Assignment 2
COMP SCI 7209 - Introduction to Statistical Machine Learning
AdaBoost implementation

Julian Cabezas Pena
Student ID: a1785086
University of Adelaide, SA 5005 Australia
julian.cabezaspena@student.adelaide.edu.au

1. Introduction

One of the most common applications of machine learning techniques is their utilization in classification problems, where an algorithm takes a vector or array of input data, and assigns it to one or many discrete classes [1]. These algorithms usually depend on the availability of labelled data ans its corresponding attributes or explanatory variables, and thus are classified in the field of supervised learning [2].

Boosting is based on the simple premise of combining the output of several weak classifiers, such as simple decision trees,to generate a most accurate "committee" . In these kind of algorithm, a set of weak learners are trained in sequence result [2]. One of the most common boosting algorithms is the Adaptative Booting (AdaBoost), that iteratively calls a base or weak algorithm, that is fitted on the dataset, in each of the iterations the algorithm is trained over a different set of weights on a defined distribution, that are ajusted to assign a larger weight to the misclassified samples in the previous iteration [3].

In the field of classification problems, the diagnosis or determination of he risk of diseases based on clinical data is a recurrent field of study. These classification algorithms can help decision makers to predict the patient outcome based on the patient data [4], making appropriate and well-timed decisions. One of the diseases that has been researched in this field is the breast cancer, that is one of the most common cancers along with the lung, bronchus, prostate, colon and pancreas cancers [5].

The objective of this project is to implement the Adaptative Boosting (AdaBoost) method using a simple decision stump as weak learner. The implemented methods will be compared with the AdaBoost method implemented in the commonly used Scikit-learn package, and also with the Support vector Machine(SVM) algorithm. The testing and comparison will be performed using the Wisconsin Breast Cancer Dataset to predict whether the breast cancer is malign or benign based on a set of observed attributes

2. Methods

2.1. Wisconsin Breast Cancer Dataset

The Wisconsin Breast Cancer Dataset was created by Street *et al* [6] using a expert personal interpretation of image over a interactive interface to delineate the shape of nuclei of malign and benign breast cancer imagery, extracting features related to the nuclear size, shape and texture. This data consists in a total of 569 samples with 30 numerical continuous features. This dataset is frequently used to lustrate classification problems, determining weather the observed patient presents a malign or benign tumour based on its recorded characteristics [5]

In order to train the models, the first 300 samples of the dataset were used as trainign set and the remaining 269 as test data. The only preprocessing that was performed was the encoding of the target variable, that magign (M) tumours were encoded ad +1 and benign cancers as +1

2.2. AdaBoost custom implementation

In this paper, the implementation of the AdaBoost algorithm follows the steps described by Freund and Schapire [3]. If we have a matrix of features X containing n observations and a label y also containing n observations, and we have that $y = -1, +1$. We can train M weak or base learners ($G(x)$).

Firstly, the weights are initialized as $W_1 = 1/n$.

Then for each iteration of the algorithm $m = 1, 2, 3...M$:

- Train a weak learner $G_m(x)$ using the weights W_m
- Calculate the error using:

$$err_m = \sum_{i=1}^{N} W_i I(y_i \neq G_m(x_i)) \tag{1}$$

- Calculate the α value as

$$\alpha_m = \frac{1}{2} ln(\frac{1 - err_m}{err_m}) \tag{2}$$

-Update the weights of the samples using the following equation, that results in the weights adding a total of 1:

$$W_{m+1}(i) = \frac{W_m exp(-\alpha_m y_i g_m(x_i))}{\sum_{i=1}^{N} W_i} \tag{3}$$

Then, to generate the prediction using the ensemble of weak learners, the α of each iteration acts like the weights of each learner, as:

$$G(X) = sign(\sum_{m=1}^{M} \alpha_m g_m(x)) \tag{4}$$

Thus, the output of the AdaBoost algorithm can can is -1 or +1

2.2.1 Weak learner: Decision Stump

In order to train the AdaBoost algorithm, a base learner, that is usually slightly better than random guessing, has to be applied [3]. In this case, a simple decision stump was used. The decision stump is supervised algorithm that involves using a single feature to classify the sampling a threshold that minimizes the error or the amount of misclassifications in the data [7], they can be also interpresed as desicion trees of depth equal to one. this kind of weak learner is commonly used in implementations of AdaBoost [2]

In this case, to construct the decision tree, a greedy method was implemented. The decision stump algorithm goes feature by feature testing a number of threshold values equivalent to the number of samples, starting from the minimum value and finishing in the maximum value, the threshold values that are tested are evenly distributed between these values. Aditionally, the threshold values are tested using two different polarities (classifying the samples to -1 or +1 depending on the side of the threshold they are located)

The error of the classification, used to pick the best threshold, polarity and feature, was measured using the following equation [3]

$$err = \sum_{i=1}^{N} W_i I(y_i \neq g(x_i)) \tag{5}$$

Where $g()$ is the weak learner and w the set of weights in the corresponding iterations

2.3. Third Party AdaBoost Implementation: *Scikit-learn*

In order to compare the custom AdaBoost implemented in this paper with a commonly used library implementation, a third party library was used to solve the classification problem. In this case, the Scikit-learn library was chosen due to its wide popularity in the machine learning community.

Scikit-learn is a Python library that integrates various tools for statistics and machine learning applications, that include classification, regression, metrics and feature selection, among many others. This library is distributed under a BSD licence and includes compiled code, that makes it very efficient. The library is built using other popular numerical Python libraries, such as *Numpy* and *Scipy* [8].

According to the library documentation, the AdaBoost classification algorithm that is implemented in this package use the Multi-class Adaboost algorithm variation introduced by Zhu *et al* [9]. In this algorithm, the authors modified the originally proposed two class AdaBoost [3] to include multiple class problems without separating the problem into several two class problems.

In this algorithm, called SAMME, the procedure allows for the classification problem to deal with a set of K distinct classes in the target variable. As follows

[9]:

Firstly, the weights are initialized as $W_1 = 1/n$.

Then for each iteration of the algorithm $m = 1, 2, 3...M$:

- Train a weak learner $G_m(x)$ using the weights W_m
- Calculate the error using:

$$err_m = \frac{\sum_{i=1}^{N} W_i I(y_i \neq G_m(x_i))}{\sum_{i=1}^{N} W_i} \tag{6}$$

- Calculate the α value as

$$\alpha_m = ln(\frac{1 - err_m}{err_m}) + ln(K - 1) \tag{7}$$

-Update the weights of the samples using the following equation, that results in the weights adding a total of 1:

$$W_{m+1}(i) = \frac{W_m exp(-\alpha_m y_i g_m(x_i))}{\sum_{i=1}^{N} W_i} \tag{8}$$

Then, to generate the prediction, the maximum number on each of the k categories is used

$$G(X) = argmax(\sum_{m=1}^{M} \alpha_m g_m(x) I(g_m(x) = k)) \tag{9}$$

In this case, as the SAMME algorithm is being used for a two class classification problem ($K = 2$), the term $ln(K + 1)$ in equation XXXXXX becomes zero, making the calculation of $/alpha$ the same as in the custom AdaBoost implemented in this paper.

In order to make a comparison y similar terms, the base learner that was applied is the decision tree implemented in the *Scikit-learn* package, using a depth of 1, making the learner a decision stump in practice.

2.4. Comparison with Support Vector Machine

The Adaboost implementations were compared with the Support Vector Machine classification algorithm. This supervised classification algorithm was developed by Vapnik [10], and consist in an hyperplane that separates a high dimensional space defined by the input variables into discrete classes. This hyperplane is defined to have the largest possible distances to the closest point of either class , thus, maximizing the margin between two classes. The Support Vector Machine has been extensively used in classification problems, as it can include a custom kernel, that can handle non linearly separable cases [2].

In this study, the AdaBoost methods were compared with the *Scikit-learn* package implementation of the Support Vector Machine classifier, using different kernel functions to better adjust the data.

2.5. Train and testing

In order to asses the error in the training ans testing data for the AdaBoost implementation, different number of learners from 1 to 500 were tested, and the train and test errors were calculated as:

$$Error = \frac{FP + FN}{TP + FP + TN + FN} \tag{10}$$

Where TP is the number of true positive, TP: true negative, FP: false positives and FN: false negatives.

In the case of the SVM, the algorithm was tested using different cost values from 0.5 to 10, and testing a linear, polynomial, radial and sigmoid kernels.

2.6. Code

The code to reproduce this project was attached to this report. The primal and dual implementations of the models were programmed in Python 3.7 using only commonly used libraries such as numpy for numerical computation, pandas for the reading of databases and copy and os for general utilities.

As the testing of different number of learners in the case of the AdaBoost algorithm, or the testing of several C values and kernels in the case of SVM can take several hours (depending of the computing power of the equipment), the codes of the different implementations contain the saving of the results into a .csv file, that contains the results of the different parameter testing

The instructions to run the attached codes can be found in Appendix 1, or in the accompanying README.txt file.

3. Results and discussion

4. Analysis of the custom implementation

The custom implementation of the AdaBoost algorithm was tested using 1 to 500 iterations, giving the results that are shown in Figure 1, where it is possible to appreciate that the training value reaches an error

equal to zero at 30 iterations, while the smaller test error (1.49%) is archived with 100 iterations. After this point the custom algorithm results show a slight overfitting effect, presenting larger test error with more iterations, and reaching 3.35% with iteration 400.

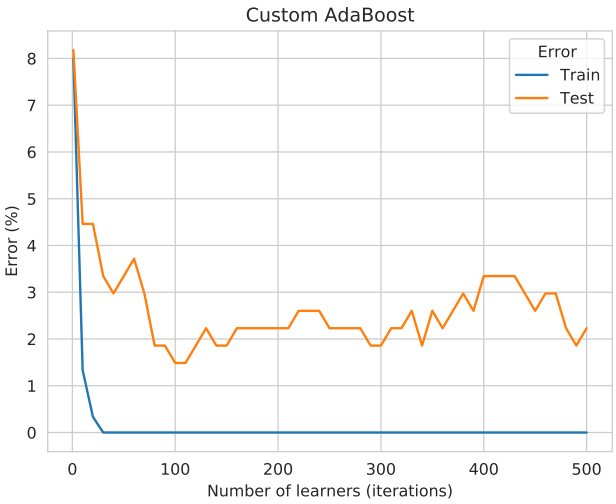

Figure 1. Training and testing error with different number of weak learners using the AdaBoost implementation of this paper (custom) and the third party implementation (Sklearn)

4.1. *Scikit-learn* AdaBoost

The *Scikit-learn* implementation of the AdaBoost algorithm, that uses the SAMME algorithm, similarly to the custom implementation, reach a minimum test error of 1.86% using 110 iterations, and reaching 0% train error with 30 iterations (Figure 2), just as in the custom implementation. After reaching the minimum text error, the *Scikit-learn* implementation do not show a significant overfitting effect, not presenting errors greater than 3% after iteration 140.

4.2. Support Vector Machine

The results of the *Scikit-learn* Support Vector Machine implementation (Figure 3) show that in this case the best results overall results are presented by using a linear kernel. In this case, the smaller test error (4.09%) is accomplished using a cost equal to 2.0 on the linear kernel, while the the radial and polynomial kernel reach worse results, in general above 5% of error.

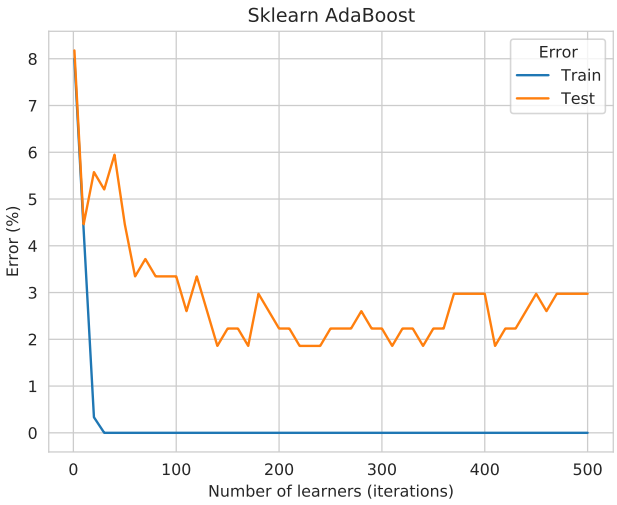

Figure 2. Training and testing error with different number of weak learners using the AdaBoost implementation of this paper (custom) and the third party implementation (Sklearn)

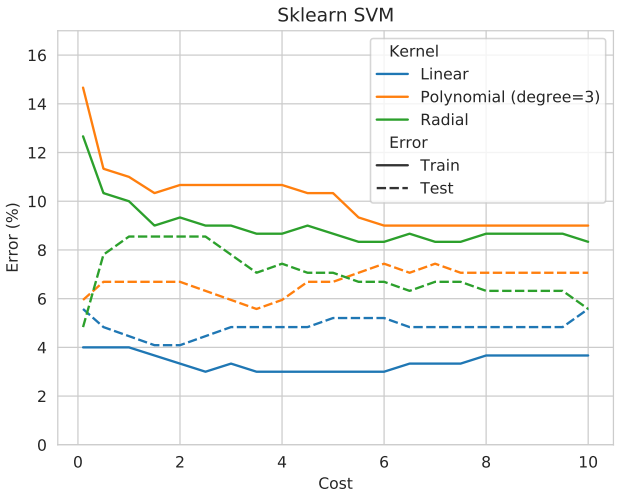

Figure 3. Training and testing error using different cost and kernel values

4.3. Comparison of AdaBoost implementations and SVM

The best test error results (Table 1), show that the custom implementation reached a slightly smaller test error, that can be attributed to the greedy approach that was used to code the decision stump, that also caused the algorithm perform several times slower than the third party implementation

On the other hand, the SVM algorithm, despite test-

ing several cost values and types of kernels, did not acomplished to accurately predict the breast cancer class.

Algorithm	Best test error	Training time
Custom AdaBoost	1.49%	0.9764
Scikit-learn AdaBoost	1.86%	0.9774
Scikit-learn SVM (linear kernel)	4.09%	0.9764

Table 1. Best test results for the implemented algorithm

5. Conclusion

This project showed that with enough understanding of the principles and processes behind a maximum margin classifier such as SVM, it is possible to implement an effective classification algorithm using convex optimization libraries. The implemented methods produced almost equal results when compared with third party implementations, showing that it is possible to program complex algorithms that can produce accurate results when the mathematics behind it are well understood.

References

[1] C. M. Bishop, *Pattern Recognition and Machine Learning*. New York, NY, USA: Springer, 2006.

[2] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*. 2009.

[3] Y. Freund and R. E. Schapire, "A Short Introduction to Boosting," *Journal of Japanese Society for Artificial Intelligence*, vol. 14, no. 5, pp. 771–780, 1999.

[4] R. Bellazzi and B. Zupan, "Predictive data mining in clinical medicine: Current issues and guidelines," 2008.

[5] A. F. M. Agarap, "On Breast Cancer Detection: An Application of Machine Learning Algorithms on the Wisconsin Diagnostic Dataset," in *ICMLSC '18 Proceedings of the 2nd International Conference on Machine Learning and Soft Computing*, (New York, NY, USA), pp. 5–9, Association for Computing Machinery, 2018.

[6] W. N. Street, W. H. Wolberg, and O. L. Mangasarian, "Nuclear Feature Extraction For Breast Tumor Diagnosis," in *International Symposium on Electronic Imaging? Scienceand Technology*, vol. 1905, pp. 861–870, 1993.

[7] J. J. Oliver and D. Hand, "Averaging over decision stumps," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 784 LNCS, pp. 231–241, 1994.

[8] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[9] J. Zhu, H. Zou, S. Rosset, and T. Hastie, "Multi-class AdaBoost," *Statistics and its interface*, vol. 2, pp. 349–360, 2009.

[10] V. N. Vapnik, *The Nature of Statistical Learning Theory*. New York, NY, USA: Springer, 1995.

Appendix 1: Instructions on how to run the attached code:

This code was tested under a Linux 64 bit OS (Ubuntu 18.04 LTS), using Python 3.7.7

In order to use this code:

1. Install Miniconda or Anaconda 2. Add conda forge to your list of channels

In the terminal run:

```
conda config --add channels conda-forge
```

3. Create a environment using the requirements.yml file included in this .zip:

Open a terminal in the folder were the requirements.yml file is (Assign1-code) and run:

```
conda env create -f requirements.yml --name svm-env
```

4. Make sure the folder structure of the project is as follows

```
Assign1-code
 Input_Data
 Cross_Validation
 Results
 support_vector_machine_primal_dual.py
 support_vector_machine_sklearn.py
 ...
```

If there are csv files in the Cross_Validation folder the code will read them to avoid the delay of the cross validation and go straight to fitting the models

5. Run the code in the conda environment: Open a terminal in the Assign1-code folder and run

```
conda activate svm-env
python support_vector_machine_primal_dual.py
```

or run the support_vector_machine_primal_dual.py code in your IDE of preference, (VS Code with the Python extension is recommended), using the root folder of the directory (Assign1-code) as working directory to make the relative paths work.

6. For comparison, run the code of the SVM implementation in Scikit-learn

```
python support_vector_machine_sklearn.py
```

Note: Alternatively, for 2 and 3 you can build your own environment following the package version contained in requirements.yml file