ASSESSMENT COVER SHEET

Assignment Details

Course: Introduction to Statistical Machine Learning

Semester/Academic Year: 2020 Sem 2

Assignment title: AdaBoost

Assessment Criteria

Assessment Criteria are included in the Assignment Descriptions that are published on each course's web site.

Plagiarism and Collusion

Plagiarism: using another person's ideas, designs, words or works without appropriate acknowledgement.

Collusion: another person assisting in the production of an assessment submission without the express requirement, or consent or knowledge of the assessor.

Consequences of Plagiarism and Collusion

The penalties associated with plagiarism and collusion are designed to impose sanctions on offenders that reflect the seriousness of the University's commitment to academic integrity. Penalties may include: the requirement to revise and resubmit assessment work, receiving a result of zero for the assessment work, failing the course, expulsion and/or receiving a financial penalty.

DECLARATION

I declare that all material in this assessment is my own work except where there is clear acknowledgement and reference to the work of others. I have read the University Policy Statement on Plagiarism, Collusion and Related Forms of Cheating:

http://www.adelaide.edu.au/policies/?230

I give permission for my assessment work to be reproduced and submitted to academic staff for the purposes of assessment and to be copied, submitted and retained in a form suitable for electronic checking of plagiarism.

Vandit
6/10/20

SIGNATURE AND DATE

2020 Introduction to Introduction to Statistical Machine Learning Assignment 2: AdaBoost

Vandit Jyotindra Gajjar
School of Computer Science, The University of Adelaide
vanditjyotindra.gajjar@student.adelaide.edu.au
Student ID: a1779153

Abstract

As time passes the Machine Learning community has seen massive growth in the field in theory as well as in applications. One of the earliest concepts for binary classification after Support Vector Machines (SVMs) [1] was Boosting [2]. This assignment/report will mainly focus on Boosting especially AdaBoost [2]. AdaBoost is a special kind of boosting algorithm specifically used for the Binary Classification problem which is mainly focused on boosting the decision trees. AdaBoost is a supervised learning model used for classification as well as regression in certain applications. In our assignment, we focus on the classification task and especially the problem of binary classification. Additionally, based on the assignment requirements we compare our custom implemented AdaBoost classifier with the available implementation of the classifier by open-source library scikit-learn [3] and with the SVMs [1]. The experiment was conducted on the publicly available dataset – Breast Cancer Wisconsin Diagnostic dataset [4] which consists of 569 number of instances – 300 training samples and 269 testing samples. This report focuses on solving the assignment questions of the AdaBoost classifier with detailed analysis. The source code is publicly available here.

A General Introduction to Boosting and AdaBoost

The AdaBoost – a boosting algorithm is basically an ensemble method. The purpose of introducing AdaBoost is to boost the performance of decision trees for the binary classification problem. In general, an ensemble learning/approach is to use several machine learning approaches/methods to get better predictive performance compared to standalone approaches/methods. Now coming to the main purpose of boosting, boosting is an algorithm that tries to create a strong model/classifier/estimator based on the many weak learners (Now weak learner is the model who performs better than guessing but performs poorly on certain

samples). In general, Boosting tries to reduce/lessen the bias error when the model is not able to gather the information for several trends in the given data.

AdaBoost which is also known as Adaptive Boosting is an algorithm/method which aims for creating a strong classifier/estimator based on the many weak learners. In certain case, a single classifier might perform poorly for prediction in one of the datasets but when we gather these types of failed classifiers and by learning progressively from the wrong classification, we can create a strong model (The classifier could be either Logistic Regression, Decision Trees, etc.). AdaBoost works with the *Decision Stumps*. Now the *Decision Stumps* are the tress, however, they do not have too many nodes nor leaves. *Decision Stumps* only have one node and two leaves, and AdaBoost uses a kind of forest having *Decision Stumps* rather than having full-grown trees. Only having a single *Decision Stump* is not a good choice for making predictions. In general, a fully grown tree combines many decisions from all the features for prediction. A *Decision Stump* can use only one variable to predict the decision. In the following sub-section, we will aim to understand the mathematical component of AdaBoost.
Consider we have a dataset of K sample points. So,

$$x_i \in R^k, y_i \in \{-1, 1\}$$

Here, k is the number of features in our dataset, x is the sample points, and y is the label which either -1/1 as we are dealing with a binary classification problem. We calculate the weighted samples for every sample point. AdaBoost provides the weight to sample points for the significance in the train set. It also has the threshold value, in which when the provided value becomes high enough it will set the training sample points more likely having maximum influence and likewise for minimum influence for lower values. At first, sample points have the same weighted sample w, where K is the sample points.

$$w = 1/K \in [0, 1]$$

It is obvious that the weighted sample will sum to 1, so the individual weight will be in the range from 0 and 1. At this stage, we will calculate the alpha influence based on the formula given in the algorithm [2].

$$\propto_t = \frac{1}{2} \ln \frac{(1 - \text{totalError})}{\text{totalError}}$$

Here alpha is the influence a particular *Decision Stump* has in final prediction. The totalError is misclassification/prediction for the training set. Consider the case when *Decision Stump* performs well the value of alpha will be relatively large. Same when the *Decision Stumps* prediction will be 0.5 for classification and 0.5 for misclassification the value of alpha will be 0, and likewise for misclassification which leads to a large negative value. Finally, for the update of the sample weight, it can be done by the following formula:

$$w_i = w_{i-1} * e^{\pm \propto}$$

AdaBoost has several advantages because of less tweaking in parameters compare to SVMs and likewise approaches. It is also useful for improving the accuracy of weak learners which makes AdaBoost more flexible and easy to use. Thus, it can be extended from the binary classification problem to text/image classification problems. However, the quality of data really matters when one wants to use AdaBoost as boosting methods learn progressively. Having a noisy/corrupt data while using AdaBoost is highly sensitive, and after some findings comparing to other boosting technique such as XGBoost [5], AdaBoost is much slower.

Dataset Statistics

In our assignment, the requirement is to use the provided data which is Breast Cancer Wisconsin Diagnostic Dataset [4]. There is 569 number of instances are given in which 300 sample points are for training, while the other 269 sample points are for testing purposes. There are 32 number of attributes/features are given for individual data points and the assignment task focuses on the classification problem.

Implementation Details

In our experiment, we have performed the training on a workstation with the Intel i-7 core processor and accelerated by NVIDIA GTX 1060 6 GB GPU. All the experiments run in the Windows platform using Anaconda virtual environment and Google Colab Notebooks [6]. The training and testing data is 300 and 269 sample points consisting of 32 features. For simplicity, we have converted the label into two classes -1 and 1, hence it becomes the binary classification problem.

Experiments

We perform experiments on the dataset and compare it with the existing implementation available from the Scikit-learn library [3] and we also compared the result with the SVMs for a better understanding of the difference. First, we have implemented the algorithm given by Freund *et al.* [2] which is our custom implementation of AdaBoost. In which for building *Decision Stumps* we created own several functions and based on that we train our AdaBoost classifier for prediction on the test set. Figure 1 shows the error curve of our custom AdaBoost classifier trained on the training set and tested on the testing set. We observe the test error of **0.060594** for our custom implementation.

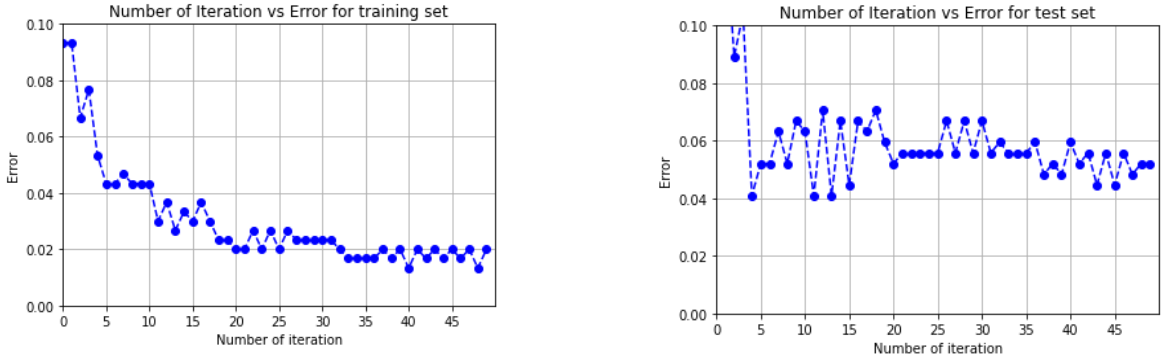

Figure 1. **Error Curve for training and testing set** for our custom AdaBoost Classifier implemented from the Freund *et al.* [2]. We also have predicted the values of the test set which have been attached in the supplementary material as well as in the notebook checkpoints. The graph has been plotted using Matplotlib [9] library.

After implementing the custom AdaBoost Classifier, we have used the available implementation provided by the open-source library scikit-learn [3]. As the provided implementation has various parameters, we fixed several parameters in order to compare the result from our custom implementation and scikit-learns open-sourced implementation, which can be shown in Table 1.

Table 1. A **fixed-parameter** in scikit-learn AdaBoost Classifier for better comparison.

Parameters	**Value**
Decision Tree Classifier - Max Depth	1
Number of Epoch	50
Number of Batch	30
Number of Estimators	30
Algorithm	SAMME

The tuning parameter for our case is the learning rate which we varied from 1 to 0.001 (1, 0.1, 0.01, and 0.001). Figure 2 showcases the error curve of open-sourced scikit-learns AdaBoost Classifier trained on the training set and tested on testing set having different learning rates. Table 2 shows the training and testing error rate for different learning rates and from the observations, it seems that learning rate 1 has the lowest training as well as a testing error which is good for generalization.

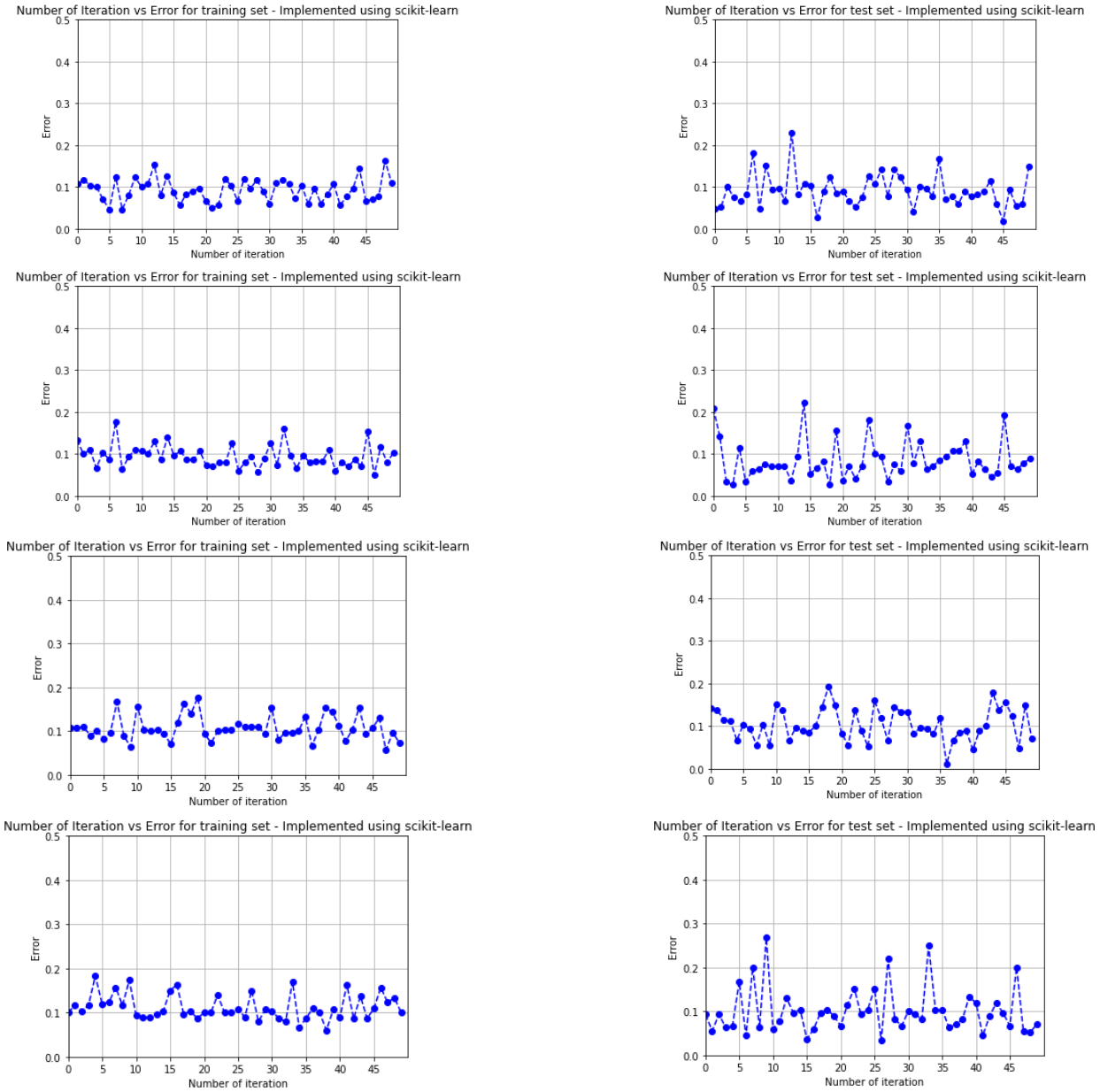

Figure 2. **Error Curve for training and testing set** for open-sourced implementation available of scikit-learns AdaBoost Classifier. From top to bottom, the learning rate reduces from 1, 0.1, 0.01, and 0.001. The training curve is on the left side, whereas the testing curve is shown on the right side. The graph has been plotted using Matplotlib [9] library.

Table 2. **Training and Testing Error** for different learning rates from 1 to 0.0001. From the observations learning rate 1 has the lowest training and testing error.

Learning Rate	Training Error	Testing Error
1	0.0868	0.081561
0.1	0.094933	0.085799
0.01	0.107466	0.112342
0.001	0.112266	0.101040

After successfully completing the analysis from the custom and scikit-learns implementations, our last experiments focus on the comparison with the SVMs. Here as we already completed the SVM's custom implementation in Assignment – 1 [7], and by taking permission from the professor, we use existing scikit-learns available SVMs implementation. In which we use '**linear**' Kernel and set the regularization parameter to '**1**'. We also tuned the regularization parameter and checked the test error performance and reported in Table 3.

Table 3. **Test error rate** for different regularization parameter in SVMs.

Regularization Parameter	Test Error
1	0.04460
5	0.05204
10	0.05576
25	0.04089
50	0.04089
100	0.05204

At last, we finally report the total duration for the training + testing time and observe the comparison for these three implementations which is shown in Table 4.

Table 4. **Training + Testing time** for different implementations.

Algorithms/Implementations	Total Time
Custom Implementation	27.4s
Scikit-learns Implementation	52s
Scikit-learns SVMs Implementation	4.16s

Based on the above experiments and observations, we discuss some important notes in this sub-section. Considering the training volume is very large and computational time is a concern then SVMs are a good choice as AdaBoost takes more time to compare to SVMs. AdaBoost can simply learn non-linear decision boundaries when the data provided is not separable by a linear line. However, as SVMs maximizes the margin in between the different point clouds, thus for linearly separable data SVMs outperform the AdaBoost by a large margin, since SVMs can be easily generalized. The reason AdaBoost is used because sometimes it is fast enough for the large-scale datasets and because of that when the deep learning was not that hyped the object detection was performed via Boosting by Viola *et al.* [8].

References:

[1] Cortes, Corinna, and Vladimir Vapnik. "Support-vector networks." Machine learning 20.3 (1995): 273-297. 1

[2] Freund, Yoav, Robert Schapire, and Naoki Abe. "A short introduction to boosting." Journal-Japanese Society For Artificial Intelligence 14.771-780 (1999): 1612. 1, 2, 4

[3] Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830. 1, 4

[4] Street, W. Nick, William H. Wolberg, and Olvi L. Mangasarian. "Nuclear feature extraction for breast tumor diagnosis." Biomedical image processing and biomedical visualization. Vol. 1905. International Society for Optics and Photonics, 1993. 1, 3

[5] Chen, Tianqi, and Carlos Guestrin. "Xgboost: A scalable tree boosting system." Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining. 2016. 3

[6] Bisong, Ekaba. "Google Colaboratory." Building Machine Learning and Deep Learning Models on Google Cloud Platform. Apress, Berkeley, CA, 2019. 59-64. 3

[7] https://github.com/Vanditg/COMP-SCI-7314---Introduction-to-Statistical-Machine-Learning/tree/master/Assignment%20-%201/Code 6

[8] Viola, Paul, and Michael Jones. "Rapid object detection using a boosted cascade of simple features." Proceedings of the 2001 IEEE computer society conference on computer vision and pattern recognition. CVPR 2001. Vol. 1. IEEE, 2001. 7

[9] Hunter, John D. "Matplotlib: A 2D graphics environment." Computing in science & engineering 9.3 (2007): 90-95. 4, 5