**The Experiment Report of Machine Learning**



**SUBJECT: SOFTWARE ENGINEERING**

**October 24, 2018**

***Grade:***

**Undergraduate**

***Student ID：***

**201630664673**

***Supervisor:***

**Mingkui Tan**

***Author:***

**Yi Lan**

**SCHOOL: SCHOOL OF SOFTWARE ENGINEERING**

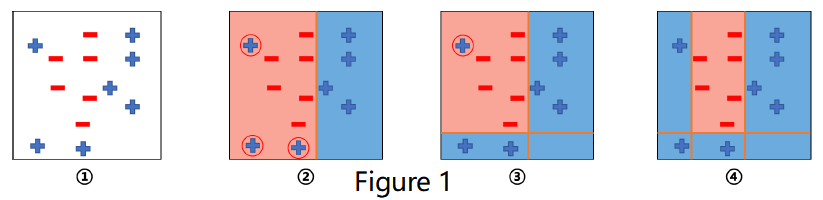
Face Detection Based on AdaBoost Method

*Abstract -* In order to understand the foundation theories of AdaBoost furtherly and get familiar with the method for face detection with AdaBoost solving multi-classes classification problem. What's more, we will also explore the uses of parameters of base classifier, Decision Tree Classifier in Scikit Learn.

# INTRODUCTION

Face detection is a common use of machine learning and has a lot of application. The main methods of face detection can be divided into image based methods and feature based methods. In this experiment, we will train a classifier based on AdaBoost method to detect if there is a face in a picture.

AdaBoost, Adaptive Boosting, is an aggressive learning algorithm. A strong AdaBoost classifier is composed of several or more weaker classifiers (we call them base classifier) which are combining linearly. For example, in Figure 1, we make use of three base classifiers through three iteration and solve a non-linear-separable problem. The main idea is *making the wrong predictive samples more import and handle it in next time.*



Therefore, the main problem can be divided into three portions: how to train a base classifier, how to combine base classifiers into a strong AdaBoost classifier and how to update the data distribution to make the wrong predictive samples more import.

In this experiment, we use the Decision Tree Classifier in Scikit Learn as base classifier and its own *fit()* function to fit our preprocessed data sets.

# METHODS AND THEORY

1. **Train a base classifier**

Because of using the Decision Tree Classifier in Scikit Learn as our base classifier, it is unnecessary to decide and code a base classifier by ourselves again. We can just use its own *fit()* function to fit our data sets.

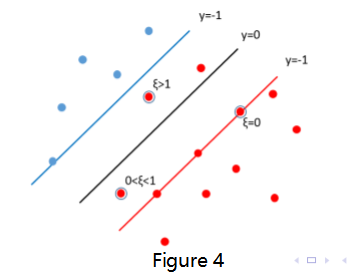
1. **Support Vector Machine**

The purpose of SVM is to find two parallel hyperplanes which can separate the two classes of samples, and at the same time, maximize the distance of the two hyperplanes. The region bounded by them is called margin, which can be calculate as

According to Max-margin Method, the optimization can be formulated as

Equivalently:

Training data may not be simply linearly separable, although it is in global view. So we introduce variable , for each , which represents how much example is on wrong side of margin boundary. If ,then this sample is at correct side; if , then it is correctly classified, but with a smaller margin than ; if , then it is incorrectly classified. (Figure 4)



Now, the optimization problem has trans to:

Now, we use Hinge Loss function as:

The complete final loss function and the gradient of the classification model is:

1. **GD: Gradient Descent**

True gradient descent is a batch algorithm, slow but sure

: the hyper-parameter learning rate

Simple procedure:

1. Initialize parameter and learning rate
2. Loop: while an approxiamte minimum is no obtianed do (a)
3. end

However, when the dataset is so large, doing a fully gradient descent will cost too much time and memory. So we could try MSGD.

1. **MSGD: Minibatch Stochastic Gradient Descent**

Rather than using a single point (SGD), use a random subset where the size is less than the original data size

Like the single random sample, the full gradient is approximated via an unbiased noisy estimate. Random subset reduces the variance by a factor of , but is also times more expensive.

Simple procedure:

1. Initialize parameter and learning rate
2. Loop: while an approxiamte minimum is no obtianed do
   1. Randomly select examples in the training set
3. end

# Experiment

**Data Set Source:** https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html#a9a

The data set, with 123 features, includes 32561 samples for training and 16281 for testing

**Experiment Steps:**

* 1. Load and preprocess the experiment data.
  2. Define and initialize hyper-parameters: learning rate, the max epochs of training, batch size.
  3. Initialize model parameters 1-diemond matrix as ones.
  4. Code the loss function and optimizer.
  5. Update the model parameters .
  6. Calculate the training loss and testing loss using current model. Record them.
  7. Output monitor information.
  8. Based on GD or MSGD repeat (e) to (f) until the train end.
  9. Output the model’s performance information.
  10. Drawing graph of training losses and testing losses, as well as with the number of iterations.

**Experiment Hyper-parameters Selection Scheme:**

1. **Logistic Regression**

Hyper-parameters Tables

|  |  |
| --- | --- |
| Learning Rate | 0.01 |
| Iteration Time | 1000 |
| Batch Size | 100 |

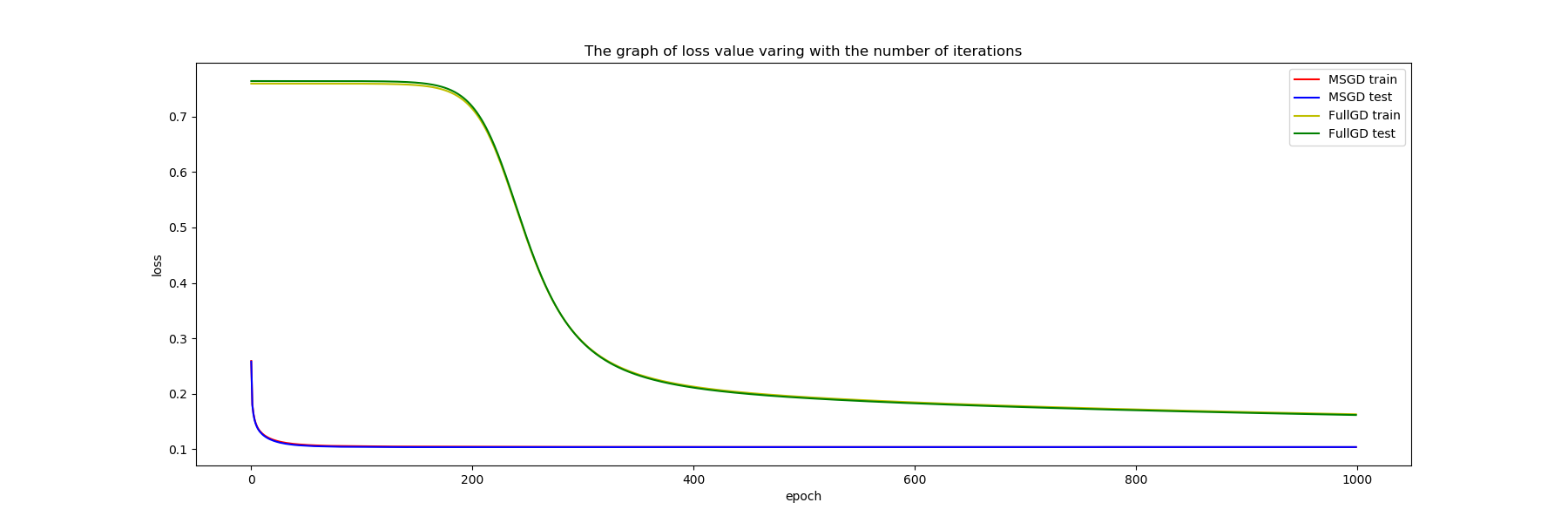
**Result reports:**

* + 1. MSGD

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Precision | Recall | F1-score | Support |
| Positive | 0.8825 | 0.9270 | 0.9042 | 12435 |
| Negative | 0.7179 | 0.6009 | 0.6542 | 3846 |
| Avg/total | 0.8436 | 0.8499 | 0.8451 | 16281 |

* + 1. Full Gradient Descent

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Precision | Recall | F1-score | Support |
| Positive | 0.8261 | 0.8817 | 0.8530 | 12435 |
| Negative | 0.5111 | 0.3999 | 0.4487 | 3846 |
| Avg/total | 0.7517 | 0.7679 | 0.7575 | 16281 |



1. **SVM**

Hyper-parameters Tables

|  |  |
| --- | --- |
| Learning Rate | 0.0001 |
| Iteration Time | 1000 |
| C | 0.7 |
| Batch Size | 100 |

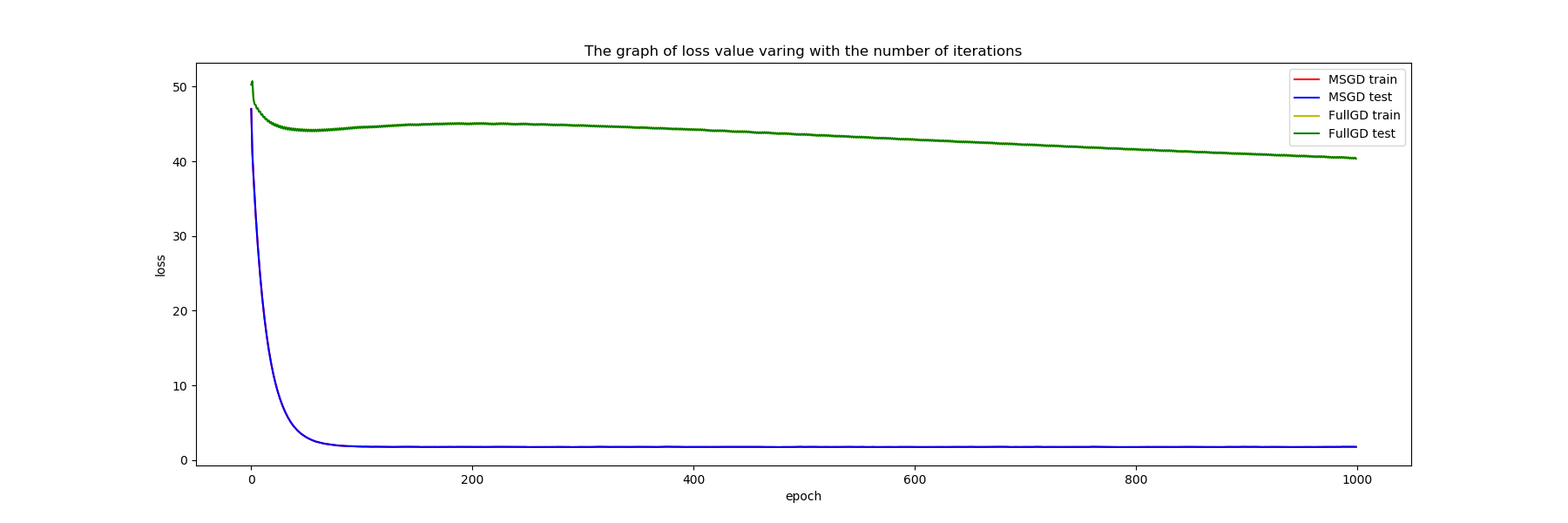
**Result reports:**

* + 1. MSGD

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Precision | Recall | F1-score | Support |
| Positive | 0.8618 | 0.9464 | 0.9021 | 12435 |
| Negative | 0.7460 | 0.5094 | 0.6054 | 3846 |
| Avg/total | 0.8345 | 0.8431 | 0.8320 | 16281 |

* + 1. Full Gradient Descent

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Precision | Recall | F1-score | Support |
| Positive | 0.9360 | 0.8096 | 0.8682 | 12435 |
| Negative | 0.5715 | 0.8211 | 0.6739 | 3846 |
| Avg/total | 0.8499 | 0.8123 | 0.8223 | 16281 |



# conclusion

Obviously, the model with SVM is better than logistic regression. And in both methods, using MSGD can converge the loss curve more fast.

Logistic Regression and Support Vector Machine are two of the foundational methods in machine learning area and both of them make a huge contribution to the development of machine learning algorithm. It is believed that many improved methods could be used to both of the algorithm and address a better results. Also, different parameters of different optimization function can lead to various results. So the future work may aim at adjusting the different parameters to get the better results and explore the optimization functions.