Exploring Semi-Supervised Learning on Hypergraphs

Final Report

Chen Jiali 3035085695

Date of Submission: 16/4/2016
I. Abstract

Graph based learning is one of the most significant fields in semi-supervised learning. The existing methods mainly focus on objects with pairwise relationships, which can be illustrated as normal graphs. However, relationships among objects are always too complex for normal graphs to summarize. Based on the fact, hypergraph learning methods have become increasingly significant. Among the related methods, Hubert Chan provides us an effective and efficient approach based on directed hypergraph. The subgradient method, as a semi-supervised method, is able to produce a reasonable prediction with a tiny training set. Our project has implemented both Python version and c++ version for this method, with parallel computing to accelerate the learning process. During the project, experiments analyzing the accuracy as well as time complexity with respect to the traditional machine learning models have also been conducted. Furthermore, group members delivered generalized version of the method to deal with multi-classification problems. Moreover, sigmoid functions and smoothed ReLu functions are applied as activation functions in order to alleviate bias and overfitting. Gaussian function are selected as an alternative of the original quadratic regularization potential function. Experiments for the effects of the activation functions and altered regularization potential functions are conducted. In the future, further research can commence for other activation and potential functions. Experiments with our version of implement can be transplanted to the cloud template, in order to fully utilize the computation acceleration of parallel computing. Besides, currently the algorithm supports well only the categorical features, a further generalization for integer and real features may also be the future target.
II. Acknowledgment

We would like to express our deepest appreciation to all those who provided me the possibility to complete this report. A special gratitude we give to our supervisor Dr Chan, Hubert Chan and supervisor helper Mr Zhang, Zhang Chenzi. As we have very limited knowledge about machine learning, their suggestions and instructions provided us a number of shortcuts, helped us to complete this report.

Also, great amount of appreciation will be given to Dr Wu, Chuan Wu, our second examine, for her advices and comments on our project and presentations.

Furthermore I would also like to acknowledge with much appreciation the crucial role of the staff of Computer Science Department HKU, who gave the permission to use all required equipment and the necessary materials to complete the tasks. Last but not least, many thanks go to our CAES class instructor, Ken Ho. I have to appreciate the guidance given by him especially in our project presentation as well as report writing that has improved our skills.

Thanks and appreciations go to all of the individuals who have given precious comments and suggestions in developing the project.
III. Table of Contents

I. Abstract .............................................................................................................. 1

II. Acknowledgment .................................................................................................. 2

III. Table of Contents ................................................................................................ 3

IV. List of Figures ..................................................................................................... 4

V. List of Tables ........................................................................................................ 5

VI. Abbreviations .................................................................................................... 7

VII. Group Information ........................................................................................... 8

1. Introduction ......................................................................................................... 9

2. Literature Review ............................................................................................... 10
   2.1 Semi-supervised Learning ........................................................................... 10
   2.2 Hypergraph .................................................................................................. 11
   2.3 Apply Hypergraph to Semi-supervised Learning ........................................ 12
   2.4 True Hypergraph Approach (Hein’s method) ............................................... 12
   2.5 Subgradient Approach and Directed Hypergraph (Hubert’s method) .......... 13

3. Motivation ........................................................................................................... 14

4. Theoretical Framework ....................................................................................... 15
   4.1 General Logic ............................................................................................... 15
   4.2 Potential Improvement .................................................................................. 16
       4.2.1 Activation Function Application .......................................................... 17
       4.2.2 regularization potential function Alternatives ........................................ 20
   4.3 Generalization .............................................................................................. 21

5. Scope ................................................................................................................ 23
   5.1 Comparison Candidates ............................................................................... 23
       5.1.1 Machine Learning Algorithms Candidates .......................................... 23
       5.1.2 Activation Function Candidates ............................................................ 23
       5.1.3 regularization potential function Candidates ........................................... 23
       5.1.4 Parallel versus Nonparallel ................................................................... 23
   5.2 Data Source ................................................................................................... 24
       5.2.1 Data Repository ....................................................................................... 24
       5.2.2 Dataset Introduction .............................................................................. 25
       5.2.3 Dataset Difficulty Rate ........................................................................... 25

3
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.2.4 Dataset Selection</td>
<td>26</td>
</tr>
<tr>
<td>5.2.5 Training Set Selection</td>
<td>26</td>
</tr>
<tr>
<td><strong>6. Implementation</strong></td>
<td>27</td>
</tr>
<tr>
<td>6.1 Programming platform</td>
<td>27</td>
</tr>
<tr>
<td>6.1.1 Reason of choosing Python</td>
<td>27</td>
</tr>
<tr>
<td>6.1.2 Reason of choosing C++</td>
<td>28</td>
</tr>
<tr>
<td>6.2 Program Structure</td>
<td>28</td>
</tr>
<tr>
<td>6.2.1 Hypergraph Class</td>
<td>29</td>
</tr>
<tr>
<td>6.2.2 Subgradient Class</td>
<td>29</td>
</tr>
<tr>
<td>6.3 Program Techniques</td>
<td>30</td>
</tr>
<tr>
<td>6.4 Complexity of the Implementation</td>
<td>30</td>
</tr>
<tr>
<td><strong>7. Experiment and Result</strong></td>
<td>31</td>
</tr>
<tr>
<td>7.1 Parallel Computing v.s. Non Parallel Computing</td>
<td>31</td>
</tr>
<tr>
<td>7.2 Activation Functions Experiment</td>
<td>33</td>
</tr>
<tr>
<td>7.3 Regularization Potential Function Experiments</td>
<td>36</td>
</tr>
<tr>
<td>7.3.1 Accuracy Experiment</td>
<td>36</td>
</tr>
<tr>
<td>7.3.2 Convergence Speed Experiment</td>
<td>36</td>
</tr>
<tr>
<td>7.4 Learning Compared with Popular Machine Learning Models</td>
<td>37</td>
</tr>
<tr>
<td><strong>8. Distribution of Labor</strong></td>
<td>40</td>
</tr>
<tr>
<td><strong>9. Limitations and Difficulties</strong></td>
<td>42</td>
</tr>
<tr>
<td>9.1 Computation Ability of the Computer</td>
<td>43</td>
</tr>
<tr>
<td>9.2 Selection of Training Set</td>
<td>43</td>
</tr>
<tr>
<td>9.3 Selection of Categorical Multi-classification Dataset</td>
<td>43</td>
</tr>
<tr>
<td><strong>10. Future Work and Conclusion</strong></td>
<td>43</td>
</tr>
<tr>
<td>10.1 Future Work</td>
<td>44</td>
</tr>
<tr>
<td>10.1.1 Head and Tail Configuration</td>
<td>44</td>
</tr>
<tr>
<td>10.1.2 Experiment on the Cloud</td>
<td>44</td>
</tr>
<tr>
<td>10.1.3 Further Generalization</td>
<td>44</td>
</tr>
<tr>
<td>10.1.4 Further Research on Activation Function and Regularization Potential Function</td>
<td>45</td>
</tr>
<tr>
<td>10.2 Conclusion</td>
<td>45</td>
</tr>
</tbody>
</table>
IV. List of Figures

Figure 1: The learning scenario of semi-supervised learning algorithms. 10

Figure 2: Hypergraph versus normal graph. 11

Figure 3: Clique Approximation (left) versus True Hypergraph (right). 12

Figure 4: Directed hyperedge. 13

Figure 5: Subgradient iteration illustration (quadratic regularization potential function) 16

Figure 6: graph for rescaled sigmoid function. 17

Figure 7: graph for rescaled and translated softsign function. 18

Figure 8: graph for rescaled and translated softplus function. 18

Figure 9: graph for rescaled and translated ELU function. 19

Figure 10: Rescaled gaussian function and its gradient. 20

Figure 12, parallel versus nonparallel on mushroom, audiology and car evaluation datasets. 32

Figure 13: accuracy - train size table and graph for activation functions on mushroom dataset. 34

Figure 14: accuracy - train size table and graph for activation function experiments on car evaluation dataset. 35

Figure 15: accuracy - train size table and graph for potential functions on mushroom dataset. 36

Figure 16: convergence speed by quadratic versus gaussian. 37

Figure 18: Final result for car evaluation dataset. 39

Figure 19: final prediction result for nursery dataset. 40
V. List of Tables

Table 1: multi-class version mushrooms. 22

Table 2: Dataset selection. 26

Table 3: Training set size selected for each dataset. 27

Table 4: Table recording result for activation function experiments on the mushroom dataset. 34

Table 5: Table recording result for activation function experiments on the car evaluation dataset. 35

Table 6: Table recording result for potential function experiments on the mushroom dataset. 36

Table 7: Table recording result for final result of the mushroom dataset. 38

Table 8: Table recording result for final result of the car evaluation dataset. 39

Table 9: Table recording result for final result of the nursery dataset. 40

Table 10: tasks list and distribution of labor. 42
VI. Abbreviations

- **LR**: Logistic regression, a regression model generally evolve itself when absorbing input data. A traditional, efficient and popular learning model in the machine learning field.

- **SVM**: Support vector machine, is supervised learning model with associated learning algorithms that analyze data used for classification and regression analysis.

- **PCA**: Principal component analysis, a popular statistical method used in the machine learning field for feature extraction. A set of observations of possibly correlated variables is converted into a set of values of linearly uncorrelated variables called principal components. Within which an orthogonal transformation is utilized.

- **LDA**: Linear discriminant analysis, a popular method used in the field of machine learning to find a linear combination hence extraction of features that characterizes or separates different classes.

- **ELU function**: Exponential linear unit function, a smoothed version of ReLu function.

\[
f(\alpha, x) = \begin{cases} 
\alpha(e^x - 1) & \text{for } x < 0 \\
x & \text{for } x \geq 0 
\end{cases}
\]

- **ReLu function**: Rectified Linear Units, a function frequently used as activation function in machine learning models.

\[
f(x) = \begin{cases} 
0 & \text{for } x < 0 \\
x & \text{for } x \geq 0 
\end{cases}
\]

- **UCI**: University of California Irvine
## VII. Group Information

<table>
<thead>
<tr>
<th>Exploring Semi-Supervised Learning on Hypergraphs</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Project Website</strong></td>
</tr>
<tr>
<td><strong>Supervisor</strong></td>
</tr>
<tr>
<td><strong>Second Examiner</strong></td>
</tr>
<tr>
<td><strong>Group Members</strong></td>
</tr>
<tr>
<td><strong>Contact Email</strong></td>
</tr>
</tbody>
</table>
1. Introduction

Machine learning is the process giving the computer ability to imitate the learning behavior of human beings to acquire new knowledge or skills. In recent decades, based on the rapid development of both hardware and algorithms, machine learning has broadened its application and attracted increasing attention.

Semi-supervised learning, as a branch of machine learning approaches, combines idea of both supervised learning and unsupervised learning. Both labels and similarity between data will participate into the learning process. Hence, prediction result can be produced with much less training set and with labels.

Hypergraph modeling provides a more intuitive modeling of categorical data without information loss compared with normal graph. As using one-hot-encoding to pre-process categorical datasets, hypergraph modeling provides a template for machine learning models to learn on categorical datasets.

Subgradient method, delivered by our supervisor, Hubert Chan, is a more intuitive and effective hypergraph-based semi-supervised learning, compared with Hein’s approach delivered in 2013.

As there are no formal implementation of subgradient method and relatively complete analysis of Subgradient method, our project aims to implement Hubert’s method, analyze its performance on categorical datasets with different characteristics, with reference to traditional machine learning models such as LR, SVM, and so on.

Moreover, in most real world cases, number of possibilities of labels will be larger than two. In the original version, multi-classes will be processed as one versus others and accordingly classified one by one. This is not a true multi-classification approach and will influence the efficiency of the algorithm. Therefore, this project another target is to provide a generalized and extended version that can learn with multi-classification datasets.

With these two targets, this project provides:
1. Python implementation of Subgradient method with Numpy, Scikit-learn, Pandas
2. C++ implementation of Subgradient method with mllib and Armadillo
3. Programming techniques to accelerate the learning process
4. Analysis subgradient method with traditional machine learning model: LR, SVM, decision tree, random forest upon different sorts of categorical dataset.
5. Different approaches for multi-classification subgradient method and analysis and comparison between them.
2. Literature Review

2.1 Semi-supervised Learning

Absorbing idea from both supervised learning algorithms and unsupervised learning algorithms, semi-supervised learning algorithms produce an improvement by minimizing the training set. Training with a large size of labeled data, supervised learning models are accurate but annotation demanding. With insufficient labeled data, these models are not capable of providing a convincing result. While unsupervised learning algorithms require no labeled data but hence cannot producing a labeled result.

With the approach introduced in Figure 1, the training can be conducted with a small labeled dataset and a large unlabeled dataset combined. Accordingly, semi-supervised learning can give a sufficiently high accuracy with a small labeled dataset [5]. Since labeling data is usually expensive, time-consuming and highly demanding (e.g. Labeling species according to DNA sequence), semi-supervised learning has the capability of reducing human effort and cost.

![Figure 1: The learning scenario of semi-supervised learning algorithms. Similar data points are clustered into the same group, and then groups are labeled according to the labeled data in this group. This scenario can provide a solution with high accuracy using a small training set.](image-url)
2.2 Hypergraph

Traditionally, normal graphs are used for modeling cases with the premise to treat the relationship between data points as to be pair wise, or to utilize feature extraction techniques like PCA, LDA. Such modeling will possibly degenerate the dataset and hence suffers information loss and enhances the difficulty of understanding and visualization of the origin dataset. Therefore, hypergraph is introduced for modeling. As illustrated in Figure 2, the complicated relationships between mushrooms are expressed concisely with hypergraph modeling. To be specific, which of the mushrooms share which choice of which feature is definite. In contrast, information is lost in normal graph since an edge between two vertices only implies that these two mushrooms share one same feature but not what exactly the feature is, while hypergraph can provide detailed illustration [2]. Therefore learning on hypergraph can provide solutions to the cases with much more complicated relations.

Moreover, traditional machine learning models like LR, SVM function well only on numerical datasets. Categorical datasets are out of their capability if without external assistance. Hypergraph modeling uses one hot encoding to encode the categorical features, providing a template for these models to exhibit their learning ability on categorical datasets. To illustrate how one-hot encoding works, the example can be color feature in Figure 2 (left). Grey, brown, blue, the three choices of this feature will be encoded as [1 0 0], [0 1 0], [0 0 1]. With one-hot encoding, distance between any two choices of feature will be identical, which means categorical features are modeled as numerical feature without any unnecessary additional information.

Figure 2: Hypergraph (left) versus normal graph (right). Left: in hypergraph modeling, a hypergraph edge is a representation of a particular choice in a feature (e.g. grey, brown or blue in color feature). In this case 3 hyperedges are picked to illustrate. e1 represents grey mushrooms, e2 represents brown mushrooms while e3 represents mushrooms sharing China as their habitat. Right: in normal graph, an edge represents the existence of same choice of some feature between two mushrooms (e.g. edge between v1, v2 indicates that v1, v2 are both grey for color feature).
2.3 Apply Hypergraph to Semi-supervised Learning

As hypergraph can deliver a more explicit modeling of data, semi-supervised learning algorithms and hypergraph modeling are combined to broaden the application in real world machine learning tasks. Focus increases recently on the research of different mathematical approaches to such combination and on improving the efficiency of the implementation. For instance, D. Zhou, J. Huang, and B. Scholköpf introduced a method viewing the hyperedge as a clique (fully connected normal graph, explained in Figure 3 below (left)) of all the vertices and generalized clustering, classification and embedding to hypergraphs accordingly.[3]

2.4 True Hypergraph Approach (Hein’s method)

M.Hein, S.Setzer, L.Jost and S.S.Rangapuram produce an improvement motivated from the complaint about the efficiency of clique modeling. The team overcame the limitation with the assistance of a family of total variation based regularization functions on hypergraphs and a balanced graph cutting method. The time consuming normal-graph-clique construction process is hence avoided, as illustrated in Figure 3. By this way, they succeeded in the discovery of a purely hypergraph-based structural approach and decreasing the time complexity of the learning process as well [3].

Figure 3: Clique Approximation (left) versus True Hypergraph (right). Left: in the clique approximation, a clique is constructed on each hypergraph to figure out a minimum graph cut. All constructed normal edges will participate in the computation. Therefore, in the case of an edge containing n vertices (6 in this example), complexity will rise from n to n^2-n (30 in this example). Right: in Hein’s method, no clique construction and the original hypergraph will be the only participant in finding the balanced cut. Hence, complexity of n (6 in this example) remains.
2.5 Subgradient Approach and Directed Hypergraph (Hubert’s method)

Hubert Chan and his team, inspired from diffusion process on hypergraph, constructed a Markov operator that gives a subgradient of the current prediction during every iteration. Process subtracting current subgradient inside every iteration, imitating the normal gradient descent method, will converge the prediction to the critical point of the regularization potential function if it is convex. The prediction result will be accordingly given by the converged result. Besides, a framework on directed hypergraph was also delivered.

Directed hypergraph, illustrated in Figure 4, is a directional version of hypergraph. A set of the vertices in one edge is defined as head set while the other set of the vertices is tail set and the direction of this hyperedge is given from the tail set to the head set. Rely on directed hypergraph, more complicated relationships among vertices can be captured. For example, still consider the mushrooms dataset; assume there is the claim that mushrooms with identical cap-shape belongs more likely to the same class. The directed hypergraph can help to capture such feature. As is in Figure 4, hypothesis can be captured that the class of mushroom is unlikely to switch from “poisonous” to “edible”, along the direction of e3.

Detailed introduction to this method and how the group members improve and generalize the algorithm will be detailed discuss in Section 4.

![Directed hyperedge](image)

*Figure 4: Directed hyperedge. A hyperedge modified from e3 in Figure 1(middle). Recall that e3 is the hyperedge representing set of all mushrooms live in China. Further design v2, v3 have round cap, v4, v6 have triangle cap, and accordingly v2, v3 are defined as heads and v4, v6 are defined as tails, denoted separately by square and circle. The hyperedge now has a direction from \{v2, v3\} to \{v4, v6\}.*
3. Motivation

As discussed in section 2, semi-supervised learning algorithms and hypergraph are both effective and recently popular tools in the field of machine learning, therefore, it's valuable to do researches and obtain deeper understanding of these methods.

Moreover, currently there is a lacking in public accessible implementation of these semi-supervised learning algorithms. If the branch of semi-supervised learning on hypergraph modeling becomes maturely developed some time, our implementation under Python and c++ will be good prototype for the further development. During the implementation process, with assistance of programming technology such as parallel computing, whether and to what extent the learning efficiency increases also deserves studied.

Furthermore, as hypergraph provides the possibility for the traditional machine learning models to process learning on the categorical datasets, whether and how the subgradient method outperforms the traditional learning models is also research provoking, especially with tiny training sets. Experiments are essential to examine how efficient the subgradient method in real world cases to avoid labeling much data, which is expensive, time-consuming and highly-demanding.

Besides, plenty of alternatives of different aspects of a model exist and may lead to improvement of the final prediction result. For instance, different activation functions like sigmoid function, softplus function and so on can be used to map the prediction result to deal with frequent machine learning problems like over fitting and bias. Also, other convex function might be a replacement of the current quadratic function to change the speed of convergence. For example, a potential candidate in the machine learning field is Gaussian function. These alternatives are worthy of being explored and analyzed.

Last but not the least, a generalized version of subgradient method for multi-classification is also a necessity. As in many real world classification tasks, it is not as simple as black-and-white classification, the prediction result may be not practical enough if it is labeled only as true or false. How should the multi-classification subgradient method approach the result is another interesting objective of this function. Experiments should be conducted to figure out the performance of the generalized version compared with other classification models.

The above aspects trigger the project and the design of the experiments and exploration.
4. Theoretical Framework

4.1 General Logic
The first step of the subgradient method is to preprocess the input dataset and construct the H matrix correspondingly. On every feature, one hot encoding is applied as mentioned and illustrated in section 2.2. The encoded features of a data point are joined together as the row in the H matrix representing this data point. For instance, if some data point is characterized by three features and they are separately encoded as \([0 1 0]\) (second choice among three possibilities), \([0 1]\) (second choice between two possibilities), \([1 0 0]\) (first choice among four possibilities), then the edge in H matrix for this data point will be derived as \([0 1 0 0 1 1 0 0 0]\).

After the construction of the H matrix, the learning procedure will start from the initial states; here the initial states contain all the extreme possibilities of the labeling. For example, all the data points in the mushroom dataset belong to two classes, poisonous or edible. Accordingly, the two initial states start separately with two initial values: all mushrooms labeled as poisonous and all mushrooms labeled as edible. Entries for the training set in the initial states will be recovered to the given labels after the construction.

Start from each initial state, a number of iterations of the subgradient process will be conducted. The number varies from dataset to dataset, related to its features, class and dataset size. Generally, thousands of iterations are able to produce an enough acceptable result. The raw prediction result for the whole learning process before the thresholding step is derived from the all rough result iterated from different initial extreme possibilities. Method for this will be discussed in section 4.3.

Within the computation of each iteration mentioned in the paragraph above, subgradient of the regularization potential function of current prediction value will be computed via constructing the Markov operator. To make a more detailed and clearer explanation, we denote the regularization potential function for computing subgradient as \(L\), current prediction value as \(F\), and subgradient of this iteration \(G\). At the beginning of each iteration, \(G\) will be initialized as all-zero. Assume for some edge, we found the maximum of tail of this edge at i-th data point and \(F(i) = u\), while the minimum of head locates at j-th data point with \(F(j) = v\) (here i, j are the indices for these two data points among all, not for this edge). Then the learning obtained by this edge is given by:

\[
G(i) = G(i) + L(u - v).
\]

After the completion of the process in the paragraph above for every edge in the H
matrix, subgradient of this iteration is given by the G computed. The normalized subgradient \( \frac{G}{|G|} \) with a well-tuned coefficient will be subtracted from the current prediction result, i.e. \( f = f - \frac{G}{|G|} \), k currently used in the project is 0.9. After the subtraction, entries for the labeled data will be recovered again. After thousands of iterations of such procedure, the prediction value will converge at the critical point of the regularization potential function as it is chosen to be convex, as illustrated in figure 5.

![Subgradient iteration illustration](image)

**Figure 5: Subgradient iteration illustration (quadratic regularization potential function).** The two black points are the extreme cases in the initial state. F\(^+\) represents all labels being 1 (all mushrooms are poisonous), F\(^-\) represents all labels being 0 (all mushrooms are edible). Each arrow is the computation operated by an iteration (subtract the normalized subgradient). With sufficient iterations, both f\(^+\) and f\(^-\) will converge to be stabled. The final rough result comes from the average of the iterated f\(^+\) and f\(^-\).

After the raw prediction result being computed by the iterations of subgradient subtractions, it will be thresholded to be the discrete final prediction outcome.

Above is the general and kernel logic for the subgradient method. Based on the understanding of this whole procedure, group members have suggested their own potential improvement and generalization of this algorithm.

### 4.2 Potential Improvement

As in machine learning field, a large number of alternatives can be applied to different aspects of a machine learning field. For example, during the subgradient converging, many replacement for the origin quadratic function exist. Similarly, different sorts of activation functions can be applied to map the rough result for better thresholding and relief of bias and overfitting. Therefore, the improvement is consist of two subsections: activation function application and regularization potential function alternatives.
4.2.1 Activation Function Application

Inspired from neural network learning process, activation functions are applied by group members to deliver a preliminary mapping for the rough input derived from each initial case before the averaging procedure. Generally speaking, the major functionality of the activation function is conduct an injection from the original data without loss of the absolute distribution (i.e. larger results are still larger, smaller results are still smaller). While different sort of activation function has different handling style of the rough data and hence may have different effects on the subgradient learning model. Following is the introduction for the activation functions chosen. These functions are all translated and rescaled to fit the domain of the prediction value (the prediction outcome will be first mapped from [0, 1] to [-1, 1] for clearer illustration).

- Sigmoid function: \( a(x) = \frac{1}{1 + e^{-4x}} \)

![Graph for rescaled sigmoid function. Domain: \( x \in [-1, 1] \), codomain: \( a \in [0.0180, 0.9820] \). With \( x < 0 \), the derivative increase, and with \( x > 0 \), the derivative decreases. Hence value locate in the middle of the domain will be mapped more divergently while \( x \) near the two extremes of its domain will be injected convergently.](image-url)
• **Softsign function:** \( a(x) = \frac{1.5x}{1+|2x|} + 0.5 \)

Figure 7: graph for rescaled and translated softsign function. Domain: \( x \in [-1, 1] \), codomain: \( a \in [0, 1] \). Similar to sigmoid function illustrated in figure 6, the turning point of the convexity of softsign function is still at \( x=0 \). Therefore, the mapping characteristics for softsign function is similar to the sigmoid function.

• **Softplus function:** \( a(x) = 1.75\log(1 + e^{2x-1}) \)

Figure 8: graph for rescaled and translated softplus function. Domain: \( x \in [-1, 1] \), codomain: \( a \in [0.0369, 0.9981] \). Different from sigmoid and softsign function. Softplus function tends to enhance the prediction with value > 0 and discourage prediction with value < 0. That is to say, positive prediction will be encouraged while the effects of negative prediction will be decreased.
• ELU function: \( a(x) = 0.5e^x \text{ for } x < 0; \ 0.5x + 0.5 \text{ for } x > 0 \)

The four functions introduced above can be divided into classes, sigmoid functions and smoothed ReLu functions. The simplest sigmoid function and the softsign function belong to the first while softplus function and ELU function are smoothed ReLu functions.

For the sigmoid functions, as illustrated in figure 6 and figure 7, are characterized by their S-shape. X within the middle range of its domain, the intermediate part of “S”, will be mapped divergently, while x locates at the top or bottom of the “S” will be mapped convergently. Through this characteristic, classification with majority weak agreement will be more preferred compared to one extreme dominating prediction. For example, assume the dataset contains five classes and the learning procedures start from the five initial states produce the prediction of some data point to be class 1 as 1, 0.5, 0.5, 0.5, 0.5 separately and this data point classified to be class 2 as 0.59 0.59 0.59 0.59 0.59. Before the activation mapping, final prediction for this data point will be class 1 as 0.6 > 0.59. However the prediction result to class 1 might be biased as according to the learning mechanism introduced before, the extreme prediction may be the consequence of insufficient learning. With the activation function mentioned in section 4.2.1, prediction for class 1 will be mapped as 0.982 0.5 0.5 0.5 0.5 while injected prediction for class 2 is 0.672, 0.672, 0.672, 0.672, 0.672, the selection of the learning model is changed since 0.596<0.672 now. The result with majority agreement and hence less bias will be prefered by learning model with sigmoid activation function.

For the smoothed ReLu functions, gradient of these activation functions is monotone increasing on the domain of the function. Therefore, raw result with positive prediction will be injected more divergently and hence have an enhanced influence on the final prediction. It is believed by ReLu function the sparsity of the learning process. As proven effective in the learning of acoustic dataset by neural network[8]. It is possible
that ReLu function is able to improve the subgradient method as its ability of relieving overfitting. According to the learning logic introduced in section 4.1, the prediction value commence learning at 1, since the normalized subgradient is subtracted from the maximum of the current prediction. Thus more negative the prediction is, the more probable that the entry is overfitted. Therefore, with assistance of the ReLu function, influence of overfitted prediction value would be relieved.

4.2.2 regularization potential function Alternatives

The regularization potential function is the kernel part of the subgradient method. During the learning procedure in each iteration, normalized subgradient of the regularization potential function will be deducted. As illustrated and discussed in section 4.1, the regularization potential function decides the convergent speed of the algorithm as well as the understanding of the learning process.

Among the popular convex functions, group member selected rescaled gaussian function to be the alternative of the original quadratic regularization potential function.

- Rescaled Gaussian function:

\[-2.5e^{\frac{2}{3}(x-0.5)^2}\]

Gradient of the function:

\[1.5(x - 0.5)e^{\frac{2}{3}(x-0.5)^2}\]

Figure 10: Rescaled gaussian function and its gradient. Left: Rescaled gaussian function, domain of the function will be [0, 1]. Right: Gradient of the rescaled function, domain is [0, 1].
As illustrated in figure 10, gaussian function is another convex function on the domain of the prediction values. As compared in the right graph of the figure, gradient of the rescaled gaussian is larger than one of quadratic function on $[0.5, 1]$ and smaller on $[0, 0.5]$. Through this characteristic, the Gaussian regularization will converge faster. Moreover, as a rescaled distribution function, this function mimics the information entropy and may produce higher accuracy accordingly.

### 4.3 Generalization

The original subgradient function is effective and efficient only when learning on bi-classification dataset, however, in many real world cases, classification is not as simple as black and white classification, take the mushrooms as an example, learning model just classify them into poisonous or edible may not be powerful enough in practical application. One solution for adjusting current learning model to multi-classification problems is classify the data one class by another. Such approach provides a solution to the multi-classification problem, but the efficiency may hardly reach the demand, as the same process will be duplicated for the number of classes of times.

Inspired from the one-hot encoding applied in the construction of H matrix, group members derived a generalized version of the subgradient function. The multi-classification subgradient method commences from **one-hot encoding not only on features but also on labels**, as illustrated in table 1, labels of the data points in multi-classification dataset will be enumerated and accordingly one-hot encoded. With this operation, the label vector will be injected to be a label matrix, with each row represents all class prediction for a particular data point and each column for the prediction of all data points about a particular class. Hence, procedure of the original subgradient method can be operated separately on each column, i.e., each class of the label matrix. Each class itself will be treated as a bi-classification of yes or no. With this mechanism, the initial states of the learning process for the data points in table 1 will be 4 $6 \times 4$ matrices:

$$
\begin{bmatrix}
1 & 0 & 0 & 0; & 1 & 0 & 0 & 0; & 1 & 0 & 0 & 0; & 1 & 0 & 0 & 0; & 1 & 0 & 0 & 0; & 1 & 0 & 0 & 0; \\
0 & 1 & 0 & 0; & 0 & 1 & 0 & 0; & 0 & 1 & 0 & 0; & 0 & 1 & 0 & 0; & 0 & 1 & 0 & 0; & 0 & 1 & 0 & 0; \\
0 & 0 & 1 & 0; & 0 & 0 & 1 & 0; & 0 & 0 & 1 & 0; & 0 & 0 & 1 & 0; & 0 & 0 & 1 & 0; & 0 & 0 & 1 & 0; \\
0 & 0 & 0 & 1; & 0 & 0 & 0 & 1; & 0 & 0 & 0 & 1; & 0 & 0 & 0 & 1; & 0 & 0 & 0 & 1; & 0 & 0 & 0 & 1; \\
\end{bmatrix}.
$$
Table 1: multi-class version mushrooms. As a modified version from figure 2(left), these mushrooms now belong to four classes, edible to human, edible to animal, edible to all and poisonous. Each class will first be enumerated, as edible to human => 1, edible to animal => 2, edible => 3, poisonous => 4, and accordingly, one-hot encoded will be applied on the enumerated labels. As 1 => [1 0 0 0], 2 => [0 1 0 0], 3 => [0 0 1 0], 4=> [0 0 0 1].

<table>
<thead>
<tr>
<th>Mushroom</th>
<th>Color</th>
<th>Habitat</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>grey</td>
<td>Brazil</td>
<td>Edible to human =&gt;1 =&gt; [1 0 0 0]</td>
</tr>
<tr>
<td>2</td>
<td>grey</td>
<td>China</td>
<td>Edible to animal =&gt; 2 =&gt; [0 1 0 0]</td>
</tr>
<tr>
<td>3</td>
<td>brown</td>
<td>China</td>
<td>Edible to all =&gt; 3 =&gt; [0 0 1 0]</td>
</tr>
<tr>
<td>4</td>
<td>brown</td>
<td>China</td>
<td>Edible to human =&gt; 1 =&gt; [1 0 0 1]</td>
</tr>
<tr>
<td>5</td>
<td>blue</td>
<td>Brazil</td>
<td>Poisonous =&gt; 4 =&gt; [0 0 0 1]</td>
</tr>
<tr>
<td>6</td>
<td>blue</td>
<td>China</td>
<td>Poisonous =&gt; 4 =&gt; [0 0 0 1]</td>
</tr>
</tbody>
</table>

Moreover, there are no critical sections during the learning procedure operated on each class until the final merge prediction after all these learning procedures completed. Therefore, parallel computing techniques can be applied in the implementation of the generalized algorithm with one thread deal with one class. Ideally, if the processor’s parallel computing ability is sufficient, the time complexity for multi-classification should be identical to bi-classification.

After the achievement of convergence of learning process in every thread, activation function, as introduced and discussed in section 4.2.1, will be used to relieve the influence of insufficient learning as well as over-fitting.

Absorbing the idea of **majority voting**, the final prediction value merges all the results from every initial case similarly. The first step of the merging procedure is to take average of the results. Besides the activation function, taking average is another methodology to relieve the bias caused by insufficient learning or over learning. After achieving the raw data, we iterate for each data point, class with highest prediction value is the final classification of this data point.
5. Scope

5.1 Comparison Candidates

5.1.1 Machine Learning Algorithms Candidates
   a. Logistic regression
   b. Rbf Kernel SVM
   c. Random forest
   d. KNN
   e. Naive Bayes
   f. Multi-classification Subgradient method

This project only concentrates on the performance of the subgradient method, compared with traditional and popular machine learning classification models: LR, SVM, decision tree, random forest and naive Bayes. Subgradient method, as a hypergraph based semi-supervised classification method, is capable of achieving acceptable prediction result with tiny labeled data. Among the other candidates, decision tree and random forest can handle categorical data well, while the others require hypergraph modeled data.

5.1.2 Activation Function Candidates
   a. No activation function (identity function)
   b. Sigmoid function
   c. Softplus function
   d. Softsign function
   e. ELU function

5.1.3 regularization potential function Candidates
   a. Quadratic function
   b. Gaussian function

5.1.4 Parallel versus Nonparallel
Due to the generalization mechanism introduced in 4.3, there is no critical section between the learning procedures start from different initial state. Hence the implementation of the algorithm can be parallel, with one thread handling one iterative process start from one initial state. Ideally running with multi-thread can increase
greatly the program’s efficiency. However, there are overheads when applying multi-thread programming. Creating and joining threads, as well as switch context from thread to thread will cause extra time consumption. Moreover, because of the reason that the distribution ability of the program is fully decided by the number of possibilities of the dataset, the program is far not parallel enough (usually less than 20) to fully utilize the distributed computation ability of a GPU (usually more than 100 cores). Therefore, the program is better executed by CPU and hence if the parallel level exceeds the number of CPUs in the computer (larger than 4 for one used for this project), large amount of switch-thread overhead will be the consequence. Because of the reason above, experiments should be conducted to examine under what level of parallelism will the parallel computing version outperform.

5.2 Data Source

5.2.1 Data Repository

When conducting the implementation, datasets are selected from the UCI Machine Learning Repository (http://archive.ics.uci.edu/ml), which is a free online collection of datasets, whose home page is shown in Figure 5.

David Aha and fellow graduate students at UC Irvine created this repository in 1987. Since then, the repository has been a primary source of machine learning datasets for
projects worker on by students, educators, and researchers from all over the world. Furthermore, as most of the papers studied by our team during the literature review process adopted datasets from UCI Machine Learning Repository, we are quite familiar with this repository, which will make the testing experiment proceed more smoothly.

### 5.2.2 Dataset Introduction

All the datasets used by this project are retrieved from UCI machine learning repository for convenience (Reasons in detail have been explained above). The criteria of the data selection is whether the this dataset is categorical, as learning categorical data is a trouble for most of the traditional machine learning models and characterized by subgradient method on hypergraph and whether the dataset is classification dataset, as clustering problem is not in the scope. Datasets with different difficulty are chosen, and “difficulty” of a dataset will be defined in the next subsection.

### 5.2.3 Dataset Difficulty Rate

Different dataset varies in their number of features, choices of features and dataset size. These distinction decides the performance of subgradient model as well as the number of iterations to converge. Therefore, a dataset difficulty is defined referred to the subgradient algorithm so as to provide an overview before the learning process.

Firstly, difficulty within an iteration should be defined. Based on the theoretic framework in section 4, the learning rate in each iteration positively correlates to the dataset size $n$ as well as the number of edges $e$, i.e. the total choices of features. Number of edges decides the amount of learning process an iteration can provide while the dataset size shows on average how many data points will contribute into one learning process. Therefore, the information provided by a dataset is proportional to the product of dataset size $n$ and number of edges $e$. On the other hand, product of number of classes $c$ and dataset size $n$ decides how much information should the learning procedure provides. Thus in a particular iteration, the learning difficulty is defined as:

$$\frac{n \times c}{n \times e}$$

Afterwards, it should be figured out that for a particular dataset, how many iterations can be conducted within a fixed period. In each iteration, the computation complexity is $O(nce)$, the detailed explanation will be in Section 6. With larger product of class number and dataset size, longer time will be consumed in each iteration, and hence less iterations of learning will be operated during the same amount of time. Therefore, the difficulty of a dataset should also be proportional to the product of $cn$.

Conclusively, the difficulty rate of a dataset is defined by:

$$\frac{n \times c}{n \times e} \times nce = nc^2$$
5.2.4 Dataset Selection

Four datasets have been selected by group members, which are mushroom, car evaluation, nursery and audiology. The properties of the datasets are demonstrated in the table below. With relatively small dataset size and proper number of classes, mushroom dataset and car evaluation dataset can be learnt in a moderate period of time. Since experiments for activation functions, regularization potential function alternatives as well as parallel-nonparallel comparison requires a large number of duplicate executions, mushroom and car evaluation are selected to be the examinees of this part. Audiology dataset is also an examinee of parallel versus nonparallel comparison as it is characterized by its 25 classes. Mushroom, car evaluation and nursery dataset are the participants for the final performance analysis.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Dataset Size</th>
<th>No. of Edges</th>
<th>No. of Classes</th>
<th>Difficulty Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>mushroom</td>
<td>8124</td>
<td>119</td>
<td>2</td>
<td>3.2 * 10^4</td>
</tr>
<tr>
<td>car evaluation</td>
<td>1728</td>
<td>22</td>
<td>4</td>
<td>2.8 * 10^4</td>
</tr>
<tr>
<td>nursery</td>
<td>12960</td>
<td>28</td>
<td>5</td>
<td>3.2 * 10^5</td>
</tr>
<tr>
<td>audiology</td>
<td>226</td>
<td>388</td>
<td>25</td>
<td>1.4 * 10^5</td>
</tr>
</tbody>
</table>

Table 2: Dataset selection. Mushroom dataset is the most basic dataset and group members are sufficiently familiar with the prediction result of it. Car evaluation is a multi-classification dataset with similar difficulty compared with mushroom. Nursery dataset is a dataset with high difficulty as it should classify for 12960 datasets with 5 classes. Audiology is a relatively small dataset, however, it is characterized by its large number of classes and edges. It is a small but difficult dataset.

5.2.5 Training Set Selection

For different dataset, the number of training set varies. The selection of training data point in this project is demonstrated as table 3. The size of training set is much smaller than one used in the traditional classification models.
<table>
<thead>
<tr>
<th>Dataset</th>
<th>Dataset Size</th>
<th>Start Size</th>
<th>End Size</th>
<th>Fold Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>mushroom</td>
<td>8124</td>
<td>20</td>
<td>200</td>
<td>20</td>
</tr>
<tr>
<td>car evaluation</td>
<td>1728</td>
<td>10</td>
<td>100</td>
<td>10</td>
</tr>
<tr>
<td>nursery</td>
<td>12960</td>
<td>500</td>
<td>2500</td>
<td>500</td>
</tr>
<tr>
<td>audiology</td>
<td>226</td>
<td>20</td>
<td>100</td>
<td>20</td>
</tr>
</tbody>
</table>

Table 3: Training set size selected for each dataset. Start size is the size of smallest training set while the end size is the size for the largest. Fold size is the size of interval between different selection of training set for a particular dataset. For example, for nursery dataset, the training set picked will be 500, 1000, 1500 and 2000.

6. Implementation

A large amount of skill and labor is spent during the implementation of the program. It is insufficient to have only understanding on the subgradient method itself. Knowledge about the programming language Python and C++ as well as understanding of the packages like Scikit-learn in Python and armadillo in C++ are also necessities for the implementation. In this section, programming platform and packages used, program structure, programming techniques as well as complexity analysis will be the four subsections.

6.1 Programing platform

Within the whole project, two platforms have been chosen for the implementation and experiments, Python with SciPy, sklearn, NumPy, pandas and C++ with armadillo and mlpack.

6.1.1 Reason of choosing Python

Python is chosen as the programming language to implement and analyze the methods in the first half of the project. In the designing process, both Python and Matlab are popular in the machine learning field since both of them are very convenient in performing numerical computation and plotting. They both have a number of powerful supporting libraries like Statistics and Machine Learning Toolbox™ for Matlab, and SciPy, NumPy, sklearn for Python. Ultimately, Python is selected for the reason as follows:

a. Python is open source, which means it is free. Normal users as us are allowed to
modify the source.

b. Python is more coder friendly. Python is well designed with useful functions are provided like map, concat etc., popular data structures as set, matrix, array and powerful build-in functions for them. These will simplify our code and shorten the coding time.

c. Python has powerful machine learning modules like sklearn, which contains a large number of written machines learning algorithms. Modules can be called within only a few lines. Therefore, it’s convenient to call these APIs as reference for the experiments for Subgradient method.

6.1.2 Reason of choosing C++
Despite the convenience provided by Python, group members decided to translate the python version implementation into C++ after analyzing the advantages and disadvantages of C++ with Armadillo and mlpack compared with Python.

Advantages:

a. As a compiling language, C++ is tens of times faster than Python, as an interpreting language. With the improved speed, more experiments can be done thus more thoughts can be tested.

b. Although there exist modules like Theano and Cython supporting python language to be compiled and executed with C compiler, the modules have their own grammar and hence the reconstruction of the implementation can not be avoided. Moreover, logic supported by these modules are too simple to handle the complicated logic of the implementation neatly. Therefore, reconstruct the algorithm with C++ leads to no additional work.

Disadvantages:

a. Install manager apt-get in C++ is less powerful than anaconda or pip in Python, therefore the installation of the necessary packages and figuring out the method for compiling the program with these packages caused extra work for the group members.

b. Programming with C++ requires notion of memory management, as machine learning algorithms will usually occupy a large piece of memory when executing.

c. Armadillo and mlpack are not as powerful as the modules in Python. Therefore, many functions provided by Python APIs should be implemented for the C++ version.

6.1.2 Translation between Platforms
Held the belief that with sufficient programming experience and skills, the disadvantages for the reconstruction by C++ is handleable, group members translated the implementation with Armadillo and mlpack packages in C++ during the second half of the project. After the translation, time spent by a single execution has been accelerated by approximately 10 times.
6.2 Program Structure

The core part of the implementation consists of five files, main.cpp, hypergraph.cpp, hypergraph.h, subgradient.cpp, subgradient.h. The hypergraph class provides interfaces for the hypergraph modeling from a given dataset while the subgradient class offers fit-predict function for the subgradient method.

6.2.1 Hypergraph Class

Within the hypergraph class, constructHMat function is the core logic for the construction of an hypergraph. Given the filename for a csv file, the function reads the dataset and construct an instance of the Hypergraph class corresponds to the dataset. hMat, lMat, head and tail are arma::Mat<unsigned int> type for hypergraph matrix, label matrix, head matrix and tail matrix separately. Currently, head and tail are setted to be all the data points.

As mlpack package does not provides API for one-hot encoding, group members implemented the one-hot encoding function. The function takes the numeric categorical matrix and one-hot encodes the matrix as introduced in section 4.1 and 4.3.

The shuffle function is used to generate randomness of the dataset, current time will be the seed of shuffling.

The constructHMat function is visible by users of the function while one-hot encoding function and shuffle function are private.

6.2.2 Subgradient Class

fitPredict function is the major function for the multi-classification subgradient method. With the constructed Hypergraph of the dataset, the algorithm produce the learning. The fitPredict is also charge of managing the worker threads. It will wait until all the worker threads complete their work and merge the raw prediction results together. The final result will be recorded in the log directory and accuracy value.

ComputeDelta is the function for the computation of the subgradient. Within each iteration, this function is called for the G function mentioned in section 4.1.

Sgm function is the interface for the iterative procedure of a dataset. The function will be provided by the index of the task (i.e. class number), accordingly generates the initial state and operates the learning procedure. Moreover, the sgm function is the function executed by the worker threads and hence is the core part of the parallel computing.
RecoverF function is responsible for the recovery the labeled entries to their original values, as they are training set and slight modification on these entries in each iteration should be recovered.

The actFunc function and lossGradient function are functions for the selection of activation and alternatives of the regularizer potential function. These two function should be provided a character to decide the activation or potential function to use for the learning process.

The evalAcc function is charged of analyzing the accuracy of the final prediction.

### 6.3 Program Techniques

There are two things worth mentioned in our implementation. Firstly, there are no critical sections in the iterations of the sgm function. The only part that will probably lead to racing state is the merging of the final raw result in this working thread. Therefore, the majority of the sgm logic can be executed parallelly within the worker threads. The improvement of the parallel computing will be discussed and illustrated in section 7, the experiments. Secondly, due to the sparsity of the learning procedure, the matrix operation is optimized. Theoretically, the G matrix should be computed through the product of two matrix. However, as usually the number of edge $e$ is much smaller than the number of dataset size $n$, direction matrix multiplication will cause a large amount of waste. Therefore, group members optimized by an alternative as suggested by our supervisor that only the changed part will be take into consideration.

### 6.4 Complexity of the Implementation

Symbol denoting:

- Number of edge: $e$
- Dataset size: $n$
- Number of classes: $c$
- Number of iterations: $k$

To analyze the time complexity, there are two parts to be considered. The first phase taken into consideration is the construction of the hypergraph object. Afterwards, the core subgradient learning is target for analyzing.

During the construction of the Hypergraph object for the dataset, both hMat for hypergraph denoting and lMat for label recording will be constructed, it is easy to be analyzed that the complexity of construction of these matrix is proportional to their sizes. Therefore, complexity of the first phase is:

$$O(ne + nc) = O(ne)$$
In the multi-classification procedure, complexity of each iteration should first be analyzed. It will looped for every edge and every class, within the loop, $f_{out}$ of size $n$ by $c$ will first be constructed to record the change of this iteration. Afterwards, maximum and minimum prediction value will be separately found for tail and head of this edge. Since group members found that using heaps to store the extreme will leads to tiny logical flaws, the head and tail will be iterated over to find the maximum and minimum, which is linear to the number of data points on this edge. For every edge, the complexity can be analyzed as $n$ as edges for one particular feature will share the total $n$ data points and hence for average the number of data points on every edge should be approximately proportional to $n$. Hence, complexity of the subgradient learning in each iteration should be:

$$O(\mathcal{O}(nc) + \mathcal{O}(\frac{nce}{\alpha}))$$

As the $\alpha$ in the above big O notation is approximately a constant, the general complexity within each iteration is therefore:

$$O(\mathcal{O}(nc) + \mathcal{O}(nce)) = O(nce)$$

Consider the number of iterations $k$, the complexity of the whole algorithm is:

$$O(\mathcal{O}(ne) + k\mathcal{O}(nce)) = O(knce)$$

### 7. Experiment and Result

#### 7.1 Parallel Computing v.s. Non Parallel Computing

As discussed in section 5.1.4 and 6.3, parallel computing is utilized in the implementation. However, limited by number of cores of the computer (four for the one used in this project), the parallel version may suffer overhead for context switch between the threads. Whether and to what extent parallel computing will accelerate the learning will be examined in this section.

Mushroom, audiology and car are the three participants in this experiments. For each dataset, 500, 1000, 2000 number of iterations are operated, and they are executed with separately the parallel version and the non-parallel version. Within each slot of the experiments, 5 identical duplicates will be executed and final time spent is the average of the 5 executions.
**Mushroom Dataset: 2 threads**

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Singular</th>
<th>Parallel</th>
<th>Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>500</td>
<td>00:58</td>
<td>00:38</td>
<td>0.66</td>
</tr>
<tr>
<td>1000</td>
<td>01:55</td>
<td>01:27</td>
<td>0.76</td>
</tr>
<tr>
<td>2000</td>
<td>03:55</td>
<td>02:32</td>
<td>0.65</td>
</tr>
</tbody>
</table>

**Car Evaluation Dataset: 4 threads**

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Singular</th>
<th>Parallel</th>
<th>Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>500</td>
<td>00:00:08</td>
<td>00:00:04</td>
<td>0.45</td>
</tr>
<tr>
<td>1000</td>
<td>00:00:17</td>
<td>00:00:08</td>
<td>0.45</td>
</tr>
<tr>
<td>2000</td>
<td>00:00:33</td>
<td>00:00:16</td>
<td>0.48</td>
</tr>
</tbody>
</table>

**Audiology Dataset: 25 threads**

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Singular</th>
<th>Parallel</th>
<th>Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>500</td>
<td>00:05:52</td>
<td>00:03:15</td>
<td>0.55</td>
</tr>
<tr>
<td>1000</td>
<td>00:11:31</td>
<td>00:06:30</td>
<td>0.56</td>
</tr>
<tr>
<td>2000</td>
<td>00:26:24</td>
<td>00:12:53</td>
<td>0.48</td>
</tr>
</tbody>
</table>

Figure 12, parallel versus nonparallel on a 4-core computer for mushroom, audiology and car evaluation datasets. Left: table for running time record of the mushroom dataset and the ratio for execution with singular version and execution with parallel version. Right: bar graph drawn according to the table in the left, blue bars are the running time for singular version, right bars represent the time for parallel version.

**Result and Explanation:** As illustrated in figure 12, implementation with multi-thread can definitely accelerate the learning process. As the execution and computation resource assignment are not manageable by group members, the result will be influenced by the randomness as well as the state of the computer. However, generally speaking, the result matches the group members’ expectation. To explain more clearly, time for part that can not be parallelly executed is denoted by $T_1$, time for parallelable
part as $T_2$, overhead for threads context switch is $m$, number of classes of the dataset is still $c$. Accordingly, time for non parallel version should simply and ideally be:

$$T_1 + T_2$$

Now considering the parallel version, CPU in the computer for this project is 4-core, the time for dataset with $c<4$ should be:

$$T_1 + \frac{T_2}{c}$$

Time for dataset with $c>4$ should be:

$$T_1 + \frac{T_2}{4} + (c - 4) \times m$$

As only four threads can be executed parallelly by the computer used in this project, the others will only increase the threads context switch overhead.

Hence optimization for the dataset with 4 classes reaches the best through parallel computing for my computer, which conforms to the experiment result.

Accordingly, it can be inferred that with distributed computing, the parallel version of the multi-classification subgradient method.

### 7.2 Activation Functions Experiment

As introduced in section 4.2.1, four activation function: sigmoid function, softsign function, softplus function and ELU function will be used to preliminarily map the separate raw prediction before the merging process. This experiment purposes to figure out whether the activation function can relieve overfitting and bias hence increase the prediction accuracy.

In this experiment, mushroom dataset and car evaluation dataset will be used because of their moderate difficulty.

For mushroom dataset, the learning procedure contain 1000 iterations and the accuracy in the table is the average for 5 duplicate operations.
Activation Functions Experiment for Mushroom Dataset

<table>
<thead>
<tr>
<th>Function</th>
<th>20</th>
<th>40</th>
<th>60</th>
<th>80</th>
<th>100</th>
<th>120</th>
<th>140</th>
<th>160</th>
<th>180</th>
<th>200</th>
</tr>
</thead>
<tbody>
<tr>
<td>no</td>
<td>16.69% ± 5.29%</td>
<td>9.81% ± 0.49%</td>
<td>7.12% ± 0.86%</td>
<td>8.06% ± 1.13%</td>
<td>7.73% ± 0.88%</td>
<td>8.10% ± 2.35%</td>
<td>6.13% ± 1.21%</td>
<td>5.85% ± 1.22%</td>
<td>7.11% ± 1.94%</td>
<td>5.97% ± 0.78%</td>
</tr>
<tr>
<td>sigmoid</td>
<td>16.87% ± 5.22%</td>
<td>7.34% ± 1.08%</td>
<td>6.39% ± 1.01%</td>
<td>4.81% ± 0.52%</td>
<td>5.66% ± 0.87%</td>
<td>5.22% ± 0.87%</td>
<td>5.71% ± 0.44%</td>
<td>5.46% ± 1.24%</td>
<td>7.93% ± 2.10%</td>
<td>6.04% ± 1.52%</td>
</tr>
<tr>
<td>softplus</td>
<td>11.74% ± 1.14%</td>
<td>6.93% ± 0.49%</td>
<td>8.62% ± 0.82%</td>
<td>5.73% ± 1.02%</td>
<td>5.22% ± 0.35%</td>
<td>5.27% ± 0.61%</td>
<td>6.13% ± 1.45%</td>
<td>7.89% ± 1.18%</td>
<td>4.56% ± 0.73%</td>
<td>6.82% ± 1.32%</td>
</tr>
<tr>
<td>softsign</td>
<td>12.85% ± 1.62%</td>
<td>8.88% ± 1.13%</td>
<td>6.35% ± 0.74%</td>
<td>8.62% ± 0.59%</td>
<td>5.54% ± 1.12%</td>
<td>7.66% ± 0.62%</td>
<td>4.71% ± 0.87%</td>
<td>5.59% ± 0.92%</td>
<td>4.81% ± 0.75%</td>
<td>5.35% ± 0.80%</td>
</tr>
<tr>
<td>elu</td>
<td>16.02% ± 2.29%</td>
<td>12.89% ± 5.57%</td>
<td>6.10% ± 1.04%</td>
<td>7.04% ± 1.41%</td>
<td>7.14% ± 0.80%</td>
<td>8.00% ± 1.51%</td>
<td>6.47% ± 1.93%</td>
<td>6.54% ± 1.18%</td>
<td>5.71% ± 0.54%</td>
<td>4.75% ± 0.67%</td>
</tr>
</tbody>
</table>

Table 4: Table recording result for activation function experiments on the mushroom dataset.

Figure 13: accuracy - train size table and graph for activation functions on mushroom dataset.
### Activation Functions Experiment for Car Evaluation Dataset

<table>
<thead>
<tr>
<th></th>
<th>10</th>
<th>20</th>
<th>30</th>
<th>40</th>
<th>50</th>
<th>60</th>
<th>70</th>
<th>80</th>
<th>90</th>
<th>100</th>
</tr>
</thead>
<tbody>
<tr>
<td>no</td>
<td>29.49% ± 0.26%</td>
<td>27.78% ± 1.52%</td>
<td>29.27% ± 0.51%</td>
<td>30.13% ± 0.13%</td>
<td>30.20% ± 0.18%</td>
<td>30.08% ± 0.10%</td>
<td>30.05% ± 0.08%</td>
<td>30.01% ± 0.10%</td>
<td>29.98% ± 0.12%</td>
<td>30.06% ± 0.14%</td>
</tr>
<tr>
<td>sigmoid</td>
<td>28.99% ± 0.83%</td>
<td>27.34% ± 1.30%</td>
<td>29.59% ± 0.42%</td>
<td>29.99% ± 0.27%</td>
<td>30.11% ± 0.06%</td>
<td>30.19% ± 0.10%</td>
<td>29.98% ± 0.14%</td>
<td>30.06% ± 0.13%</td>
<td>30.07% ± 0.38%</td>
<td>30.27% ± 0.38%</td>
</tr>
<tr>
<td>softplus</td>
<td>28.40% ± 0.87%</td>
<td>28.40% ± 0.48%</td>
<td>29.36% ± 0.48%</td>
<td>29.17% ± 0.91%</td>
<td>29.49% ± 0.51%</td>
<td>30.01% ± 0.15%</td>
<td>30.01% ± 0.09%</td>
<td>30.22% ± 0.29%</td>
<td>29.90% ± 0.16%</td>
<td>30.14% ± 0.07%</td>
</tr>
<tr>
<td>softsign</td>
<td>29.62% ± 0.42%</td>
<td>27.17% ± 0.94%</td>
<td>29.98% ± 0.19%</td>
<td>29.27% ± 0.86%</td>
<td>30.15% ± 0.07%</td>
<td>30.22% ± 0.22%</td>
<td>30.14% ± 0.08%</td>
<td>29.96% ± 0.13%</td>
<td>29.80% ± 0.14%</td>
<td>30.20% ± 0.13%</td>
</tr>
<tr>
<td>ELU</td>
<td>29.62% ± 0.25%</td>
<td>28.57% ± 0.64%</td>
<td>29.63% ± 0.44%</td>
<td>29.94% ± 0.04%</td>
<td>29.98% ± 0.08%</td>
<td>29.80% ± 0.14%</td>
<td>30.05% ± 0.08%</td>
<td>30.24% ± 0.31%</td>
<td>29.98% ± 0.12%</td>
<td>30.21% ± 0.07%</td>
</tr>
</tbody>
</table>

Table 5: Table recording result for activation function experiments on the car evaluation dataset.

![Car Evaluation](image)

Figure 14: accuracy - train size table and graph for activation function experiments on car evaluation dataset.

**Result and Explanation:** As illustrated in table 5 and figure 14 compared with experiments done with no activation (the blue line), in general the activation functions are able to increase the prediction accuracy of the multi-classification subgradient method. The reason for such an increase might be the ability of the activation functions to alleviate the influence of the biased learning and overfitting to the final result, as discussed in section 4.2.1.

### 7.3 Regularization Potential Function Experiments

The rescaled gaussian function is the candidate for the exploration of regularization
function. In this experiments, it will be examined Gaussian function’s influence on both accuracy and convergence rate. Candidates have been introduced in section 5.1.1.

7.3.1 Accuracy Experiment

In the experiment for the accuracy of gaussian function compared with quadratic function, 1000 iterations will be taken for both regularization potential function, and result for each entry of the table is derived from five duplicates.

**Potential Functions Experiment for Mushroom Dataset**

<table>
<thead>
<tr>
<th></th>
<th>20</th>
<th>40</th>
<th>60</th>
<th>80</th>
<th>100</th>
<th>120</th>
<th>140</th>
<th>160</th>
<th>180</th>
<th>200</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quadratic</td>
<td>16.69%</td>
<td>± 5.29%</td>
<td>9.81%</td>
<td>± 0.49%</td>
<td>7.12%</td>
<td>± 0.86%</td>
<td>8.06%</td>
<td>± 1.13%</td>
<td>7.73%</td>
<td>± 0.88%</td>
</tr>
<tr>
<td>gaussian</td>
<td>14.51%</td>
<td>± 1.58%</td>
<td>11.04%</td>
<td>± 0.79%</td>
<td>7.88%</td>
<td>± 1.20%</td>
<td>7.93%</td>
<td>± 1.08%</td>
<td>7.62%</td>
<td>± 0.71%</td>
</tr>
</tbody>
</table>

*Table 6: Table recording result for potential function experiments on the mushroom dataset.*

*Figure 15: accuracy - train size table and graph for potential functions on mushroom dataset.*

**Result:** The prediction accuracy will not be affected by the gaussian potential function, the accuracies of mushroom prediction with different training set size are generally identical between quadratic function and gaussian function.

7.3.2 Convergence Speed Experiment

To figure out the speed of convergence, precision is defined as 2-norm of $f - f_{old}$, where $f$ is the prediction value of current iteration while $f_{old}$ is for the last one. The algorithm will stop learning only when the precision reaches the given value. Number
of iterations will be recorded representing the speed of convergence.

![Convergence with Different Potential Function](image)

**Figure 16:** Convergence speed by quadratic versus gaussian. The vertical axis is the number of iterations that required to converge. The horizontal axis is the inverse of the precision.

**Result and Explanation:** As illustrated by figure 16, with increase of precision\(^{-1}\), number of iterations required to converge for gaussian function is less than quadratic function. Hence, for mushroom dataset, the quadratic function has indeed accelerated the convergence speed.

### 7.4 Learning Compared with Popular Machine Learning Models

Mushroom, car evaluation and nursery are the three candidates in this section. The objective of this experiment is to examine the performance of the classification models when the training set is small compared to the whole dataset size. The selection of training set and dataset size follows table 3 in section 5.2. As the prediction result provided by Python scikit learn module is stable, only standard error of subgradient method has been recorded. Candidates are one mentioned in section 5.
**Mushroom**
The total dataset size of mushroom is 8124, with size of selected training set between 20 to 200, which is between 0.25% ~ 2.5% of the original dataset. There are in total 119 hyperedges. Result is derived from learning with 2000 iterations and average of 5 duplicated experiments. Softsign and gaussian function are used as activation function and regularization potential function.

**Final Result for Mushroom Dataset**

<table>
<thead>
<tr>
<th></th>
<th>20</th>
<th>40</th>
<th>60</th>
<th>80</th>
<th>100</th>
<th>120</th>
<th>140</th>
<th>160</th>
<th>180</th>
<th>200</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Subgradient</strong></td>
<td>12.91%</td>
<td>8.29%</td>
<td>6.33%</td>
<td>7.26%</td>
<td>5.66%</td>
<td>4.76%</td>
<td>7.52%</td>
<td>5.92%</td>
<td>5.42%</td>
<td>3.63%</td>
</tr>
<tr>
<td></td>
<td>± 0.57%</td>
<td>± 0.79%</td>
<td>± 0.89%</td>
<td>± 0.86%</td>
<td>± 1.31%</td>
<td>± 0.81%</td>
<td>± 0.64%</td>
<td>± 0.63%</td>
<td>± 0.67%</td>
<td>± 0.95%</td>
</tr>
<tr>
<td><strong>LR</strong></td>
<td>11.11%</td>
<td>4.59%</td>
<td>6.21%</td>
<td>3.80%</td>
<td>3.46%</td>
<td>2.39%</td>
<td>2.29%</td>
<td>1.89%</td>
<td>1.70%</td>
<td>1.47%</td>
</tr>
<tr>
<td><strong>rbf kernel SVM</strong></td>
<td>21.36%</td>
<td>9.11%</td>
<td>8.16%</td>
<td>7.53%</td>
<td>6.79%</td>
<td>6.47%</td>
<td>5.10%</td>
<td>4.46%</td>
<td>4.63%</td>
<td>4.04%</td>
</tr>
<tr>
<td><strong>random forest</strong></td>
<td>2.57%</td>
<td>2.59%</td>
<td>2.73%</td>
<td>2.08%</td>
<td>2.09%</td>
<td>2.00%</td>
<td>2.07%</td>
<td>2.19%</td>
<td>1.89%</td>
<td>1.89%</td>
</tr>
<tr>
<td><strong>KNN</strong></td>
<td>19.75%</td>
<td>11.71%</td>
<td>9.47%</td>
<td>8.14%</td>
<td>6.14%</td>
<td>5.64%</td>
<td>4.86%</td>
<td>4.51%</td>
<td>4.07%</td>
<td>3.17%</td>
</tr>
<tr>
<td><strong>Naive Bayes</strong></td>
<td>12.13%</td>
<td>8.88%</td>
<td>7.53%</td>
<td>7.36%</td>
<td>10.18%</td>
<td>8.99%</td>
<td>10.63%</td>
<td>9.45%</td>
<td>9.52%</td>
<td>8.61%</td>
</tr>
</tbody>
</table>

Table 7: Table recording result for final result of the mushroom dataset.

**Figure 17: final prediction result for mushroom dataset.**

**Result:** According to figure 17, the multi-classification subgradient method with 2000 iterations has generally outperformed rbf kernel SVM, KNN and Naive Bayes.
Car Evaluation
The total dataset size of car evaluation is 1728, with size of selected training set between 10 to 100, which is between 0.58% ~ 5.8% of the original dataset. There are in total 22 hyperedges. Each result is derived from average of 5 duplicated learning process with 2000 iterations. ELU and quadratic functions are used separately for activation function and regularization potential function.

**Final Result for Car Evaluation Dataset**

<table>
<thead>
<tr>
<th>Method</th>
<th>10</th>
<th>20</th>
<th>30</th>
<th>40</th>
<th>50</th>
<th>60</th>
<th>70</th>
<th>80</th>
<th>90</th>
<th>100</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subgradient</td>
<td>29.93%±0.89%</td>
<td>27.79%±1.50%</td>
<td>28.02%±1.52%</td>
<td>29.88%±0.07%</td>
<td>29.98%±0.16%</td>
<td>29.98%±0.12%</td>
<td>30.28%±0.24%</td>
<td>30.07%±0.08%</td>
<td>30.02%±0.10%</td>
<td>29.94%±0.12%</td>
</tr>
<tr>
<td>LR</td>
<td>43.56%</td>
<td>34.00%</td>
<td>34.47%</td>
<td>32.83%</td>
<td>32.99%</td>
<td>32.90%</td>
<td>33.25%</td>
<td>32.72%</td>
<td>32.95%</td>
<td>33.12%</td>
</tr>
<tr>
<td>linear kernel SVM</td>
<td>49.36%</td>
<td>29.80%</td>
<td>29.74%</td>
<td>33.37%</td>
<td>32.80%</td>
<td>33.27%</td>
<td>29.66%</td>
<td>29.85%</td>
<td>29.84%</td>
<td>29.84%</td>
</tr>
<tr>
<td>random forest</td>
<td>34.95%</td>
<td>33.00%</td>
<td>32.61%</td>
<td>32.28%</td>
<td>32.25%</td>
<td>31.65%</td>
<td>31.11%</td>
<td>31.70%</td>
<td>32.04%</td>
<td>32.07%</td>
</tr>
<tr>
<td>KNN</td>
<td>29.80%</td>
<td>30.09%</td>
<td>32.79%</td>
<td>33.01%</td>
<td>32.80%</td>
<td>32.59%</td>
<td>31.86%</td>
<td>31.25%</td>
<td>31.14%</td>
<td>32.13%</td>
</tr>
<tr>
<td>Naive Bayes</td>
<td>39.58%</td>
<td>38.21%</td>
<td>39.45%</td>
<td>37.08%</td>
<td>37.16%</td>
<td>37.06%</td>
<td>38.10%</td>
<td>37.24%</td>
<td>42.83%</td>
<td>43.06%</td>
</tr>
</tbody>
</table>

Table 8: Table recording result for final result of the car evaluation dataset.

Figure 18: Final result for car evaluation dataset.

**Result:** As illustrated in table 8 and figure 18, subgradient method dominates the learning accuracy compared with other classification methods when the training set is small for car evaluation dataset.
Nursery
The total dataset size of nursery is 12960, with size of selected training set between 500 to 2500, which is between 3.9% ~ 19.3% of the original dataset. Since according to table 2 of section 5.2.4, nursery is a dataset with high difficulty, therefore, more labels are required for learning process. There are in total 28 hyperedges. Since the convergence of nursery is quite slow, group members have no enough time to conduct duplicate experiments. Softsign and gaussian functions are used as activation function and regularization potential function.

<table>
<thead>
<tr>
<th>Method</th>
<th>500</th>
<th>1000</th>
<th>1500</th>
<th>2000</th>
<th>2500</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subgradient</td>
<td>38.92%</td>
<td>35.63%</td>
<td>34.08%</td>
<td>36.89%</td>
<td>37.83%</td>
</tr>
<tr>
<td>LR</td>
<td>47.74%</td>
<td>46.74%</td>
<td>45.86%</td>
<td>45.43%</td>
<td>45.36%</td>
</tr>
<tr>
<td>rbf kernel SVM</td>
<td>53.40%</td>
<td>48.64%</td>
<td>46.27%</td>
<td>45.32%</td>
<td>44.92%</td>
</tr>
<tr>
<td>random forest</td>
<td>54.98%</td>
<td>54.20%</td>
<td>54.82%</td>
<td>56.20%</td>
<td>57.66%</td>
</tr>
<tr>
<td>KNN</td>
<td>56.25%</td>
<td>53.72%</td>
<td>52.35%</td>
<td>53.49%</td>
<td>53.22%</td>
</tr>
<tr>
<td>Naive Bayes</td>
<td>57.49%</td>
<td>54.95%</td>
<td>58.80%</td>
<td>58.25%</td>
<td>57.61%</td>
</tr>
</tbody>
</table>

Table 9: Table recording result for final result of the nursery dataset.

Figure 19: final prediction result for nursery dataset.

Result: The nursery dataset is very difficult for learning, all of the classification methods are not able to perform well with small training set. Relatively speaking, multi-classification subgradient algorithm outperforms other popular classification model when learning nursery dataset with training set of size 500 ~ 2500.
## 8. Distribution of Labor

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Member in charge</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project Plan</td>
<td>Jiali, Chen</td>
</tr>
<tr>
<td>Literature Review</td>
<td>Jiali, Chen</td>
</tr>
<tr>
<td>Experiment Setup &amp; Data preprocessing</td>
<td>Jiali, Chen</td>
</tr>
<tr>
<td>Python Implementation</td>
<td>Jiali, Chen</td>
</tr>
<tr>
<td>Algorithm 1 Markov Operator</td>
<td>Jiali, Chen</td>
</tr>
<tr>
<td>Algorithm 2 Subgradient Method SGM</td>
<td>Ying, Zhang</td>
</tr>
<tr>
<td>Algorithm 3 Semi-Supervised Learning</td>
<td>Jiali, Chen</td>
</tr>
<tr>
<td>Plotting and comparing (Original Implementation)</td>
<td>Jiali, Chen</td>
</tr>
<tr>
<td>Interim Presentation &amp; Report</td>
<td>Jiali, Chen</td>
</tr>
<tr>
<td>Study of C++ machine learning library &amp; potential function</td>
<td>Ying Zhang</td>
</tr>
<tr>
<td>Study of multi-class classification &amp; activation functions</td>
<td>Jiali, Chen</td>
</tr>
<tr>
<td>C++ version of Implementation</td>
<td>Jiali, Chen</td>
</tr>
<tr>
<td>Sem 2</td>
<td>(with Multi-class Extension &amp; activation)</td>
</tr>
<tr>
<td>-------</td>
<td>----------------------------------------</td>
</tr>
<tr>
<td></td>
<td>Parallel Computing Extension</td>
</tr>
<tr>
<td></td>
<td>Testing and plotting (Multi-classification)</td>
</tr>
<tr>
<td></td>
<td>Testing and plotting (parallel programming)</td>
</tr>
<tr>
<td></td>
<td>Testing and plotting (activation function)</td>
</tr>
<tr>
<td></td>
<td>Testing and plotting (potential function)</td>
</tr>
<tr>
<td></td>
<td>Final Presentation &amp; Individual Reports</td>
</tr>
</tbody>
</table>

*Table 10: tasks list and distribution of labor.*
9. Limitations and Difficulties

9.1 Computation Ability of the Computer

During the experiments, the limited computation ability of the computer to some extent restrict the project. The probable consequence caused by the insufficient of computation ability may appear in two aspects. Firstly, for the large dataset like nursery dataset requires a large amount of time to complete an iteration, thus it might be impossible to wait until the learning process finally converge. This increases the possibility that the prediction accuracy in the project have not reached the best yet, which may influence the objectivity and precision of the conclusion. Secondly, as prediction experiments might be repeated enough due to the limited computing ability, the fluctuation caused by randomness of computation may not be completed smoothed by enough duplications.

9.2 Selection of Training Set

In the current C++ version implementation, the training set is selected by absolutely randomness. Therefore, it is difficult to avoid insufficient learning caused by the imbalance in the selection of training data especially for small training set. If there is a dominating class in the training set selected, the learning process will be greatly decelerated. To clarify, for instance 20 data points are selected as the training set, within which 19 of them belong to class 1. According to the learning entries recovery mechanism introduced in section 4.1, the initial state for class 1 will be straight 1’s expect one exception. Under this condition, most edges will have identical maximum for tail and minimum for head. As the subtraction between them is 0, no learning will happen on almost all of the edges. Hence, a reasonable and efficient training set selection method is suggested to be designed in the future work.

9.3 Selection of Categorical Multi-classification Dataset

During the project, group members found it difficult to find suitable dataset for the algorithm. The subgradient requires classification datasets with pure categorical features. Therefore, even from UCI machine learning repository, one of the most popular machine learning data sources, it is not easy to find a dataset satisfying the requirement of the algorithm as well as perfectly handleable by our devices.
10. Future Work and Conclusion

10.1 Future Work

10.1.1 Head and Tail Configuration

Head and tail are defined identical as the h matrix in the current version. As mentioned in section 2.5, with well-defined head set and tail set, higher order causal relationship can be expressed. For that reason, it is worthy of being figured out a reasonable and efficient way of defining the head data point and tail data point according to the dataset information. As more information is expressed by the modeling, the learning accuracy may have a further improvement.

10.1.2 Experiment on the Cloud

As illustrated and discussed in section 7.2, the learning procedure is accelerated by parallel execution. However, the speed limitation of personal computer restrict the extent of the acceleration, as there are only 4 CPU to execute the algorithm. Cloud computing is the potential stage to fully exhibit the acceleration ability for parallel computing. As there are larger number of processors with strong computation ability, almost all the threads can start and be executed synchronously.

GPU execution may be another alternative. However, since GPU is characterized by the large number of cores but relatively singular computation ability. As in most cases, the number of classes of a dataset can not match the number of cores in a GPU, the GPU may not performs its best.

10.1.3 Further Generalization

As mentioned in section 9.3, group members encountered difficulty in finding classification dataset with pure categorical features. The narrow of the set of applicable datasets revealed the fact that the algorithm is not generalized enough although it is able to handle multi-classification. Further generalization may be found in the hypergraph modeling for integer features, and even for real features. With such generalization, the subgradient method is capable of most of the classification problems.
10.1.4 Further Research on Activation Function and Regularization Potential Function

Experiments in this project have illustrated a slight improvement given by the activation function and gaussian function. However, due to the limitation of computation ability and time, the experiment conducted is not sufficient enough. Therefore, more experiments should be operated in the future to deliver a more solid result.

Moreover, as there are many other alternatives for the activation function and regularization potential function, much more researched can be emphasized on accordingly. For example, the cross entropy function $x \times \ln(x) + (1-x) \times \ln(1-x)$ is another potential candidate for the subgradient method. Therefore, further researchers could focus on the alternatives of regularization potential function.

10.2 Conclusion

To conclude, this project implemented the subgradient algorithm on both Python platform and C++ platform. Group members also generalized the current subgradient algorithm to deal with multi-classification problems. Accordingly, parallel computing is used in the C++ implementation. Experiment shows that the multi-thread computing can approximately double the speed of the learning process on the 4-core computer used in this project. Moreover, this project introduced sigmoid functions and smoothed ReLu function as activation functions to alleviate the bias and overfitting of the separate rough result. Experiments proved the efficiency of activation functions in the mushroom and car evaluation datasets. Furthermore, alternatives for the original quadratic regularization potential function is found, the rescaled Gaussian function. Experiment illustrates that Gaussian function can improve the convergence speed of mushroom dataset without decrease the accuracy. Comparison experiments are also conducted to examine the efficiency of the multi-classification subgradient method. It is demonstrated that this algorithm is able to outperform most of the popular classification model for both bi-classification and multi-classification problems when the training set is small compared with the dataset size. In the future, research may be emphasized on head and tail configuration, transplant of the implementation onto cloud template, further generalization for integer and real features and figuring out other alternatives for the quadratic regularization potential and activation function.
References


