WITS

Parallel Democratic Support Vector Machines for Multi-Class Classification

Authors:

Dario Vieira (806414)

Marco Lerena (1242414)

July 25, 2018

Contents

1 Introduction

This report outlines a parallel implementation of multi-class classification using support vector machines with ensemble learning. We first introduce the serial implementation which uses the Pegasos algorithm to train multiple SVMs, and a voting paradigm that performs multi-class classification using the trained set of SVM models. We then discuss how this implementation is parallelized using MPI to distribute the model creations across multiple nodes. We outline our methodology, including the datasets used, the testing environment, and the experiment setup. Finally, we present the results of our experiments, alongside an analysis of the training time and accuracy achieved.

2 Background

2.1 Support Vector Machine & PEGASOS

The support vector machine (SVM) is a supervised learning model used for binary classification problems. Given a labeled set of data-points, the SVM constructs a hyperplane that separates two classes using the widest margin possible. Producing a wide margin aims to lower the generalization error of the classifier. Constructing the hyperplane with the widest margin is an optimization problem.

PEGASOS stands for Primal Estimated sub-GrAdient SOlver for SVM, which is the weight update method created by Shalev-Shwartz et al. 1 for an SVM.

2.2 Ensemble Learning

Ensemble learning is a machine learning paradigm where a set of classifiers are trained to solve a single problem. The classifiers in the ensemble are known as a base learners, and are usually homogeneous i.e. all alike in some way. In our case, for example, the ensemble will be a set of support vector machines. Once an ensemble has been trained, the learners are then combined in order to solve the core problem. One common combination scheme is *majority voting*, in which each member of the ensemble casts a vote as its contribution to the solution. The voting combination scheme is what we will be using to perform multi-class classification with a set of binary SVM classifiers. This is elaborated on in section 3.

2.3 Principle Component Analysis

PCA is a method of dimensionality reduction, this is done by generating eigenvalues that best describe the variation of the data. These vectors are used to transform the original data into a lower dimensional space.

3 Algorithm Design

3.1 Context

In order to use Support Vector Machines for multi-class classification, we employ the ensemble learning scheme of majority voting. Multiple binary SVMs are produced, with each classifier being trained on a different combination of classes. The number of classifiers required is $\binom{c}{2}$, where c is the number of classes that need to be classified.

For example, with classes = $\{1, 2, 3\}$, a set of $\binom{3}{3} = 3$ weight vectors are trained, one for each classifier; W(1,2), W(1,2), and W(2,3). Once the various weight vectors have been trained, we employ the voting procedure. For an unclassified data-point x, each classifier casts a vote, which is stored in a vote matrix.

For each $W(i, j)$:

- if, $h(x, W(i, j)) < 0$, $voteMatrix(i, j) = -1$
- else if, $h(x, W(i, j)) > 0$, $voteMatrix(i, j) = 1$,
  where $h(x, W(i, j)) = x \cdot W(i, j)$

This can be interpreted as follows; if voteMatrix(i,j) = -1, then classifier i, j has voted that x is in class i.

Once all classifiers cast their votes, a vote count, of size c, is produced. Vote count is then searched linearly, and the class with the most votes is the final classification for the unseen x. If a tie is encountered during the linear search, the tie is resolved by consulting the classifier trained to distinguish the tied classes. For example, if a tie is encountered between class 5 and 7, the classifier trained to distinguish 5 and 7 is consulted to decide on which class is more likely. This can be done by checking voteMatrix(5,7) for the result of its classification. Once the linear search is complete, all possible ties have been resolved, and we are left with the final classification.

3.2 Serial

The context above describes how we utilize an ensemble of SVMs with a voting scheme, to classify a data-point. It is important to keep this in mind when training to ensure the ensemble suits our purposes. The two algorithms below facilitate the main elements of our training process.

3.2.1 Multiple Model Creation

Algorithm 1 loads the training data, and launches Algorithm 2 $\binom{c}{2}$ times. Each time Algorithm 2 is launched, it is passed the subset of data required to train a set of weights for classes i and j. The result of Algorithm 2 is stored in weight matrix at $W(i, j)$.

Algorithm 1 Multiple Model Creation

```
Load Data
Set T and λ
Initialize 3D weight matrix W.
(of size c by c by d)
for i=1 to c do
    for j=0 to i do
        S = subset of data where target y = i, or y = j
        W_(i,j) = Algorithm 2(S,λ,T) //Pegasos training algorithm
    end for
end for
```

3.2.2 Pegasos

Algorithm 2 is the Pegasos algorithm called to train a set of weights for an SVM classifier. As stated above, the data sent to the algorithm is only the subset required to train a binary classifier for two (i and j) of the c classes.

Algorithm 2 Serial Pegasos

```
Input:
Training data set: S
Regularisation Parameter: λ
Number of Iterations: T
Output:
Learned weights W
set W = 0
for t = 1 to T do
    choose a sample (x_t, y_t) uniformly at random
    set η_t = 1/(λt)
    if y_t h(x_t) < 1 then
        W_{t+1} = (1 − η_t)W_t + η_t y_t x_t
    end if
    if y_t h(x_t) ≥ 1 then
        W_{t+1} = (1 − η_t)W_t
    end if
end for
```

3.3 Parallel

Seeing as though each set of weights is generated in isolation, the creation of the SVM ensemble is an embarrassingly parallel problem. Thus, we will make use of a parallel framework to distribute the model creations across multiple nodes. MPI is perfectly suited for parallelizing this component, since minimal communication between nodes is required.

3.3.1 MPI

MPI Algorithm 3 has been used to parallelize the multiple model creations (algorithm (Algorithm 1)). This is done by initializing a separate node for each set of weights that needs to be generated. Each of these nodes calls the serial Pegasos algorithm (Algorithm 2) on their portion of the data set in order to train the weights. The data is read in by only the lead node, which then broadcasts the data to all other nodes. The lead node also decides which set of weights each other node will be training, and broadcasts this information as well. As stated earlier, each set of weights required is trained to distinguish a different combination of classes. Thus, the lead node provides each other node with an (i, j) tuple, indicating which classes data should be used for its model. The weights are then reduced to a single Matrix located only in the lead node. The lead node then contains the complete weight matrix that can be used to classify a point using the voting scheme.

Algorithm 3 MPI Multi-model Generation

1: *Load Data*
2: Broadcast Data
3: *Assign each node a value for i and j*
4: Set T and λ
5: Initialize 3D weight matrix W.
(of size c by c by d)
6: S = Data set where target y == i or y==j
7: $W_{(i,j)}$ = **Algorithm 2(S,λ,T)** //Pegasos training algorithm
8: Reduce various weight vectors into a single matrix

Note: The pegasos algorithm itself remains a serial process.

4 Experiment Setup

The aim of our experiment is to determine the accuracy of the above algorithm on various datasets, and determine if there is an improvement in training time when using MPI.

Below we describe the setup of our experiment, including the testing environment and the datasets used. We then present our results, followed by an analysis of the accuracy and training time.

4.1 Testing Environment

1. The testing environment for the serial and parallel implementation is as follows:
    - CPU: Intel core i7-7700 3.60GHz
    - Number of threads on CPU: 8
    - GPU: NVIDIA Geforce GTX 1060 6GB
    - Operating System: Ubuntu 16.04 LTS 64bit
2. **Note**: the optimal testing environment for the MPI implementation is the MSL Cluster at the University of the Witwatersrand. However, due to ongoing maintenance on the cluster, we are only able to run our experiments on the system above.

4.2 Data

The implementations are tested on the following three datasets:

1. MNIST
    - 10 image classes of handwritten numbers (0-9)
    - PCA applied to retain 99.99% variation
    - 50/50 train test split
2. Fashion MNIST
    - 10 image classes of clothing items (labeled 0-9)
    - PCA applied to retain 99.99% variation
    - 70/30 train test split
3. EMNIST (Balanced set)
    - 47 classes of handwritten letters and numbers (0-46)
    - PCA applied to retain 99.99% variation
    - 60/40 train test split

4.3 Test Configurations

In each test, we trained every set of weights on the same number of iterations; $T = 1000, 10000, 50000, 100000, 1000000$. We stopped testing when minimal change in accuracy was seen between tests.

The MPI implementation was tested in the same manner in order to have comparative times.

5 Results

5.1 Presentation of Results

The following three tables show training time in seconds for each dataset, alongside the accuracy achieved. The same accuracy is achieved on both the serial and parallel implementations, and so only a single value is shown here. The T values in a tables are the number of iterations used to train each SVM

Note: only the training time is shown here, as total time includes overheads for reading data and MPI setup, which is obviously different across the implementations.

5.1.1 MNIST

Table 1: MNIST

T	**Serial Time**	**Parallel Time**	**Accuracy**
1 000	0.69	0.07	86.05
10 000	5.3	0.78	89.8
50 000	25.99	3.47	91.06
100 000	51.6	7.14	91.09

- Converges by 100 000 iterations
- 10 Classes
- 45 Linear SVM's created

5.1.2 Fashion MNIST

Table 2: Fashion-MNIST

T	Serial Time	Parallel Time	Accuracy
1 000	1.12	0.16	75.56
10 000	4.5	0.7	82.54
50 000	19.3	2.53	84.34
100 000	37.9	4.97	83.92
1 000 000	372.6	46.8	84.9

- Converges by 1 000 000 iterations
- 10 Classes
- 45 Linear SVM's created
- This data set was used to test how the algorithm performed on separate data set.

5.1.3 EMNIST

Table 3: EMNIST

T	Serial Time	Parallel Time	Accuracy
1 000	17.3	3.1	64.7
10 000	132.8	17.8	67.07
50 000	646	87.2	67.45

- Converges by 50 000 iterations
- 46 Classes (the following classes have been joined: C & c, I & i, J & j, K & k, L & l, M & m, O & o, P & p, S & s, U & u, V & v, W & w, X & x, Y & y, Z & z). These groupings have been made as the capital letters are nearly indistinguishable from their lowercase counterparts.
- 990 Linear SVMs created
- This data set was used to test how the algorithm performs with more classes.
- The most common errors were between characters that are similar. The problematic characters are as follows: L & I & 1 & J, F & P, 0 & O, U & V.

5.2 Analysis of Results

The serial training times increase linearly in relation to the training iterations, as expected. The parallel times are roughly $\frac{1}{8}_{th}$ that of the serial times. This is due to the fact that in the testing environment, the number of threads that can be launched is 8, improving the training time 8 fold. As alluded to previously, the desired speedup is a direct result of the number of SVM classifiers being trained, $\binom{c}{2}$. Given enough cores, this speedup should be achievable (i.e. $10\times$ speed up on MNIST or $990\times$ speed up on EMNIST).

When increasing the number of classes from 10 with EMNIST, we expected the classifier's accuracy to improve. This was due to the assumption that as the number of classes increased, there would be more classes for incorrect votes to be cast on. This should result in the correct class winning the vote. However, our hypothesis was proven incorrect, at least on this dataset.

6 Conclusion

The classifier performs worse than the best recorded SVM on MNIST database, developed by Decost et al.2 which has an accuracy of 99.46%. This is a disappointing outcome but does not nullify this project. The idea of using an ensemble of binary SVMs for multi-class classification still produces reasonable results, if training time is of the utmost importance and the computational power is available then this algorithm will prove useful.

7 Further Work

7.1 Voting Procedure

The class with the most votes is found by conducting a linear search through the vote counts. During this search, the procedure will sequentially resolve any tie votes encountered. To reiterate what was explained in section 3.1, assume that during a vote count search, the most votes encountered so far has been for class A, with 5 votes. Now assume that when class B is encountered, it has also received 5 votes. This tie will be resolved immediately by consulting voteMatrix(A,B), which for this example we will say indicates class B. Now, the linear search encounters class C, also with 5 votes. The procedure will now consult voteMatrix(B,C) to resolve the tie. The issue with this is that voteMatrix(A,C) will not be consulted to break the tie. A further implementation should store all ties until the end of the search, and build a new vote count to resolve the tie. The new vote count should be produced by consulting voteMatrix(i,j), for each i,j of the tied classes. This is a more reliable classification as all tied classes are considered at once.

References

[1] Shai Shalev-Shwartz, Yoram Singer, Nathan Srebro, Andrew Cotter: Pegasos: Primal Estimated sub-GrAdient SOlver for SVM (2011)

[2] Dennis Decost, Bernhard Scholkopf: Training Invariant Support Vector Machines (2002)

[3] Zhi-Hua Zhou: Ensemble Learning (2002)

[4] Boser, Bernhard E. and Guyon, Isabelle M. and Vapnik, Vladimir N.: A Training Algorithm for Optimal Margin Classifiers (1992)

Data Sets:

[5] MNIST: http://yann.lecun.com/exdb/mnist/

[6] Fashion MNIST: https://www.kaggle.com/zalando-research/fashionmnist

[7] EMNIST: https://www.kaggle.com/crawford/emnist