D dimensional feature data, where $D > K$ (refer to Figure 2.5), to define a hash function $f(s)$ given by:

$$f(s) =< s[i_1], s[i_2], ..., s[i_K] > \tag{2.4}$$

The function $f(s)$, given in Equation 2.5 produces a concatenated features of length K. Function $f(s)$ is called "locality hashing" because the probability that pair of choosen feature vectors f_1 and f_2 have the same hash value varies directly with their similarity. For improving the accuracy we repeat the experiment l times with different random values of K. We have tested our algorithm with different values of K and l to obtain the best possible accuracy.

2.4.2 b-bit Minwise Hashing

b-bit Minwise Hashing (bMH) is similar to the minwise hashing [38] with the only difference that the former uses b-bits as compared to 64 bits used by the minwise hashing. Minwise algorithm works by choosing n min-wise independent permutations, represented as π_1, π_2, ..., π_n. The problem with this approach is that it is not feasible to permute a large dimensional dataset, since drawing random permutations is time consuming and practically inefficient. So to overcome this problem we have used universal hashing functions to simulate the effect of random permutations. This requires storing few hash values [39]. The standard universal

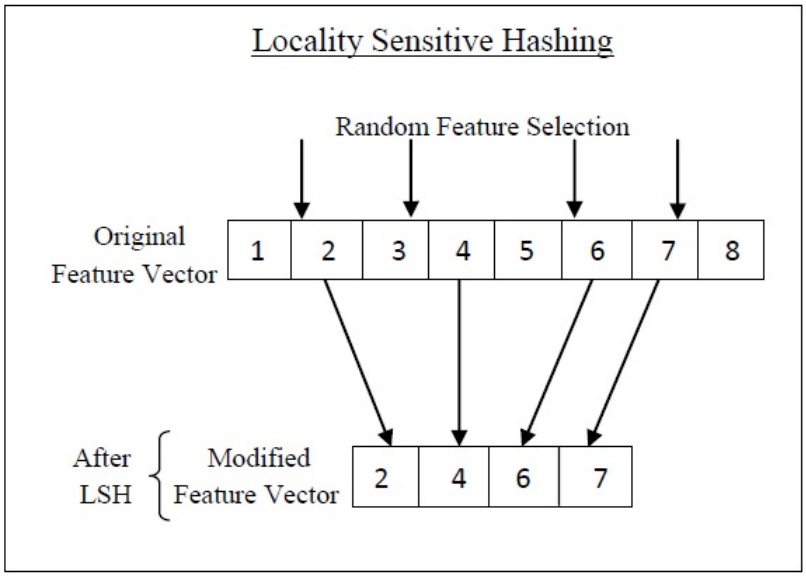


Figure 2.5: Locality Sensitive Hashing Method

hash function [40] is defined as,

$$h_i(x) = (\ (a_i x + b_i)\ mod\ p)\ mod\ m, \qquad i = 1, 2, ..., n \tag{2.5}$$

where, m is the dimension of the dataset, p is a prime number ($p > m$) and n is the number of hash functions. The parameters a_i and b_i are chosen uniformly from $\{0, 1, ..., p-1\}$. Instead of storing π_i, we now only need to store $2n$ numbers, a and b for each hash function. So now, instead of doing n random permutations, we choose n universal hashing functions $\{h_1, h_2, ..., h_n\}$ to approximate the random permutations. To compute the minwise hash values for a given dataset having feature set I, we iterate over all non-zero features and map them to their hash values $h_i(x)$. We then iterate over all the hash values to find their minimum, which will be the i^{th} min-wise value for that feature set. We formulate a $minHash$ function as the smallest element of a set I under ordering induced by the universal hashing function h, given by,

$$minHash(h(I)) = arg \min_{x \in I} h(x) \tag{2.6}$$

Using the min-wise hashing property, the probability of hashing collision for two sets is equal to their Jaccard similarity. b-bit minwise hashing is very efficient in terms of the storage and computation cost. This is due to the fact that it uses only b-bits instead of 64 bits used by minwise hashing. It has been shown in Li et. al [36] that bMH can be easily integrated with logistic regression. bMH apply k random permutations on each feature vector x_i and store the lowest b bits of each hashed value. Thus, we obtain a new dataset which can be stored using merely nbk bits. At run-time, we expand each new data point into a $2^b k$-length vector with exactly k $1's$. For example, suppose $k = 4$ and the hashed values are originally $\{20, 24, 33, 39\}$, whose binary digits are $\{010100, 011000, 100001, 100111\}$. Consider $b = 3$, then the binary digits are stored as $\{100, 000, 001, 111\}$(which corresponds to $\{4, 0, 1, 7\}$ in decimals). At run-time, we need to expand them into a vector of length $2^b k = 32$, to be $\{0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0\}$, which will be the new feature vector. We have tested our algorithm with different values of k and b.

2.4.3 One Permutation Hashing

One of the main disadvantage of b-bit minwise hashing is it's expensive preprocessing time. One Permutation Hashing (OPH) [37] overcomes large preprocessing time by reducing n number of permutation to single permutation, thus reducing preprocessing time by huge fold. In one permutation hashing each example is viewed as a $0/1$ vectors in D dimensions to get collection of sets as a binary data matrix in D dimensions. After that randomly permute the features (columns) of the data matrix followed by dividing the columns into k parts (bins) and then for each data vector take the smallest nonzero element in each bin. Next step after this is similar to bMH.

Chapter 3: Methods

This chapter discusses in detail the various approaches and methods used in selecting the models for large-scale text classification. Our task is to design model for categorizing the DMOZ dataset (master source) by leveraging wikipedia dataset (external source). To make use of wikipedia dataset for improving the model performance to categorize DMOZ dataset we use k-nearest neighbor (kNN) approach to find the similarity between each of DMOZ classes with the wikipedia classes. Depending upon how we use the similarity class from wikipedia dataset to improve model performance, we can divide model learning into three major categories, namely Semi-Supervised Learning (SSL), Transfer Learning (TL) and Multi-Task Learning (MTL).

3.1 k-Nearest Neighbor

k-Nearest Neighbor (kNN) [41][42] is non parametric lazy learning algorithm which does not make any assumptions on the underlying data distribution. It is used for grouping similar objects at the cost of expensive runtime. In this thesis, kNN method is used for finding the similar classes between the DMOZ and wikipedia datasets. We seek to improve classification models for our master source dataset (DMOZ) by using neighboing class found in our external source dataset (wikipedia). For finding nearest neighbor, we represent each of the class of DMOZ and wikipedia dataset by its representative vector, which is calculated by taking average of all examples in the particular class. We than use tanimoto similarity measure for finding the similarity between the classes. Tanimoto similarity between the two class is given as,

$$T_{sim}(A, B) = \frac{N_{AB}}{(N_A + N_B - N_{AB})} \tag{3.1}$$

where, N_{AB} is the dot product of number of features common to both A and B, N_A is the number of non-zero features in document A, N_B is the number of non-zero features in document B.

SSL method considers all the k-nearest wikipedia neighbor to be the positive examples for respective DMOZ class and than learns the parameters of the model, essentially we are increasing the number of positive examples of each class of DMOZ dataset. We have tested the SSL model performance by choosing the value of k as 2 and 5.

TL method makes use of wikipedia dataset by first learning the parameters of the k nearest neighbor and than transferring the learned parameters to respective DMOZ classes. Idea behind using this method is that, being nearest neighbor there model parameters should be close to each other. Based on how we learn the parameters of the nearest neighbor we can further categorize TL method into two categories, neighborhood pooling approach (NPA) and individual neighborhood approach (INA). In NPA we consider all the k nearest wikipedia neighbors of each DMOZ class to be one neighbor and than learn the parameters of the neighbor whereas in INA we learn the parameters of each of the k nearest neighbor individually before transferring. DMOZ model improves its performance by making use of the transferred parameters. We have tested the model performance by choosing k value as 2 and 5.

MTL method simultaneously learns the parameters of the DMOZ class and its k-nearest neighbor respectively. Idea behind learning the model parameters together (often referred to as joint training) is that each of the related task can help learning better model parameters of each other by capturing the intrinsic relatedness between the task. Based on how we use the neighbor we can divide in further into two categories, NPA and INA. In NPA all k nearest neighbors are pooled together as one neighbor whereas in INA all k nearest neighbors are considered as individual task while learning. Again we tested MTL method model performance by choosing the k value as 2 and 5.

To sum up we have implemented ten different models (refer to Figure 3.1), namely, SSL ($k = 2$ and $k = 5$), TL-NPA ($k = 2$ and $k = 5$), TL-INA ($k = 2$ and $k = 5$), MTL-NPA

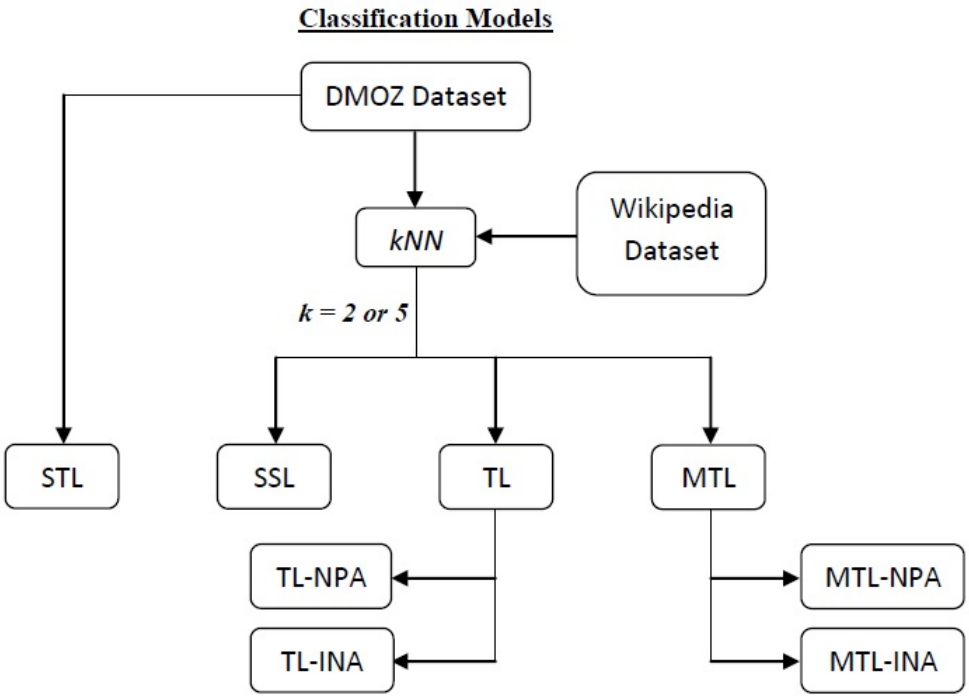


Figure 3.1: Various Classification Models

($k = 2$ and $k = 5$), and MTL-INA ($k = 2$ and $k = 5$). Each of the model performance is compared to the baseline Single-Task Learning (STL) model. Detail implementation of parameter learning for each of this model is discussed below,

3.2 STL method for parameter learning

As discussed in Chapter 2, learning objective for the regularized Single-Task Learning is represented as,

$$\min_{\theta} \sum_{i=1}^{n} \underbrace{L(\theta, x_i, y_i)}_{loss} + \lambda \underbrace{R(\theta)}_{Regularization} \tag{3.2}$$

In this thesis logistic regression is used as the loss function. Logistic regression make use of logistic loss function which is given by,

$$P(x) = \frac{1}{1 + exp(-x)} \tag{3.3}$$

where, $x \in \mathbb{R}$ is the variable of the function and $P(x) \in [0, 1]$. One advantage of using this loss function is that it is convex and hence is differentiable at all points. Logistic regression is given by,

$$P(y|\theta, x) = \frac{1}{1 + exp(-y\theta^T x)} \tag{3.4}$$

where, $y \in \{\pm 1\}$ is the label for data item x, θ is the model parameters. More detail theory behind logistic regression can be found in [43] [44]. For preventing the model from overfitting we have used the regularization term $\sum_{j=1}^{d} \frac{\lambda}{2}||\theta_j||^2$, where, λ is the regularization parameter. Thus the final equation for learning objective of STL can be written as,

$$\min_{\theta} \sum_{i=1}^{n} \frac{1}{1 + exp(-y_i \theta^T x_i)} + \sum_{j=1}^{d} \frac{\lambda}{2}||\theta_j||^2 \tag{3.5}$$

3.3 SSL method for parameter learning

SSL method works the same way as the STL method with only difference in the number of positive examples. SSL method increases the number of positive examples for each class by some mechanism. In this thesis number of positive example for each DMOZ class is increased by finding the nearest neighbor of each class of DMOZ with the Wikipedia dataset using the kNN approach. Each of this nearest neighbor are labeled as the positive examples for that particular class, thus making it semi-supervised learning (refer to Figure 3.2). Evaluation has been done for SSL with k value as 2 and 5 *i.e.* with 2 and 5 nearest neighbor. Although the number of positive examples is increased in SSL, there has been no guarantee that the

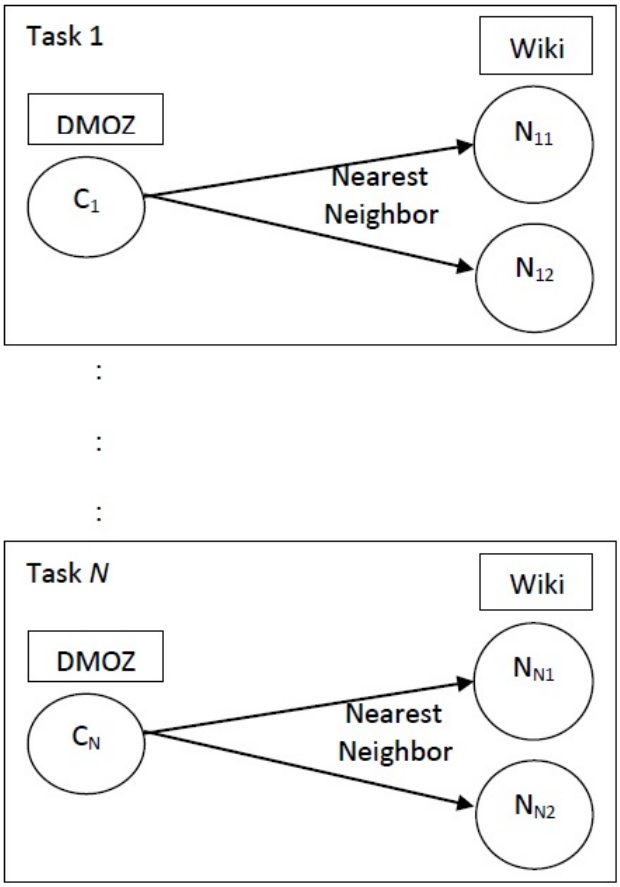


Figure 3.2: SSL for increasing the number of positive example for each class, $k = 2$ and N_{11}, N_{12} are the nearest neighbor for class C_1 of DMOZ and so on

accuracy will improve.

3.4 TL method for parameter learning

TL method makes use of the nearest neighbor learned parameters in the regularization term for learning the parameters for each class. Motivation behind using nearest neighbor parameters is that being nearest neighbor their parameters should be close to each other. Depending upon how we learn the parameter of the nearest neighbor we can divide TL in two categories: Neighborhood pooling approach and Individual neighborhood approach.

3.4.1 Neighborhood Pooling Approach (TL-NPA)

Parameters of the nearest neighbor are learned using the logistic regression as the loss function and regularization term as $\sum_{j=1}^{d} \frac{\lambda}{2}||\theta_j||^2$ (λ being the regularization parameter). However we consider all k neighbor of the class as single neighbor and consider all examples present in this k neighbor as positive examples for training (refer to Figure 3.3). We evaluate the result by taking k value as 2 and 5. Equation for learning the neighborhood parameter can be expressed as,

$$\min_{\theta_{n^*}} \sum_{i=1}^{N} \frac{1}{1 + exp(-y_i \theta_{n^*}^T x_i)} + \sum_{j=1}^{d} \frac{\lambda_n}{2} ||\theta_{n^* j}||^2 \tag{3.6}$$

where, θ_{n^*} is the neighborhood parameters of the n^{th} class of DMOZ dataset and λ_n is the regularization parameter preventing the model from overfitting.

After calculating the neighborhood parameters we can learn the parameter of the DMOZ class as,

$$\min_{\theta} \sum_{i=1}^{n} \frac{1}{1 + exp(-y_i \theta^T x_i)} + \sum_{j=1}^{d} \frac{\lambda}{2} ||\theta_j||^2 + \sum_{j=0}^{d} \frac{\lambda^*}{2} ||\theta_j - \theta_{n^* j}||^2 \tag{3.7}$$

where, λ and λ^* are the regularization parameter, λ prevents the model from overfitting and λ^* controls the degree by which model parameters should be close to neighborhood parameters.

3.4.2 Individual Neighborhood Approach (TL-INA)

In this approach, each of the k nearest neighbor model parameters of the DMOZ class are learned individually (refer to Figure 3.4). Equation for learning each of the neighborhood

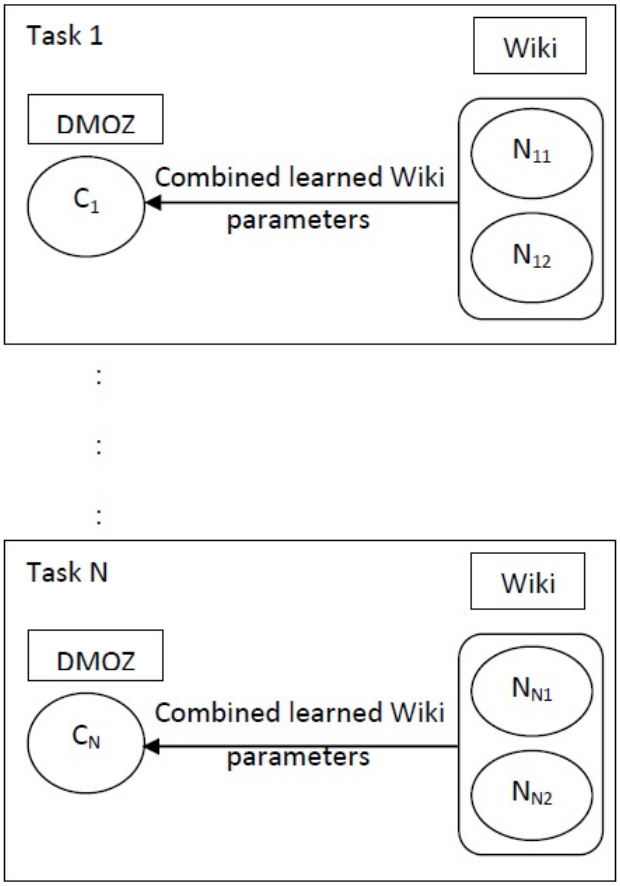


Figure 3.3: TL-NPA method for parameter learning of each class, $k = 2$

model parameters can be expressed as,

$$\min_{\theta_{nk}} \sum_{i=1}^{m} \frac{1}{1 + exp(-y_i \theta_{nk}^T x_i)} + \sum_{j=1}^{d} \frac{\lambda_n}{2} ||\theta_{nkj}||^2 \tag{3.8}$$

where, k is the number of nearest neighbor into consideration and θ_{nk} is the parameter of k^{th} neighbor of the n^{th} class of DMOZ dataset. Once we have the parameters of the neighborhood class of the DMOZ dataset we can calculate the parameters of the class of DMOZ dataset as,

$$\min_{\theta} \sum_{i=1}^{n} \frac{1}{1 + exp(-y_i \theta^T x_i)} + \sum_{j=1}^{d} \frac{\lambda}{2} ||\theta_j||^2 + \sum_{k=1}^{n_k} \sum_{j=0}^{d} \frac{\lambda_{nk}}{2} ||\theta_j - \theta_{nkj}||^2 \tag{3.9}$$

where, λ and λ_{nk} are the regularization parameter, λ prevents the model from overfitting and λ_{nk} controls the degree by which model parameters should be close to each of the n_k model parameters of the neighborhood.

3.5 MTL method for parameter learning

In MTL method all the related class model parameters are learned simultaneously. Motivation behind using MTL is that each of the related class can help learn the parameters of the each other better. Depending upon how we learn the parameter of each other we can divide MTL in two categories: Neighborhood pulling approach and Neighborhood individual approach.

3.5.1 Neighborhood Pooling Approach (MTL-NPA)

In this approach, all k neighboring parameters of DMOZ class are considered as single neighbor, same as in case of TL-NPA. However, MTL-NPA diifers from TL-NPA in the paramameter learning. In MTL-NPA parameters are learned in parallel(refer to Figure 3.5)

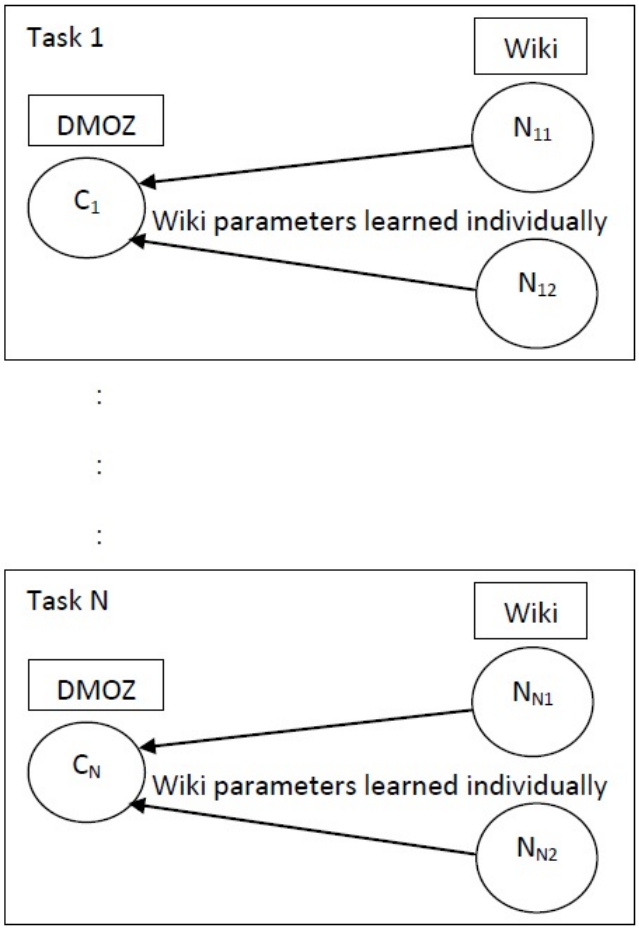


Figure 3.4: TL-INA method for parameter learning of each class, $k = 2$

while in TL-NPA first neighborhood parameters are learned then DMOZ class paramaters are learned making use of learned neighborhood parameters. MTL-NPA goal is to minimize the objective function given by,

$$\sum_{t=1}^{T}\sum_{i=1}^{n_t}\frac{1}{1+exp(-y_{it}\theta_t^T x_{it})}+\sum_{t=1}^{T}\frac{\lambda_t}{2}\sum_{j=1}^{d}||\theta_{tj}||^2+\frac{\lambda^*}{2}\sum_{t=2}^{T}\sum_{j=0}^{d}||\theta_{1j}-\theta_{tj}||^2 \tag{3.10}$$

where, θ_t are the model parameter of the t^{th} task (θ_1 is the model parameter for DMOZ class, θ_2 is the parameter for neighbor class of DMOZ), λ_t is the regularization parameter preventing the model from overfitting and λ^* is the parameter controlling the degree of relatedness between DMOZ class and its neighbor. Note that number of task T in this method is equal to 2, $T = 1$ being for DMOZ and $T = 2$ being for its neighbor.

3.5.2 Individual Neighborhood Approach (MTL-INA)

This approach is similar to TL-INA except in TL-INA first individual neighbor parameters are learned which is then used in learning the parameters of DMOZ class, while in MTL-INA all the individual neighbor parameters and DMOZ class parameters are learned in parallel (refer to Figure 3.6). Goal of MTL-INA is to minimize the objective function given by,

$$\sum_{t=1}^{T}\sum_{i=1}^{n_t}\frac{1}{1+exp(-y_{it}\theta_t^T x_{it})}+\sum_{t=1}^{T}\frac{\lambda_t}{2}\sum_{j=1}^{d}||\theta_{tj}||^2+\frac{\lambda^*}{2}\sum_{t=2}^{T}\sum_{j=0}^{d}||\theta_{1j}-\theta_{tj}||^2 \tag{3.11}$$

where, θ_t are the model parameter of the t^{th} task (θ_1 is the model parameter for DMOZ class, θ_t is the parameter for neighbor class of DMOZ, where $t = 2...(k+1)$), λ_t is the regularization parameter preventing the model from overfitting and λ^* is the parameter controlling the degree of relatedness between DMOZ class and its individual neighbor. Note that number of task T in this method is equal to (k+1), $T = 1$ being for DMOZ and $T = 2...(k + 1)$ being for its neighbor.

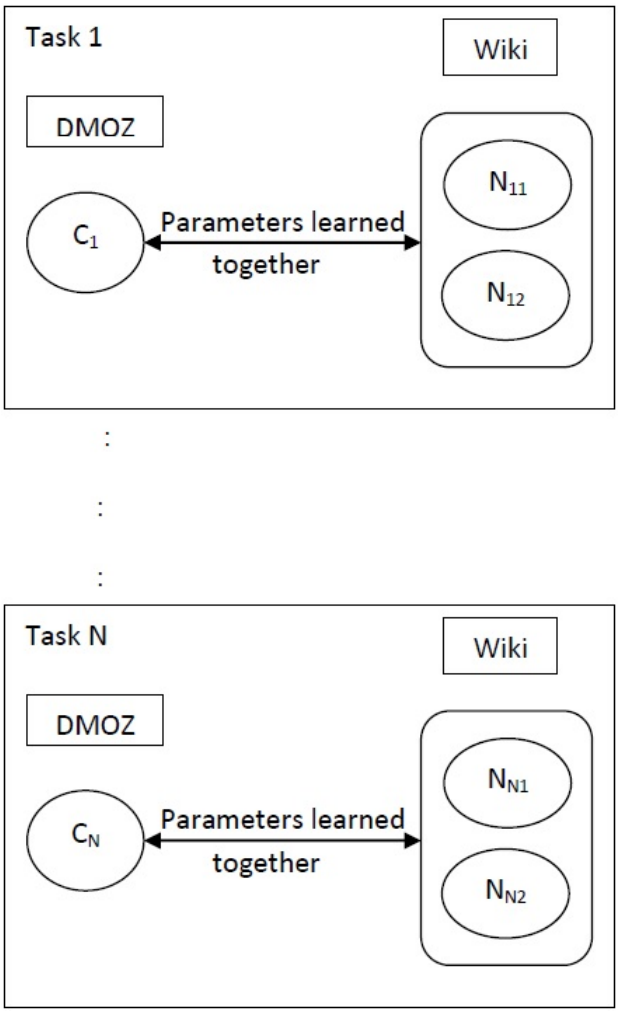


Figure 3.5: MTL-NPA method for learning parameter together for related class, $k = 2$

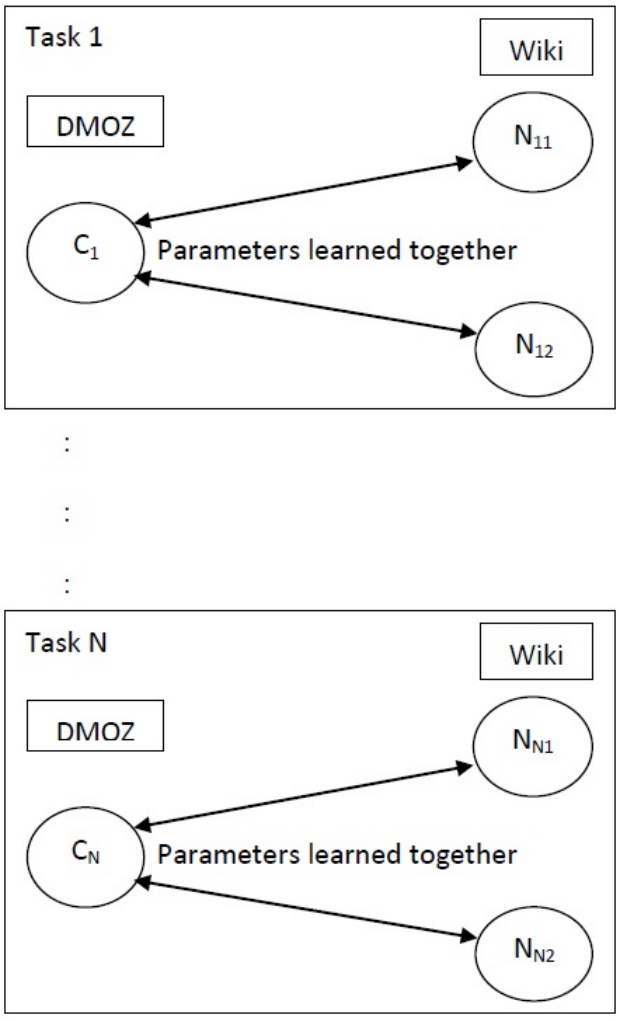


Figure 3.6: MTL-INA method for learning parameter together for related class, $k = 2$

Chapter 4: Evaluation and Result

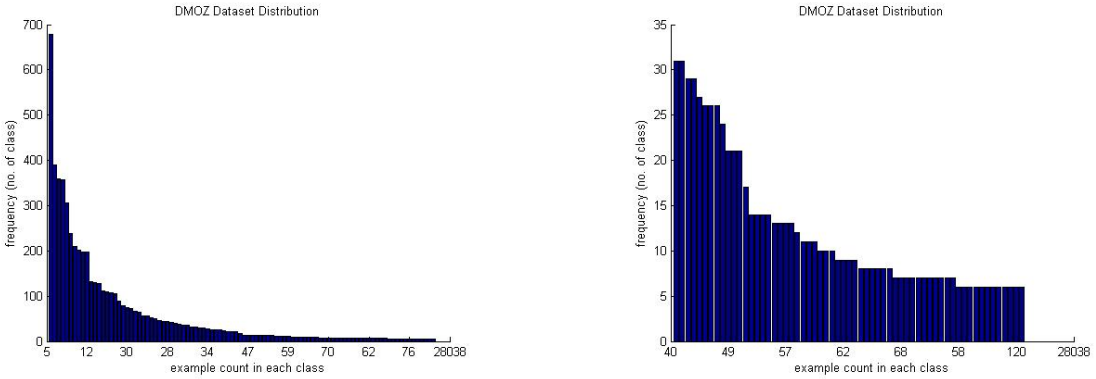
4.1 Dataset

DMOZ and Wikipedia dataset are needed in this thesis. The dataset has been taken from the ECML/PKDD 2012 discovery challenge on Large Scale Hierarchical Text Classification (LSHTC) - Track 2 challenge website[1]. Since, we don't know the label for theDMOZ test dataset and the challenge is over, we have used DMOZ train dataset for training as well as testing our model. Distribution plot of DMOZ training dataset, sorted in descending order, is shown in Figure 4.1 (a) and (b) respectively, wikipedia dataset is used as supplementary dataset in TL and MTL for better learning the parameters of the models associated with DMOZ classes. We have divided the train dataset into 3:1:1 ratio to be used as train (for training the model), validation (for determining the best value of unknown paramater) and test (for testing the model) dataset for our model testing. Also, to explore which algorithm performs best depending upon the number of examples in each class we have categorized the training dataset into three different parts. First part contains the DMOZ classes with very few positive examples (also referred as category 1 or low distribution), second part contains the classes with average number of positive examples (also referred as category 2 or medium distribution) and the last part contains classes with huge number of positive examples (also referred as category 3 or high distribution). We tested all of our models mentioned in chapter 3 for accuracy and runtime. We have also tested our model after applying various hashing techniques explained in chapter 3.

Each example in DMOZ and Wikipedia dataset is represented as sparse format. Format of each example is $< label feat : val ... feat : val >$, where, *label* is an integer that corresponds to a category in which the vector belongs. Each vector belongs to only one category

[1] http://lshtc.iit.demokritos.gr/LSHTC3_DATASETS

i.e. single label, pair $feat : value$ corresponds to a non-zero feature with index $feat$ and value val. $feat$ is an integer and val is a double that corresponds to the term's count. Snapshot of dataset representation is shown in Figure 4.2.



(a) DMOZ Dataset Distribution Plot

(b) Sampled DMOZ Dataset Distribution Plot

Figure 4.1: DMOZ Dataset distribution plot with example count in x-axis, frequency count of no. of class with given example count in y-axis, arranged in descending order

4.2 Metrics

In this thesis, we have used three standard metrics for evaluating the classification decisions that take into account True Positives (TP), False Positives (FP), True Negatives (TN) and False Negatives (FN) for each of the class respectively.

4.2.1 Micro-Averaged F_1

Micro-Averaged F_1 (μAF_1) is a conventional metric for evaluating classifiers in category assignments to test instances [45][46]. This metric gives the performance on each instance

```
33692 191:1 1434:1 10979:1
130762 84:1 1320:1 15103:2 2070846:1
352578 7345:1 50128:1
390846 9898:1 77846:1 81130:1
14661 696:1
...
```

Figure 4.2: Sample dataset representation

an equal weight in computing the average. The system made decisions on test set D with respect to a specific category $c \in C \equiv \{c_1, c_2, ..., c_{N_c}\}$ can be divided into four groups: True Positives (TP_c), False Positives (FP_c), True Negatives (TN_c) and False Negatives (FN_c), respectively. The corresponding evaluation metrics are defined as:

$$GlobalPrecision\ P = \frac{\sum_{c=1}^{N_c} TP_c}{\sum_{c=1}^{N_c}(TP_c + FP_c)} \tag{4.1}$$

$$GlobalRecall\ R = \frac{\sum_{c=1}^{N_c} TP_c}{\sum_{c=1}^{N_c}(TP_c + FN_c)} \tag{4.2}$$

$$Micro-averaged\ F_1(\mu AF_1) = \frac{2PR}{P+R} \tag{4.3}$$

where, N_c is the number of DMOZ classes.

4.2.2 Macro-Averaged Precision, Recall and F_1

Macro-Averaged Precision, Recall and F_1 (MAF_1) is another conventional metric for evaluating classifiers in category assignments [47]. This metric gives the performance on each category an equal weight in computing the average, defined as:

$$Category-specific\ Precision\ P_c = \frac{TP_c}{TP_c + FP_c} \tag{4.4}$$

$$Category-specific\ Recall\ R_c = \frac{TP_c}{TP_c + FN_c} \tag{4.5}$$

$$Macro-averaged\ Precision(MAP) = \frac{1}{N_c}\sum_{c=1}^{N_c}\frac{TP_c}{TP_c + FP_c} \tag{4.6}$$

$$Macro - averaged\ Recall(MAR) = \frac{1}{N_c}\sum_{c=1}^{N_c}\frac{TP_c}{TP_c + FP_c} \tag{4.7}$$

$$Macro - averaged\ F_1(MAF_1) = \frac{1}{N_c}\sum_{c=1}^{N_c}\frac{2P_cR_c}{P_c + R_c} \tag{4.8}$$

where, N_c is the number of DMOZ class.

4.2.3 Average Matthews Correlation Coefficient score

Matthews Correlation Coefficient (MCC) score [48] is another method for evaluating the classifiers result. It returns a value between -1 and +1. A coefficient of +1 represents a perfect prediction, 0 no better than random prediction and -1 indicates total disagreement between prediction and observation. MCC score for each class is defined as:

$$MCC_c = \frac{(TP_c * TN_c) - (FP_c * FN_c)}{\sqrt{(TP_c + FP_c)(TP_c + FN_c)(TN_c + FP_c)(TN_c + FN_c)}} \tag{4.9}$$

$$Average\ MCC\ (AMCC) = \frac{1}{N_c}\sum_{c=1}^{N_c} MCC_c \tag{4.10}$$

4.3 Accuracy Comparison

In this section we report the result of all models described in Chapter 3 in terms of metrics defined in section 4.2. We evaluated the performance of the models in three different categories. First category contains the sampled DMOZ dataset with each class having low distribution positive examples (number of classes = 75), second category corresponds to the sampled DMOZ dataset with medium distribution positive examples per class (number of classes = 53) and the last category corresponds to the DMOZ dataset containing high distribution positive examples per class (numberof classes = 18). For the purpose of testing

our model in each of these categories we have divided the DMOZ dataset into three parts in the ratio 3:1:1, to be used as train, validation and test dataset. Train dataset is used for training the model, validation dataset is used to determine the best parameters of the model and test dataset is used for testing the model.

4.3.1 Category 1: Low distribution sample DMOZ dataset

Table 4.1 shows the average performance across five runs on sampled DMOZ dataset with each class having low distribution examples. Following comparison result can be concluded from the result Table 4.1.

- **STL v/s SSL v/s TL v/s MTL**: From the Table 4.1 it can be seen that MTL models perform much better than all the other models. This is expected for low distribution DMOZ class because when number of training examples are less MTL method learns the model parameters jointly with the related task in parallel resulting in better model. SSL models are slightly better than the STL model, this is due to the fact that number of positive examples increases in case of SSL and hence learns the model parameters better than STL. TL models accuracy is approximately same as that of STL model stating that there is no significant knowledge transfer from related task, reason for this is that there are less examples in the related task and hence learned parameters are not so good and so is the transfered parameters.

- **NPA v/s INA**: NPA models are better compared to its INA models counterpart. Reason for this is that there are less number of examples in case of each related class of INA compared to NPA.

- $k = 2$ **v/s** $k = 5$: $k = 5$ gives better models compared to $k = 2$ models in case of SSL, reason being more number of positive examples in case of $k = 5$ and hence better model, but in case of TL and MTL $k = 2$ models perform better, this may be due to the fact that 3^{rd}, 4^{th} and 5^{th} neighbor are not closely related to its corresponding DMOZ class.

Table 4.1: Low distribution sample DMOZ average performance across five runs for all models

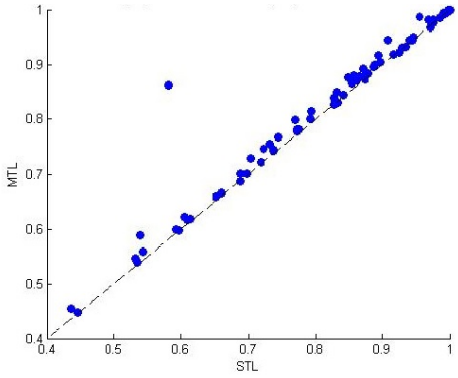
Model	μAF_1	MAP	MAR	MAF_1	AMCC
STL	0.5758 (0.0121)	0.7914 (0.0267)	0.6167 (0.0115)	0.6486 (0.0060)	0.6732 (0.0074)
SSL ($k = 2$)	0.6178 (0.0160)	0.8064 (0.0413)	0.6649 (0.0125)	0.6789 (0.0231)	0.7048 (0.0263)
SSL ($k = 5$)	0.6719 (0.0540)	**0.8091 (0.0406)**	0.7093 (0.0314)	0.7045 (0.0152)	0.7359 (0.0064)
TL-NPA ($k = 2$)	0.5739 (0.0044)	0.7987 (0.0278)	0.6247 (0.0091)	0.6481 (0.0062)	0.6732 (0.0074)
TL-NPA ($k = 5$)	0.5755 (0.0030)	0.8027 (0.0314)	0.6262 (0.0077)	0.6504 (0.0083)	0.6757 (0.0097)
TL-INA ($k = 2$)	0.5736 (0.0038)	0.7967 (0.0262)	0.6246 (0.0184)	0.6478 (0.0040)	0.6728 (0.0054)
TL-INA ($k = 5$)	0.5712 (0.0034)	0.8024 (0.0226)	0.6212 (0.0091)	0.6467 (0.0034)	0.6724 (0.0047)
MTL-NPA ($k = 2$)	**0.7442 (0.0201)**	0.7819 (0.0356)	**0.7840 (0.0169)**	**0.7373 (0.0349)**	**0.7527 (0.0335)**
MTL-NPA ($k = 5$)	0.7394 (0.0219)	0.7720 (0.0421)	0.7814 (0.0140)	0.7293 (0.0318)	0.7488 (0.0298)
MTL-INA ($k = 2$)	0.7208 (0.0180)	0.7583 (0.0503)	0.7664 (0.0211)	0.7052 (0.0520)	0.7326 (0.0367)
MTL-INA ($k = 5$)	0.7079 (0.0136)	0.7352 (0.0306)	0.7508 (0.0085)	0.6949 (0.0255)	0.7147 (0.0243)

*Table shows mean and (sd) in bracket, Standard Error (SE) for MTL-NPA ($k = 2$): 0.005413

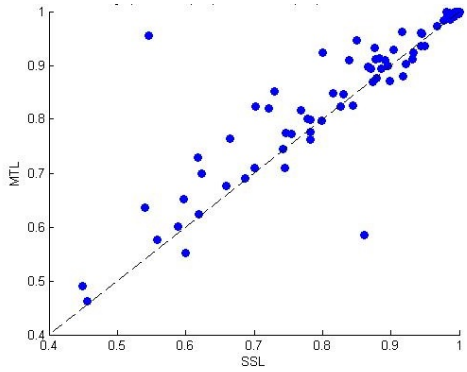
Figure 4.3 (a) gives the AUC comparison for the STL and best MTL method i.e. MTL-NPA ($k = 2$) for low distribution sample DMOZ dataset. It can be clearly seen that MTL-NPA ($k = 2$) method performs better compared to baseline STL method. Figure 4.3 (b) and 4.3 (c) shows the AUC comparison of best MTL method with best SSL ($k = 5$) and best TL method (TL-NPA ($k = 2$)) respectively. Clearly, in all of the cases MTL method performs better.

4.3.2 Category 2: Medium distribution sample DMOZ dataset

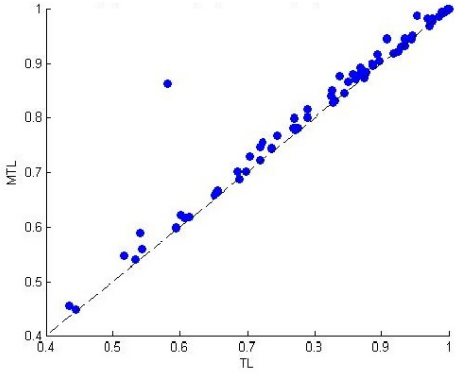
Table 4.2 shows the average performance across five runs on sampled DMOZ dataset with each class having medium distribution examples. Comparison result that is worth noting from the table 4.2 is,



(a) STL v/s MTL-NPA ($k = 2$)



(b) SSL ($k = 5$) v/s MTL-NPA ($k = 2$)



(c) TL-NPA ($k = 5$) v/s MTL-NPA ($k = 2$)

Figure 4.3: AUC comparison graph

- **STL v/s SSL v/s TL v/s MTL**: From the Table 4.2 it can be seen that MTL models perform better than all the other models. Reason being the same as explained for low distribution examples. However, the margin of difference between the models accuracy is less than the low distribution examples, this is due to the fact that as the number of training example increases, effect of better model parameter learning by increasing positive examples (in case of SSL), transferring learned parameter (in case of TL) and joint parameter learning (in case of MTL) diminishes.

- **NPA v/s INA**: INA models are better compared to its NPA models counterpart in case of MTL because of high task relatedness between the DMOZ class and each of its corresponding neighbors and hence simultaneous learning of parameters by considering the neighbors individually perform better. However in case of TL, NPA models are better compared to its INA models because of less significant learned parameter transfer in case of INA models, possibly due to less number of positive examples.

- $k = 2$ **v/s** $k = 5$: $k = 5$ gives better model compared to $k = 2$ models in case of SSL model, reason being more number of positive examples in case of $k = 5$ and hence better model, but in case of TL and MTL $k = 2$ models perform better this is due to the fact that 3^{rd}, 4^{th} and 5^{th} neighbor are not very well related to its corresponding DMOZ class.

4.3.3 Category 3: High distribution sample DMOZ dataset

Table 4.3 shows the average performance across five runs on sampled DMOZ dataset with each class having high distribution examples. It can be noted from the Table 4.3 that all the models perform nearly same, this is because of the fact that as the number of examples become huge, increasing positive examples (in case of SSL), learned parameter transfer (in case of TL) and joint parameter learning (in case of MTL) does not influence much the model parameters learning or in other words when training examples are huge even the simpler models can perform as good as the complex models.

Table 4.2: Medium distribution sample DMOZ average performance across five runs for all models

Model	μAF_1	MAP	MAR	MAF_1	AMCC
STL	0.7592 (0.0120)	0.7947 (0.0068)	0.7618 (0.0056)	0.7567 (0.0085)	0.7626 (0.0075)
SSL ($k = 2$)	0.7535 (0.0040)	0.7938 (0.0065)	0.7601 (0.0011)	0.7547 (0.0052)	0.7609 (0.0048)
SSL ($k = 5$)	0.7545 (0.0027)	0.7948 (0.0060)	0.7610 (0.0018)	0.7559 (0.0039)	0.7619 (0.0034)
TL-NPA ($k = 2$)	0.7536 (0.0042)	0.7936 (0.0054)	0.7588 (0.0015)	0.7546 (0.0048)	0.7610 (0.0043)
TL-NPA ($k = 5$)	0.7533 (0.0038)	0.7937 (0.0063)	0.7584 (0.0024)	0.7542 (0.0052)	0.7605 (0.0046)
TL-INA ($k = 2$)	0.7529 (0.0040)	0.7958 (0.0054)	0.7591 (0.0011)	0.7540 (0.0048)	0.7603 (0.0043)
TL-INA ($k = 5$)	0.7527 (0.0038)	0.7957 (0.0046)	0.7590 (0.0010)	0.7538 (0.0036)	0.7602 (0.0042)
MTL-NPA ($k = 2$)	0.7572 (0.0080)	0.7961 (0.0063)	0.7637 (0.0058)	0.7587 (0.0087)	0.7644 (0.0076)
MTL-NPA ($k = 5$)	0.7569 (0.0043)	0.7969 (0.0058)	0.7627 (0.0012)	0.7581 (0.0053)	0.7639 (0.0048)
MTL-INA ($k = 2$)	0.7579 (0.0103)	0.7978 (0.0076)	0.7644 (0.0079)	0.7599 (0.0111)	0.7657 (0.0099)
MTL-INA ($k = 5$)	**0.7657 (0.0067)**	**0.8017 (0.0073)**	**0.7717 (0.0037)**	**0.7678 (0.0081)**	**0.7726 (0.0072)**

*Table shows mean and (sd) in bracket, Standard Error (SE) for MTL-INA ($k = 5$): 0.0052

Table 4.3: High distribution sample DMOZ average performance across five runs for all models

Model	μAF_1	MAP	MAR	MAF_1	AMCC
STL	0.8414 (0.0112)	0.8417 (0.0086)	0.8420 (0.0106)	0.8378 (0.0105)	0.8308 (0.0106)
SSL ($k = 2$)	0.8404 (0.0103)	0.8407 (0.0080)	0.8411 (0.0098)	0.8367 (0.0095)	0.8297 (0.0096)
SSL ($k = 5$)	0.8404 (0.0099)	0.8413 (0.0076)	0.8414 (0.0098)	0.8367 (0.0093)	0.8299 (0.0095)
TL-NPA ($k = 2$)	0.8411 (0.0108)	0.8414 (0.0082)	0.8419 (0.0102)	0.8374 (0.0101)	0.8304 (0.0101)
TL-NPA ($k = 5$)	0.8412 (0.0119)	0.8416 (0.0084)	0.8420 (0.0108)	0.8375 (0.0100)	0.8304 (0.0096)
TL-INA ($k = 2$)	0.8410 (0.0108)	0.8414 (0.0080)	0.8419 (0.0104)	0.8374 (0.0101)	0.8305 (0.0101)
TL-INA ($k = 5$)	0.8211 (0.0109)	0.8414 (0.0080)	0.8419 (0.0100)	0.82372 (0.0102)	0.8304 (0.0098)
MTL-NPA ($k = 2$)	0.8414 (0.0107)	0.8417 (0.0081)	**0.8422 (0.0101)**	0.8377 (0.0100)	0.8307 (0.0101)
MTL-NPA ($k = 5$)	0.8414 (0.0108)	0.8415 (0.0084)	0.8421 (0.0104)	0.8376 (0.0104)	0.8306 (0.0104)
MTL-INA ($k = 2$)	0.8400 (0.0107)	0.8406 (0.0078)	0.8408 (0.0100)	0.8360 (0.0098)	0.8292 (0.0098)
MTL-INA ($k = 5$)	**0.8418 (0.0110)**	**0.8423 (0.0084)**	0.8366 (0.0189)	**0.8384 (0.0102)**	**0.8314 (0.0103)**

*Table shows mean and (sd) in bracket, Standard Error (SE) for MTL-INA ($k = 5$): 0.0067

Table 4.4: Average runtime (in sec.) performance across five runs for all models in low, medium and high distribution

Model	Low Distribution	Medium Distribution	High Distribution
STL	**2.7227**	**44.7567**	**33.3769**
SSL ($k = 2$)	4.5738	62.4305	46.1241
SSL ($k = 5$)	5.5787	62.7890	48.2801
TL-NPA ($k = 2$)	4.6570	48.6839	36.5657
TL-NPA ($k = 5$)	6.3166	52.6703	38.0725
TL-INA ($k = 2$)	4.5414	48.4921	36.4817
TL-INA ($k = 5$)	6.4049	50.3760	38.2257
MTL-NPA ($k = 2$)	5.5127	49.6016	36.4456
MTL-NPA ($k = 5$)	7.5563	54.3671	40.2081
MTL-INA ($k = 2$)	9.8754	56.7204	41.7689
MTL-INA ($k = 5$)	15.3910	78.6765	56.5831

4.4 Runtime Comparison

This section compares the models performance in terms of average runtime per class (in sec.). In Table 4.4, we report the average time (in sec.) needed to learn the model parameters of each class in each of the three categories discussed above, namely, low distribution, medium distribution and high distribution. Figure 4.4 represents the runtime graphical format. Clearly, it can be noted from table that STL model parameter learning takes the least time of all the other models because no overheads are involved. SSL models takes more time than its corresponding STL model because the number of examples is increased in case of SSL and hence the runtime. TL model runtime increases due to the fact that some time is spent in learning the neighborhood parameters. Finally, MTL method takes the maximum time of all the other models, reason behind this is that joint parameter learning leads to learning parameter updates in each of the class and its neighbor parameters together resulting in more time in each iteration and hence overall runtime. It can also be noted from table that medium distribution takes more time comparison to high distribution this is due to the fact that there are more classes in medium distribution compared to high distribution.

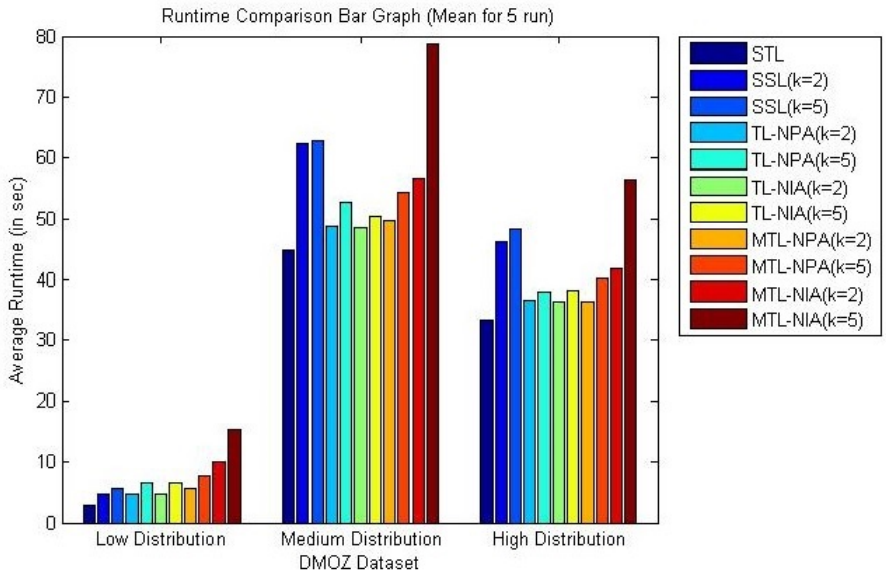


Figure 4.4: Average runtime (in sec.) comparison of all models for low, medium and high distribution

4.5 Use of Random Projections

4.5.1 Accuracy Comparison

Table 4.5 and 4.6 gives the average performance across five runs after applying LSH (parameter, $K = 4000$) on low and high distribution sampled DMOZ dataset respectively. Accuracy drops by $\approx 6\%$ between the best model (MTL-NPA ($k = 2$)) without hashing and corresponding hashing models after applying LSH (parameter, $K = 4000$) in case of low distribution while in case of high distribution it drops by $\approx 16\%$. Figure 4.5 shows the result for accuracy performance by varying the parameter K). Table 4.7 and 4.8 gives the average performance across five runs after applying b-bit minwise hashing (parameters, $b = 4$, $nP = 500$) on low and high distribution sampled DMOZ dataset respectively. From the table it can be seen that accuracy drops by $\approx 27\%$ in case of low distribution while it drops by $\approx 7\%$ in case of high distribution sample DMOZ dataset. Figure 4.6 shows the result for accuracy performance by varying the parameter (b, nP). Finally, Table 4.9 and 4.10 gives the average performance across five runs after applying one permutation hashing (parameters, $b = 4$, $nB = 500$) on low and high distribution sampled DMOZ dataset

Table 4.5: Low distribution sample DMOZ average performance across five runs after LSH (parameter, $K = 4000$) for all models

Model	μAF_1	MAP	MAR	MAF_1	AMCC
STL	0.4934 (0.1158)	0.6925 (0.0392)	0.5316 (0.1220)	0.5443 (0.0852)	0.5706 (0.0835)
SSL ($k = 2$)	0.5067 (0.1016)	0.6908 (0.0469)	0.5470 (0.1064)	0.5559 (0.0717)	0.5804 (0.0732)
SSL ($k = 5$)	0.5234 (0.1120)	0.7118 (0.0403)	0.5658 (0.1173)	0.5761 (0.0802)	0.6004 (0.0801)
TL-NPA ($k = 2$)	0.4934 (0.1158)	0.6925 (0.0392)	0.5316 (0.1220)	0.5443 (0.0852)	0.5706 (0.0835)
TL-NPA ($k = 5$)	0.4934 (0.1158)	0.6925 (0.0392)	0.5316 (0.1220)	0.5443 (0.0852)	0.5706 (0.0835)
TL-INA ($k = 2$)	0.4927 (0.1162)	0.6925 (0.0392)	0.5311 (0.1224)	0.5438 (0.0856)	0.5702 (0.0838)
TL-INA ($k = 5$)	0.4900 (0.1161)	0.6857 (0.0433)	0.5289 (0.1224)	0.5409 (0.0859)	0.5667 (0.0848)
MTL-NPA ($k = 2$)	0.4800 (0.1279)	0.6714 (0.0594)	0.5158 (0.1365)	0.5298 (0.1027)	0.5545 (0.1012)
MTL-NPA ($k = 5$)	0.6714 (0.0055)	0.7274 (0.0178)	0.7155 (0.0009)	0.6733 (0.0038)	0.6921 (0.0050)
MTL-INA ($k = 2$)	0.6660 (0.0130)	0.7295 (0.0177)	0.7112 (0.0060)	0.6691 (0.0095)	0.6889 (0.0076)
MTL-INA ($k = 5$)	**0.6727 (0.0144)**	**0.7446 (0.0066)**	**0.7160 (0.0107)**	**0.6774 (0.0117)**	**0.6981 (0.0082)**

*Table shows mean and (sd) in bracket

respectively. It can be observed from the table that accuracy drops by $\approx 30\%$ in case of low distribution while it drops by $\approx 11\%$ in case of high distribution. Figure 4.7 shows the result for accuracy performance by varying the parameter (b, nB)

It can be concluded from the hashing accuracy results that b-bit minwise hashing performs well in case of high sample DMOZ distribution dataset while in low distribution DMOZ dataset, LSH performs well.

4.5.2 Runtime Comparison

Table 4.11, 4.12 and 4.13 gives the runtime comparison for low and high distribution sampled DMOZ dataset after applying LSH (parameter, $K = 4000$), b-bit minwise hashing (parameters, $b = 4$, $nP = 500$) and one permutation hashing (parameters, $b = 5$, $nB = 500$) respectively. Clearly, in all the case STL runtime performance is best compared to all

Table 4.6: High distribution sample DMOZ average performance across five runs after LSH (parameter, $K = 4000$) for all models

Model	μAF_1	MAP	MAR	MAF_1	AMCC
STL	0.6826 (0.0199)	0.6963 (0.0244)	0.6855 (0.0192)	0.6732 (0.0194)	0.6642 (0.0213)
SSL ($k = 2$)	0.6810 (0.0210)	0.6961 (0.0250)	0.6839 (0.0204)	0.6711 (0.0206)	0.6626 (0.0224)
SSL ($k = 5$)	0.6777 (0.0227)	0.6941 (0.0282)	0.6807 (0.0219)	0.6673 (0.0228)	0.6593 (0.0248)
TL-NPA ($k = 2$)	0.6822 (0.0201)	0.6959 (0.0247)	0.6851 (0.0194)	0.6728 (0.0198)	0.6637 (0.0216)
TL-NPA ($k = 5$)	0.6823 (0.0200)	0.6960 (0.0246)	0.6853 (0.0193)	0.6729 (0.0197)	0.6639 (0.0215)
TL-INA ($k = 2$)	0.6823 (0.0200)	0.6960 (0.0246)	0.6853 (0.0193)	0.6729 (0.0197)	0.6639 (0.0215)
TL-INA ($k = 5$)	**0.6827 (0.0198)**	**0.6963 (0.0243)**	**0.6856 (0.0191)**	**0.6733 (0.0194)**	**0.6643 (0.0212)**
MTL-NPA ($k = 2$)	0.6810 (0.0203)	0.6946 (0.0250)	0.6840 (0.0195)	0.6713 (0.0202)	0.6624 (0.0220)
MTL-NPA ($k = 5$)	0.6802 (0.0208)	0.6944 (0.0252)	0.6832 (0.0200)	0.6703 (0.0208)	0.6616 (0.0224)
MTL-INA ($k = 2$)	0.6783 (0.0219)	0.6929 (0.0272)	0.6814 (0.0211)	0.6682 (0.0222)	0.6587 (0.0223)
MTL-INA ($k = 5$)	0.6758 (0.0235)	0.6922 (0.0291)	0.6789 (0.0227)	0.6644 (0.0241)	0.6568 (0.0258)

*Table shows mean and (sd) in bracket

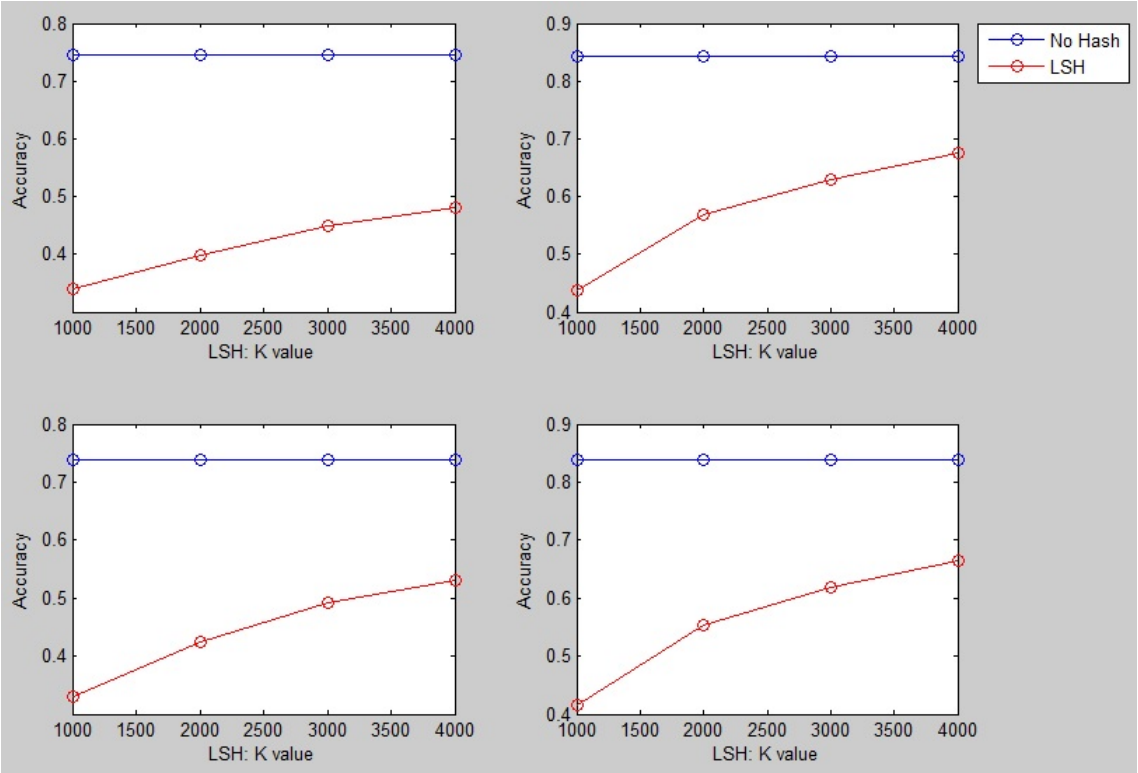


Figure 4.5: μAF_1 and MAF_1 graph after applying LSH for different values of parameter K, Topleft: LD μAF_1, Topright: HD μAF_1, Bottomleft: LD MAF_1, Bottomright: HD MAF_1

Table 4.7: Low distribution sample DMOZ average performance across five runs after b-bit minwise hashing (parameters, $b = 4, nP = 500$) for all models

Model	μAF_1	MAP	MAR	MAF_1	AMCC
STL	0.4653 (0.0522)	0.4499 (0.0729)	0.4393 (0.0426)	0.4279 (0.0729)	0.4578 (0.0094)
SSL ($k = 2$)	0.4660 (0.0931)	0.4346 (0.0063)	0.4544 (0.0529)	0.4412 (0.0625)	0.4660 (0.0195)
SSL ($k = 5$)	0.4547 (0.0437)	0.4319 (0.0600)	0.4684 (0.0421)	0.4394 (0.0382)	0.4560 (0.0083)
TL-NPA ($k = 2$)	0.4510 (0.0643)	0.4612 (0.0872)	0.4218 (0.0753)	0.4192 (0.0634)	0. 4319 (0.0436)
TL-NPA ($k = 5$)	0.4493 (0.0478)	0.4616 (0.0754)	0.4327 (0.0764)	0.4283 (0.0473)	0.4283 (0.0426)
TL-INA ($k = 2$)	0.4523 (0.0721)	0.4562 (0.0635)	0.4333 (0.0856)	0.4294 (0.0856)	0.4318 (0.0454)
TL-INA ($k = 5$)	0.4494 (0.0764)	0.4492 (0.0812)	0.4281 (0.0528)	0.4162 (0.0542)	0.4214 (0.0654)
MTL-NPA ($k = 2$)	**0.4700 (0.0051)**	**0.4844 (0.0428)**	**0.4713 (0.0092)**	**0.4691 (0.0301)**	**0.4780 (0.0062)**
MTL-NPA ($k = 5$)	0.4600 (0.0873)	0.4755 (0.0713)	0.4604 (0.0728)	0.4261 (0.0365)	0.4589 (0.0371)
MTL-INA ($k = 2$)	0.4360 (0.0642)	0.4783 (0.0542)	0.4570 (0.0753)	0.4497 (0.0209)	0.4301 (0.0264)
MTL-INA ($k = 5$)	0.4347 (0.0627)	0.4561 (0.0764)	0.4415 (0.0921)	0.4360 (0.0520)	0.4239 (0.0093)

*Table shows mean and (sd) in bracket

Table 4.8: High distribution sample DMOZ average performance across five runs after b-bit minwise hashing (parameters, $b = 4, nP = 500$) for all models

Model	μAF_1	MAP	MAR	MAF_1	AMCC
STL	0.7632 (0.0103)	0.7830 (0.0092)	0.7666 (0.0103)	0.7475 (0.0082)	0.7461 (0.0184)
SSL ($k = 2$)	0.7648 (0.0118)	0.7819 (0.0014)	0.7679 (0.0283)	0.7493 (0.0442)	0.7475 (0.0147)
SSL ($k = 5$)	0.7658 (0.0110)	0.7785 (0.0042)	0.7691 (0.0532)	0.7521 (0.0169)	0.7490 (0.0083)
TL-NPA ($k = 2$)	0.7595 (0.0318)	0.7714 (0.0092)	0.7616 (0.0175)	0.7555 (0.0732)	0.7492 (0.0992)
TL-NPA ($k = 5$)	0.7594 (0.0027)	0.7616 (0.0099)	0.7598 (0.0384)	0.7524 (0.0372)	0.7428 (0.0538)
TL-INA ($k = 2$)	0.7691 (0.0283)	0.7613 (0.0529)	0.7714 (0.0421)	0.7512 (0.0031)	0.7448 (0.0531)
TL-INA ($k = 5$)	0.7609 (0.0382)	0.7600 (0.0284)	0.7692 (0.062)	0.7497 (0.0106)	0.7434 (0.0714)
MTL-NPA ($k = 2$)	**0.7711 (0.0428)**	**0.7978 (0.0209)**	**0.7746 (0.0093)**	**0.7546 (0.0205)**	**0.7530 (0.0582)**
MTL-NPA ($k = 5$)	0.7700 (0.0301)	0.7912 (0.0072)	0.7734 (0.0092)	0.7528 (0.0140)	0.7523 (0.0081)
MTL-INA ($k = 2$)	0.7695 (0.0712)	0.7771 (0.0095)	0.7729 (0.0428)	0.7541 (0.0182)	0.7524 (0.0072)
MTL-INA ($k = 5$)	0.7663 (0.0071)	0.7737 (0.0158)	0.7696 (0.0092)	0.7519 (0.0099)	0.7497 (0.0105)

*Table shows mean and (sd) in bracket

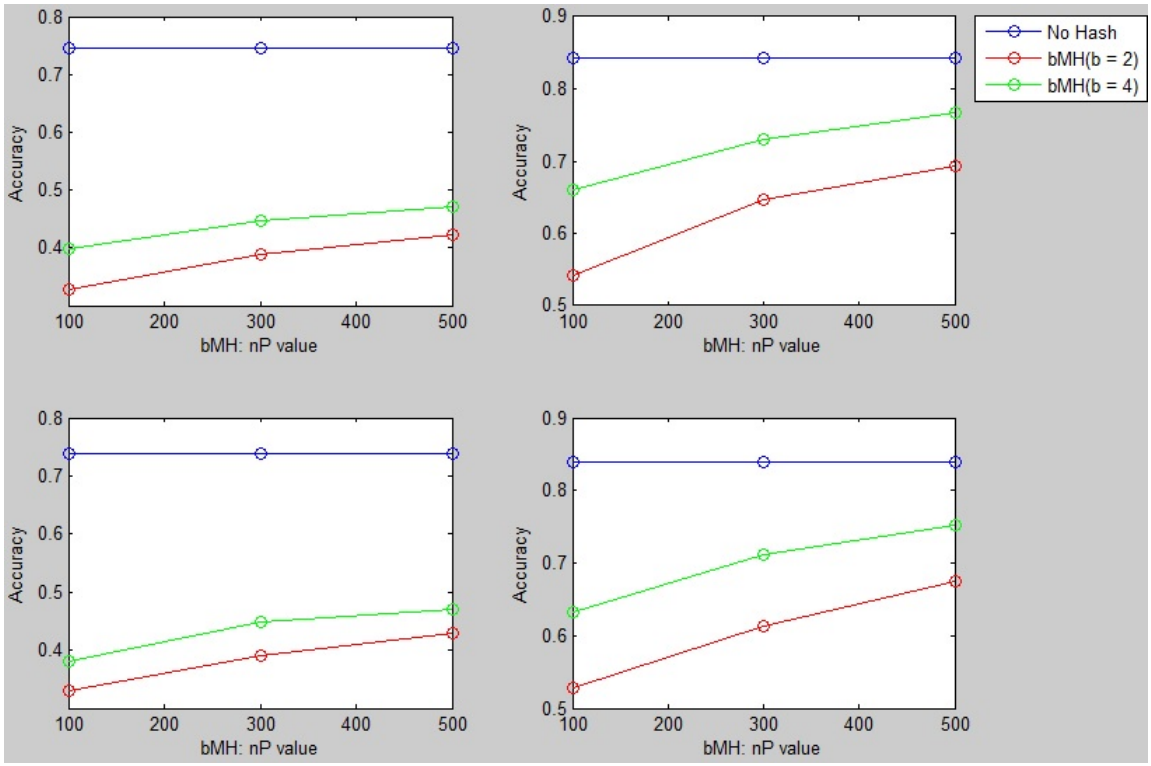


Figure 4.6: $\mu\text{A}F_1$ and $\text{MA}F_1$ graph after applying bMH for different values of parameter (b, nP), Topleft: LD $\mu\text{A}F_1$, Topright: HD $\mu\text{A}F_1$, Bottomleft: LD $\text{MA}F_1$, Bottomright: HD $\text{MA}F_1$

Table 4.9: Low distribution sample DMOZ average performance across five runs after OPH (parameters, $b = 4, nB = 500$) for all models

Model	μAF_1	MAP	MAR	MAF_1	AMCC
STL	0.4000 (0.0093)	0.4680 (0.0094)	0.4617 (0.0092)	0.382 (0.0098)	0.4124 (0.0091)
SSL ($k = 2$)	0.4124 (0.0061)	0.4680 (0.0071)	0.4528 (0.0059)	0.4196 (0.0062)	0.4324 (0.0063)
SSL ($k = 5$)	0.4073 (0.0095)	0.4736 (0.0074)	0.4287 (0.0062)	0.4314 (0.0070)	0.4383 (0.0068)
TL-NPA ($k = 2$)	0.4000 (0.0106)	0.4711 (0.0105)	0.4617 (0.0172)	0.3837 (0.0109)	0.4140 (0.0108)
TL-NPA ($k = 5$)	0.4000 (0.0105)	0.4717 (0.0108)	0.4617 (0.0102)	0.3841 (0.0094)	0.4144 (0.0131)
TL-INA ($k = 2$)	0.3973 (0.0105)	0.4710 (0.0129)	0.4595 (0.0102)	0.3822 (0.0104)	0.4126 (0.0103)
TL-INA ($k = 5$)	0.3947 (0.0108)	0.4623 (0.0946)	0.4578 (0.0103)	0.3798 (0.0091)	0.4093 (0.0103)
MTL-NPA ($k = 2$)	**0.4413 (0.0048)**	**0.4782 (0.0042)**	**0.4862 (0.0059)**	**0.4328 (0.0052)**	**0.4641 (0.0058)**
MTL-NPA ($k = 5$)	0.4368 (0.0093)	0.4714 (0.0058)	0.4261 (0.0060)	0.4108 (0.0058)	0.4312 (0.0050)
MTL-INA ($k = 2$)	0.4193 (0.0102)	0.4624 (0.0112)	0.4352 (0.0109)	0.4001 (0.0092)	0.4112 (0.0088)
MTL-INA ($k = 5$)	0.3947 (0.0083)	0.4350 (0.0063)	0.4048 (0.0059)	0.3964 (0.0079)	0.4047 (0.0069)

*Table shows mean and (sd) in bracket

Table 4.10: High distribution sample DMOZ average performance across five runs after OPH (parameter, $b = 4, nB = 500$) for all models

Model	μAF_1	MAP	MAR	MAF_1	AMCC
STL	0.7192 (0.0018)	0.7536 (0.0102)	0.7235 (0.0094)	0.6973 (0.0089)	0.7017 (0.0079)
SSL ($k = 2$)	0.7187 (0.0102)	0.7572 (0.0182)	0.7238 (0.0062)	0.6968 (0.0101)	0.7019 (0.0092)
SSL ($k = 5$)	0.7123 (0.0076)	0.7537 (0.0091)	0.7175 (0.0093)	0.6883 (0.0084)	0.6938 (0.0052)
TL-NPA ($k = 2$)	0.7192 (0.0103)	0.7532 (0.0062)	0.7237 (0.0092)	0.6975 (0.0102)	0.7017 (0.0120)
TL-NPA ($k = 5$)	0.7192 (0.0103)	0.7532 (0.0092)	0.7237 (0.0098)	0.6975 (0.0109)	0.7017 (0.0132)
TL-INA ($k = 2$)	0.7192 (0.0100)	0.7532 (0.0173)	0.7237 (0.0093)	0.6975 (0.0098)	0.7017 (0.0102)
TL-INA ($k = 5$)	0.7182 (0.0909)	0.7524 (0.0104)	0.7226 (0.0101)	0.6962 (0.0129)	0.7006 (0.0100)
MTL-NPA ($k = 2$)	0.7213 (0.0027)	0.7532 (0.0024)	0.7259 (0.0068)	0.7002 (0.0080)	0.7038 (0.0103)
MTL-NPA ($k = 5$)	0.7235 (0.0082)	0.7547 (0.0059)	0.7279 (0.0042)	0.7025 (0.0054)	0.7060 (0.0082)
MTL-INA ($k = 2$)	**0.7314 (0.0088)**	**0.7638 (0.0062)**	**0.7358 (0.0073)**	**0.7133 (0.0080)**	**0.7159 (0.0069)**
MTL-INA ($k = 5$)	0.7167 (0.0082)	0.7528 (0.0030)	0.7215 (0.0072)	0.6944 (0.0049)	0.6988 (0.0094)

*Table shows mean and (sd) in bracket

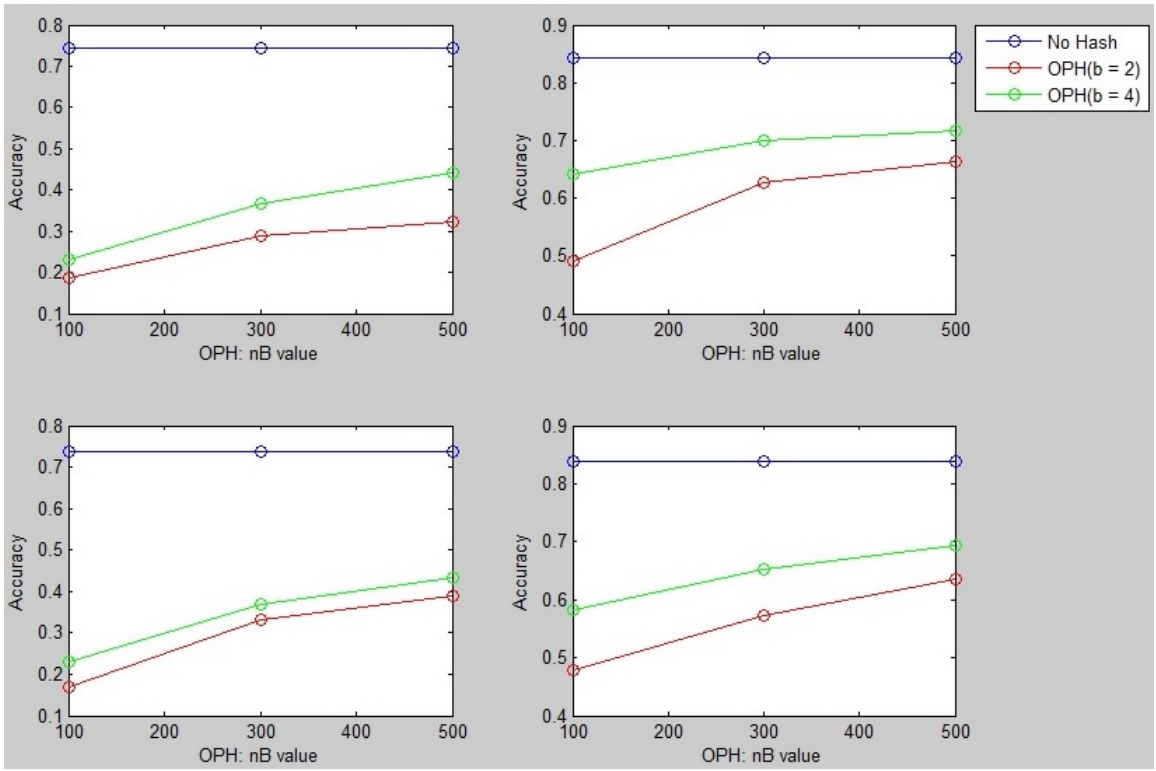


Figure 4.7: $\mu\text{A}F_1$ and $\text{MA}F_1$ graph after applying OPH for different values of parameter (b, nB), Topleft: LD $\mu\text{A}F_1$, Topright: HD $\mu\text{A}F_1$, Bottomleft: LD $\text{MA}F_1$, Bottomright: HD $\text{MA}F_1$

Table 4.11: Average runtime (in sec.) comparison of all models after applying LSH (parameter, $K = 4000$) for low and high distribution

Model	Low Distribution: Avg. Runtime(in sec)	High Distribution: Avg. Runtime (in sec)
STL	**2.2726**	**14.6553**
SSL ($k = 2$)	2.7668	20.2678
SSL ($k = 5$)	3.6739	20.9485
TL-NPA ($k = 2$)	3.0117	15.6856
TL-NPA ($k = 5$)	3.4435	17.8078
TL-INA ($k = 2$)	3.2147	15.7990
TL-INA ($k = 5$)	5.0216	18.4281
MTL-NPA ($k = 2$)	3.1034	15.2258
MTL-NPA ($k = 5$)	4.6382	15.2609
MTL-INA ($k = 2$)	6.7340	16.2862
MTL-INA ($k = 5$)	11.1680	18.1390

Preprocessing time: Low distribution $\approx$ 10 seconds, High distribution $\approx$ 40 seconds

the other models. When comparing the best runtime result of LSH with the best runtime no hashing models, there is an improvement of $\approx$ 0.5 sec per class in case of low distribution and $\approx$ 19 sec. per class in case of high distribution. Similar comparison in case of bMH gives an improvent of $\approx$ 0.1 sec. and $\approx$ 5 sec. in case of low and high distribution respectively. While for OPH results are $\approx$ 0.3 sec. and $\approx$ 7 sec. improvement for low and high distribution respectively.

Figure 4.8 and 4.9 shows the runtime comparison after applying LSH with different values of parameter K. Figure 4.10 and 4.11 shows the runtime comparison after applying bMH with different values of parameter (b, nP). Figure 4.12 and 4.13 shows the runtime comparison after applying OPH with different values of parameter (b, nB).

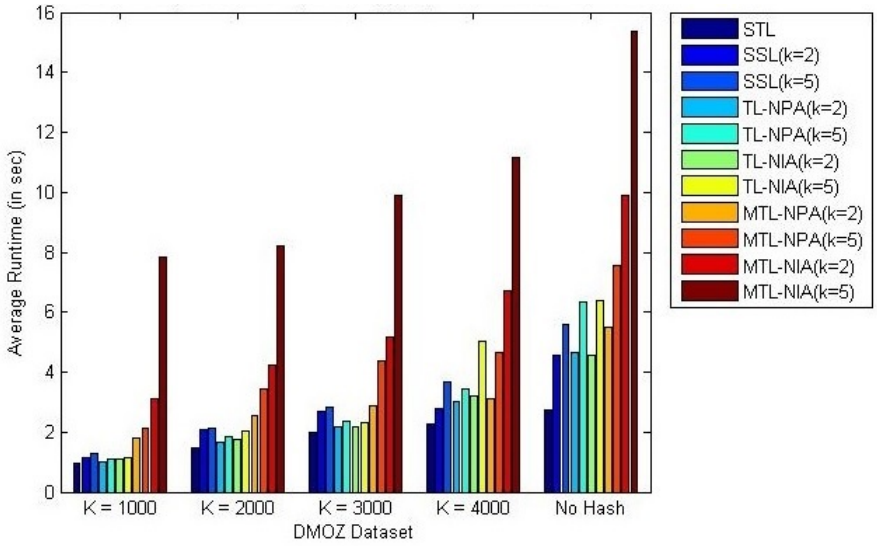


Figure 4.8: Average runtime (in sec.) comparison of all models after applying LSH for different values of parameter K for low distribution sample DMOZ dataset

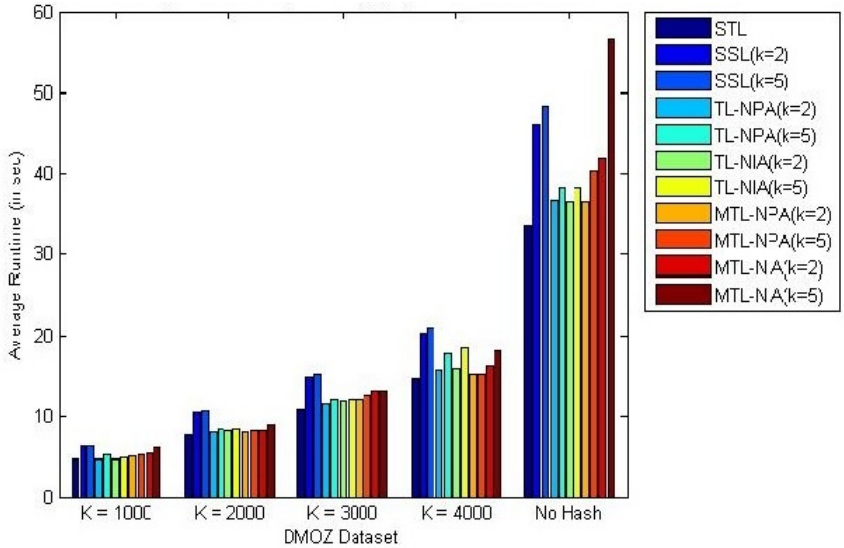


Figure 4.9: Average runtime (in sec.) comparison of all models after applying LSH for different values of parameter K for high distribution sample DMOZ dataset

Table 4.12: Average runtime (in sec.) comparison of all models after applying b-bit minwise hashing (parameters, $b = 4, nP = 500$) for low and high distribution

Model	Low Distribution: Avg. Runtime (in sec)	High Distribution: Avg. Runtime (in sec)
STL	**2.6074**	**28.0018**
SSL ($k = 2$)	3.1787	35.7084
SSL ($k = 5$)	3.8888	38.6752
TL-NPA ($k = 2$)	4.5597	30.0832
TL-NPA ($k = 5$)	5.9805	31.0087
TL-INA ($k = 2$)	4.3894	30.1320
TL-INA ($k = 5$)	6.0980	32.0916
MTL-NPA ($k = 2$)	3.9125	28.2946
MTL-NPA ($k = 5$)	4.4109	30.2840
MTL-INA ($k = 2$)	9.6608	30.7150
MTL-INA ($k = 5$)	13.6485	48.7282

Preprocessing time: Low distribution $\approx$ 3 minutes, High distribution $\approx$ 15 minutes

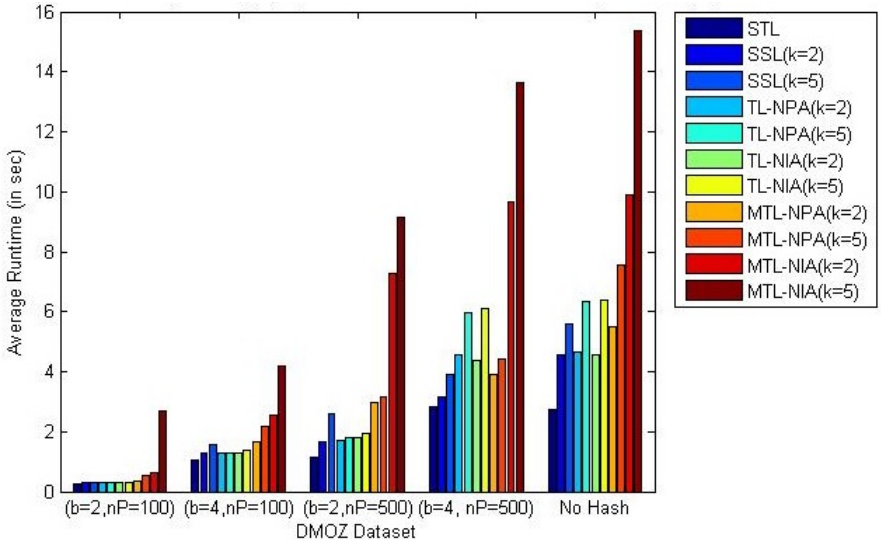


Figure 4.10: Average runtime (in sec.) comparison of all models after applying bMH for different values of parameter (b, nP) for low distribution sample DMOZ dataset

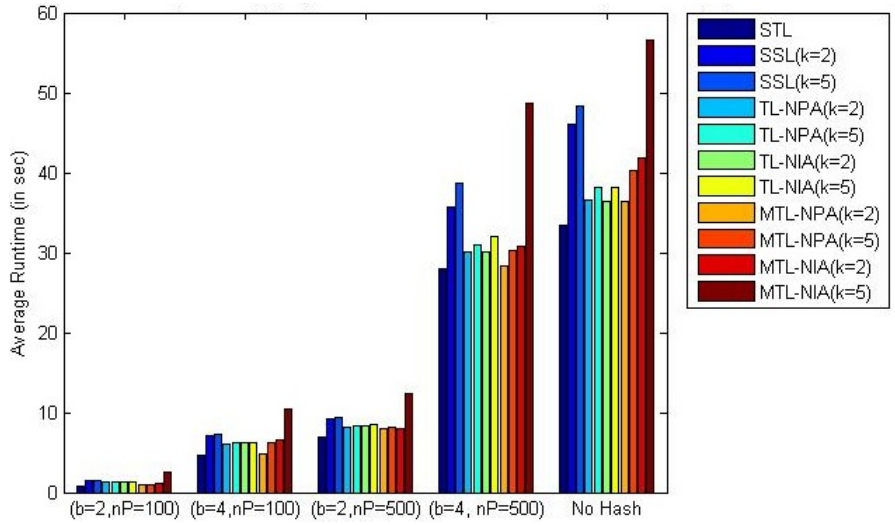


Figure 4.11: Average runtime (in sec.) comparison of all models after applying bMH for different values of parameter (b, nP) for high distribution sample DMOZ dataset

Table 4.13: Average runtime (in sec.) comparison of all models after applying one permutation hashing (parameters, $b = 4, nB = 500$) for low and high distribution

Model	Low Distribution: Avg. Runtime (in sec)	High Distribution: Avg. Runtime (in sec)
STL	**2.4774**	**27.9316**
SSL ($k = 2$)	3.1747	35.7084
SSL ($k = 5$)	3.8639	37.6752
TL-NPA ($k = 2$)	4.4597	29.1948
TL-NPA ($k = 5$)	5.9405	31.0087
TL-INA ($k = 2$)	4.3487	30.1320
TL-INA ($k = 5$)	5.9377	32.0916
MTL-NPA ($k = 2$)	3.8125	28.2946
MTL-NPA ($k = 5$)	4.4109	29.2640
MTL-INA ($k = 2$)	8.6608	29.6150
MTL-INA ($k = 5$)	11.6485	45.7282

Preprocessing time: Low distribution $\approx$ 1 minutes, High distribution $\approx$ 5 minutes

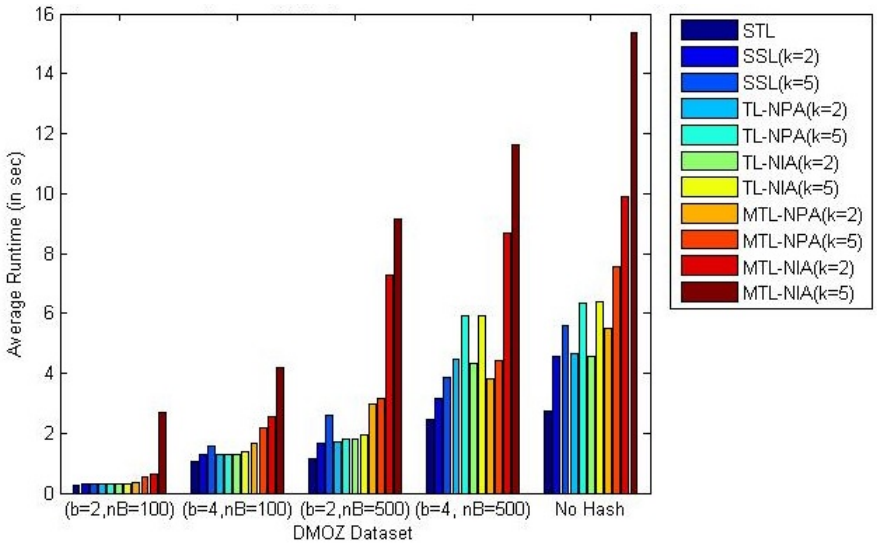


Figure 4.12: Average runtime (in sec.) comparison of all models after applying OPH for different values of parameter (b, nB) for low distribution sample DMOZ dataset

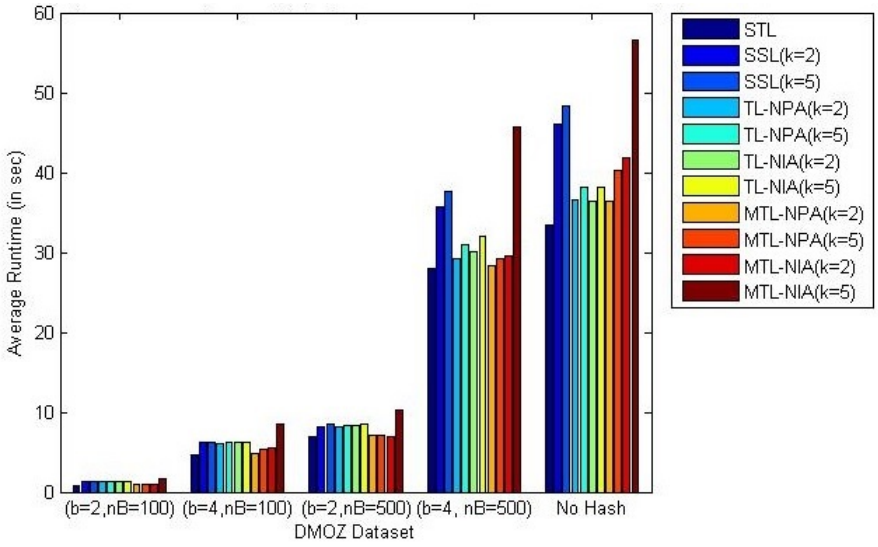


Figure 4.13: Average runtime (in sec.) comparison of all models after applying OPH for different values of parameter (b, nB) for high distribution sample DMOZ dataset

Chapter 5: Conclusion and Future Work

In this thesis MTL based models has been developed for text document classification. Performance of the developed MTL models has been evaluated and compared with other well known model development techniques, such as STL, SSL and TL. Performance measurement is done in terms of accuracy of classifying the documents and runtime for learning the parameters of the models. MTL learns the parameters of the models more accurately than other models because it captures the intrinsic relationship between the tasks by jointly learning the model parameters, especially when the number of training examples is less. However, when the number of training examples is huge, all the models discussed in Chapter 3 performs equally well in terms of accuracy due to the fact that as the number of training examples increases even the simpler models performs well. On comparing the runtime performance, STL model takes least time followed by SSL, TL and MTL. MTL method takes maximum time for parameter learning because all the related tasks model parameters needs to be updated in each of the iteration. Runtime performance can be improved by using random projections (hashing) but at the cost of reduced accuracy.

5.1 Future Work

Performance of the classifier can be further improved by the following methods discussed below,

5.1.1 Accuracy Improvement

Hierarchy contains important information related to text document classification. Hierarchy can be used to extract the parent-child relationships which can be used to improve the models parameters. However, this is achieved at the cost of expensive runtime, since

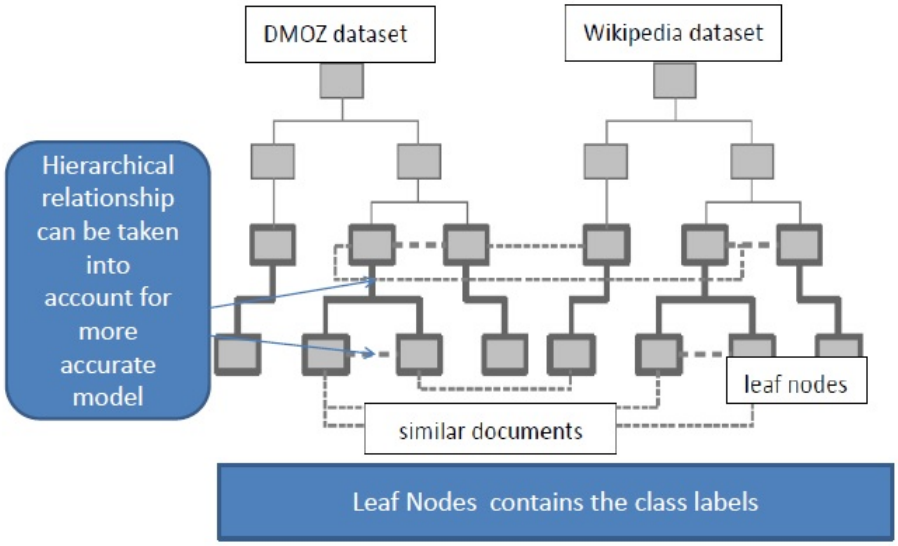


Figure 5.1: Hirerarchy relationship - for learning more accurate model

capturing the hierarchical information takes polynomial runtime. Figure 5.1 shows how we can use hierarchy for improving the model parameters.

5.1.2 Runtime Improvement

Runtime of learning the parameters of the models can be improved by writing the GPU based code. One of the greatest advantage of using the GPU is that it has huge number of threads which can be used to learn the model parameters independently in parallel thereby reducing the runtime of the parameters learning to huge fold (Figure 5.2). Further improvement can be done by implementing the accelerated gradient descent method for learning the parameters of the model.

5.1.3 Other Loss Function

As discussed in Chapter 3, in this thesis we have made use of logistic regression as the loss function for learning the model parameters. It is fruitful to compare the result with other loss function such as Hinge loss [49] and Square loss [50].

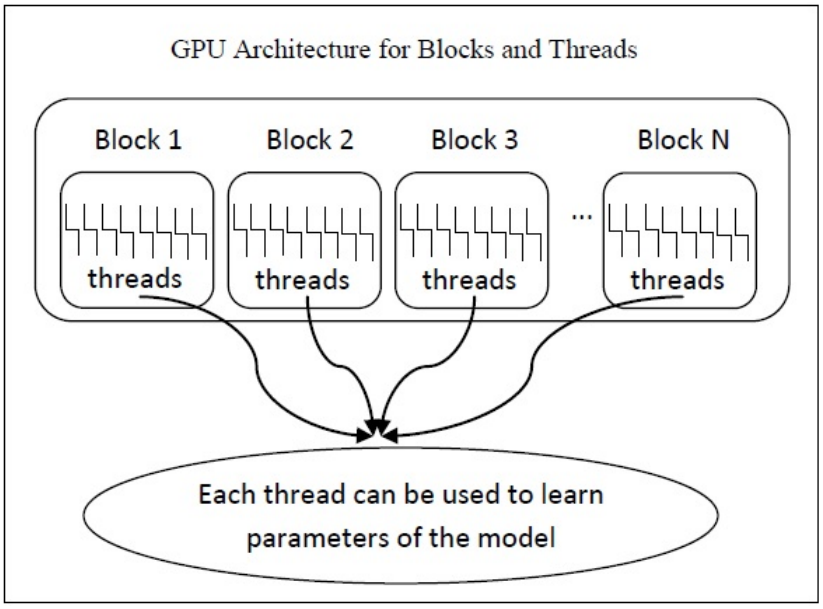


Figure 5.2: GPU Architecture for parallel parameters learning

Appendix A: GPU computation for kNN

Finding k nearest neighbor for each class of DMOZ dataset is very expensive processing in terms of runtime. To speed up the process of kNN calculation we have implemented the GPU based code. Linear implementation of kNN can be easily mapped to the GPU threads in following way. Each thread computes the nearest neighbor of each class of DMOZ dataset with all the classes of wikipedia dataset. If there are more threads than the number of classes of DMOZ than we can easily split the task between two or more threads.

A.1 Source Code for similarity matrix computation to be used in kNN computation

```
__global__ void kNNComputation(
    float *DMOZArray, /* Contains the representative DMOZ feature vector */
    float *WikiArray, /* Contains the representative Wikipedia feature vector */
    float *similarityMatrix, /* Stores the similarity value */
    float *denDMOZ, /* Contains the number of ones in DMOZ feature vector */
    float *denWiki, /* Contains the number of ones in Wikipedia feature vector */
    int *numClass, /* Number of DMOZ class */
    int *numDim)/* Dimension of each class */{
    int tid = blockIdx.x*blockDim.x + threadIdx.x;
    int N = numClass, d = numDim;
    float k;
    for(i=0; i<N; i++){
      similarityMatrix[tid*N+i] = 0.0;
      for(j=0; j<d; j++){
        similarityMatrix[tid*N+i] = DMOZArray[(tid)*d+j]*WikiArray[d*i+j] +
                                    similarityMatrix[tid*N+i];
```

```
        }
        k = denDMOZ[tid] + denWiki[i] - similarityMatrix[tid*N+i];
        if(k > 0){
          similarityMatrix[tid*N+i] = similarityMatrix[tid∗N+i]/k;
        }
      }
}
```

Bibliography

Bibliography

[1] E. Han and G. Karypis, "Centroid-based document classification: Analysis and experimental results," *Principles of Data Mining and Knowledge Discovery*, pp. 116–123, 2000.

[2] H. Borko and M. Bernick, "Automatic document classification," *Journal of the ACM (JACM)*, vol. 10(2), pp. 151–162, 1963.

[3] S. Dumais and H. Chen, "Hierarchical classification of web content," *Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval, ACM*, 2000.

[4] Q. Xipeng, X. Huang, Z. Liu, and J. Zhou, "Hierarchical text classification with latent concepts," *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics*, vol. 2, 2011.

[5] S. Gopal, Y. Yang, and A. Niculescu-Mizil, "A regularization framework for large scale hierarchical classification," *ECML*, 2012.

[6] J. Zhou, L. Yuan, J. Liu, and J. Ye, "A multi-task learning formulation for predicting disease progression," *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining, ACM*, 2011.

[7] A. Charuvaka and H. Rangwala, "Multi-task learning for classifying proteins with dual hierarchies." pp. 834–839, 12/2012 2012.

[8] C. Widmer, J. Leiva, Y. Altun, and G. Rätsch, "Leveraging sequence classification by taxonomy-based multitask learning," *14th Annual International Conference, RECOMB, Lisbon, Portugal*, pp. 522–534, April 25-28, 2010.

[9] X. Wang, C. Zhang, and Z. Zhang, "Boosted multi-task learning for face verification with applications to web image and video search," *IEEE conference on Computer Vision and Pattern Recognition*, pp. 142–149, 2009.

[10] J. Ghosn and Y. Bengio, "Multi-task learning for stock selection," *Advances in Neural Information Processing Systems*, pp. 946–952, 1997.

[11] J. Zhou, Y. Lei, J. Liu, and J. Ye, "A multi-task learning formulation for predicting disease progression," *In Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 814–822, 2011.

[12] S. Yu, K. Yu, V. Tresp, and H. Kriegel, "Collaborative ordinal regression," *In Proceedings of the 23rd international conference on Machine learning*, pp. 1089–1096, 2006.

[13] R. Caruana, "Multitask learning," *Machine Learning*, vol. 28(1), pp. 41–75, 1997.

[14] J. Baxter, "A model of inductive bias learning," *JAIR*, vol. 12, pp. 149–198, 2000.

[15] S. Thrun, "Is learning the n-th thing any easier than learning the first?" *NIPS*, pp. 640–646, 1996.

[16] S. Ben-David and R. Schuller, "Exploiting task relatedness for multiple task learning," *Learning Theory and Kernel Machines*, pp. 567–580, 2003.

[17] J. Zhou, J. Chen, and J. Ye, *MALSAR: Multi-tAsk Learning via StructurAl Regularization*, Arizona State University, 2011. [Online]. Available: http://www.public.asu.edu/ jye02/Software/MALSAR

[18] T. Evgeniou and M. Pontil, "Regularized multitask learning," *KDD*, pp. 109–117, 2004.

[19] T. Jebara, "Multitask sparsity via maximum entropy discrimination," *JMLR*, pp. 75–110, 2011.

[20] A. Argyriou, T. Evgeniou, and M. Pontil, "Convex multi-task feature learning," *Machine Learning*, vol. 73(3), pp. 243–272, 2008.

[21] J. Liu, S. Ji, and J. Ye, "Multi-task feature learning via efficient l 2, 1-norm minimization," *UIA*, pp. 339–348, 2009.

[22] G. Obozinski, B. Taskar, and M. Jordan, "Multi-task feature selection," *ICML*, 2006.

[23] R. Tibshirani, "Regression shrinkage and selection via the lasso," *Journal of the Royal Statistical Society. Series B (Methodological)*, pp. 267–288, 1996.

[24] A. Argyriou, T. Evgeniou, and M. Pontil, "Multi-task feature learning," *NIPS*, p. 19:41, 2007.

[25] T. Jebara, "Multi-task feature and kernel selection for svms," *ICML*, p. 55, 2004.

[26] T. Kato, H. Kashima, M. Sugiyama, and K. Asai, "Multi-task learning via conic programming," *NIPS*, pp. 737–744, 2008.

[27] T. Evgeniou, C. Micchelli, and M. Pontil, "Learning multiple tasks with kernel methods," *JMLR*, vol. 6(1), pp. 615–637, 2005.

[28] S. Thrun and J. O. Sullivan, "Clustering learning tasks and the selective cross-task transfer of knowledge," *Learning to learn*, pp. 181–209, 1998.

[29] L. Jacob, F. Bach, and J. Vert, "Clustered multi-task learning: A convex formulation," *NIPS*, 2008.

[30] J. Zhou, J. Chen, and J. Ye, "Clustered multi-task learning via alternating structure optimization," *NIPS*, 2011.

[31] E. Bonilla, K. Chai, and C. Williams, "Multi-task gaussian process prediction," *NIPS*, 20(October), 2008.

[32] Y. Zhang and D. Yeung, "A convex formulation for learning task relationships in multi-task learning," *In Proceedings of the Twenty-fourth Conference on Uncertainty in AI (UAI)*, 2010.

[33] S. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on Knowledge and Data Engineering*, vol. 22(10), pp. 1345–1359, 2010.

[34] X. Zhu, "Semi-supervised learning literature survey," *world*, vol. 10, p. 10, 2005.

[35] Z. Rasheed, H. Rangwala, and D. Barbará, "Efficient clustering of metagenomic sequences using locality sensitive hashing," in *SIAM International Conference in Data Mining*, Anaheim, CA, April 2012, pp. 1023–1034.

[36] P. Li, A. Shrivastava, J. Moore, and A. C. König, "Hashing algorithms for large-scale learning," *Technical Report*, 2011.

[37] P. Li, A. B. Owen, and C.-H. Zhang, "One permutation hashing for efficient search and learning," *CoRR*, vol. abs/1208.1259, 2012.

[38] A. Z. Broder, M. Charikar, A. M. Frieze, and M. Mitzenmacher, "Min-wise independent permutations," *Journal of Computer Systems and Sciences*, vol. 60(3), pp. 630–659, 2000.

[39] P. Li, A. Shrivastava, and C. A. Konig, "Gpu-based minwise hashing," *Proceedings of the 21st international conference companion on World Wide Web. ACM*, pp. 565–566, 2012.

[40] J. L. Carter and M. N. Wegman, "Universal classes of hash functions," *Journal of computer and system sciences*, vol. 18(2), pp. 143–154, 1979.

[41] N. Bhatia and Vandana, "Survey of nearest neighbor techniques," *International Journal of Computer Science and Information Security*, vol. 8(2), pp. 302–305, 2010.

[42] T. M. Cover and P. E. Hart, "Nearest neighbor pattern classification," *IEEE Trans. Inform. Theory*, vol. 13, pp. 21–27, Jan. 1967.

[43] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning.* New York, NY, USA: Springer Series in Statistics, 2001.

[44] T. M. Mitchell, *Machine Learning.* McGraw Hill series in computer science, 1997.

[45] Y. Yang, "An evaluation of statistical approaches to text categorization," *Information Retrieval*, vol. 1(1), pp. 69–90, 1999.

[46] D. Lewis, R. Schapire, J. Callan, and R. Papka, "Training algorithms for linear text classiers," *In Proceedings of the 19th annual international ACM SIGIR conference on research and development in information retrieval*, pp. 298–306, 1996.

[47] A. Özgür, L. Özgür, and T. Güngör, "Text categorization with class-based and corpus-based keyword selection," *Proceedings of the 20th international conference on Computer and Information Sciences*, pp. 606–615, 2005.

[48] P. Baldi, S. Brunak, Y. Chauvin, C. Andersen, and H. Nielsen, "Assessing the accuracy of prediction algorithms for classification: an overview," *Bioinformatics*, vol. 16(5), pp. 412–424, 2000.

[49] P. Bartlett and M. Wegkamp, "Classification with a reject option using a hinge loss," *The Journal of Machine Learning Research*, vol. 9, pp. 1823–1840, 2008.

[50] S. Yang and B. Hu, "A stagewise least square loss function for classification," *Proceedings of the 2008 SIAM International Conference on Data Mining*, 2008.

Curriculum Vitae

Azad Naik is currently a graduate student in the Department of Computer Science at the George Mason University (GMU) in the Northern Virginia Area. He is also a graduate teaching assistant in the Computer Science department at GMU, Fairfax, VA. Before joining GMU, he has worked as a Software Developer for 1 year at Aricent and as a Senior Software Developer for over 1 year at Samsung India Software Operations (SISO), Bangalore. He completed his Bachelor of Technology in Computer Science and Engineering from Indian School of Mines Dhanbad, in 2009.

His research interest can be broadly categorized as Data Mining, Machine learning, Algorithm Design, Statistical Pattern Recognition, High Performance Computation and Computational Biology.