An Exploration of Boosting For Classification

Nadim El Helou
Boston University
44 Cummington Mall, Boston, MA 02215
nadimh@bu.edu

Karim Khalil
Boston University
44 Cummington Mall, Boston, MA 02215
karimk@bu.edu

John Neal
Boston University
44 Cummington Mall, Boston, MA 02215
jsneal@bu.edu

Abstract

Ensemble learning, specifically Boosting, can be used to form a strong classifier from multiple weak learners, while still maintaining a certain level of computational simplicity. This paper examines Boosting, specifically AdaBoost, and presents experimental results of its implementation for the binary classification problem on synthetic datasets with 2 features and a real dataset with a 60 features. Our experimental results suggest that Boosting can produce a strong classifier regardless of whether the dataset is linearly separable and regardless of the number of features.

1. Introduction

A large number of techniques and algorithms have been developed to approach classification problems in supervised machine learning, many of which can be powerful yet very complex requiring very high computing power. Consequently, ensemble learning has been gaining in popularity since it is able to obtain an improved level of efficiency and accuracy by combining several weak learners instead of one very complex learner. The focus of this paper will be the utilization of sequential learning – Boosting, for classification in supervised learning.
The most widely used weak learner for Boosting is the decision stump. One of the main goals of this paper is to explore the ability of Boosting to use a weak linear classifier – the decision stump, to best classify separable datasets (separable but not by a hyperplane). One example of a strong learner that effectively predicts non-linearly separable data is the Support Vector Machine Kernel Trick, which maps data points into a higher dimensional space where they become linearly separable even though the data points are not linearly separable in their original feature space.

2. Literature Review

The main source for mathematical and conceptual understanding of Ensemble Learning, Boosting, Bagging, and AdaBoost was [4]. [5] served as a good introduction to the mathematical and conceptual understanding of AdaBoost. [2], and [6] are YouTube videos that provide simple and intuitive descriptions of Boosting, Bagging, and AdaBoost, but lack the mathematical formulation. [3] provided the pseudocode that we used to write our implementation of AdaBoost in MATLAB, though the pseudocode leaves much to be interpreted and was only understood after consulting our other sources. [1] also provided a simple understanding of ensemble learning, though it was not used as a primary source.

3. Problem Formulation and Solution Approaches

3.1 Background

Boosting addresses an abstract machine learning problem: can a collection of weak classifiers be used to generate a strong classifier? Weak classifiers tend to be less computationally complex than strong classifiers. Ideally the hope is that the strong classifier built out of weak classifiers is less computationally complex than a simply strong classifier while providing good performance.

There are multiple Boosting algorithms that are used including XGBoost, CatBoost, LogitBoost, LPBoost, and AdaBoost (Adaptive Boosting). We chose AdaBoost as an introduction to understanding Boosting and its implementation in code. Before a discussion of AdaBoost it is necessary to provide the conceptual framework that contains it. It is necessary to explain Ensemble Learning, Bagging, Boosting, and Decision Trees.

Ensemble Learning is a machine learning technique where a model is formed from the sum of weighted combinations of base (think *basis*) models.

Formal Definition of Ensemble Learning:

$$h(y|\boldsymbol{x},\boldsymbol{\theta}) = \sum_{j=0}^{M} \alpha_j h_j(y|\boldsymbol{x}) \quad (1)$$

y : label, $\boldsymbol{x}$: feature vector, α_j : weight corresponding to base model h_j, $\boldsymbol{\theta}$: $(\alpha_1, \dots, \alpha_M)$

The weights α_i are tuning parameters that can be learned from LOOCV:

$$\hat{\alpha} = \underset{\alpha}{argmin} \sum_{i=1}^{N} L(y_i, \sum_{j=1}^{M} \alpha_j \hat{h}_j(y_i|\boldsymbol{x}_i)) \quad (2)$$

Ensemble learning is a type of Adaptive Basis Model, where a non-linear model is created by learning basis functions from the data (in this case each $h_j(y|\boldsymbol{x})$). Clearly this is useful when the dataset is not linearly separable. Two common examples of ensemble learning are Bagging and Boosting.

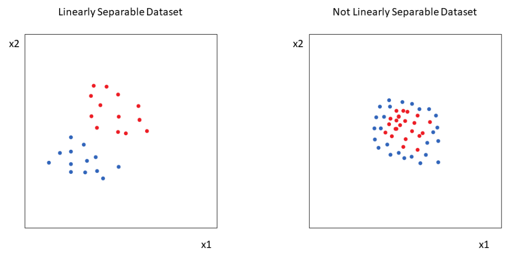


Figure 1: Example of Linearly Separable dataset and Not Linearly Separable dataset.

Now for a typical test-train split of the data set:

$$\begin{gathered} D = \{(\boldsymbol{x}_1, y_1), \dots, (\boldsymbol{x}_N, y_N)\} = \{d_1, \dots, d_N\} \\ s.t. \forall i \; D(i) = \frac{1}{N} \; (or \; d_1, \dots, d_N \; are \; i.i.d.) \\ where \; D_{Train} = \{d_1, \dots, d_{N_{(train)}}\} \\ s.t. \; \forall j \neq i, d_j \neq d_i \; and \; \therefore \; P(d_j = d_i) = 0 \\ where \; D_{Train} \in D \; and \; N_{train} < N \; (3) \end{gathered}$$

The principle of Bootstrap Aggregating (Bagging) is to improve the performance of a model by training *in parallel* a certain number of "weak learners" on a bootstrapped data set. For the binary case $(y \in \{-1, 1\})$ this looks like:

$$\begin{gathered} D = \{(\boldsymbol{x}_1, y_1), \dots, (\boldsymbol{x}_N, y_N)\} = \{d_1, \dots, d_N\} \\ s.t. \forall i \; D(i) = \frac{1}{N} \; (or \; d_1, \dots, d_N \; are \; i.i.d.) \\ D_{Train} = \{B_1, \dots, B_M\} \; where \; M = the \; total \; number \; of \; bags \\ where \; B_k = \{d_1, \dots, d_{N_{(B_1)}}\} \; s.t. \; \forall j, i \; P(d_j = d_i) \neq 0 \\ where \; \forall d_j \in D, d_j \in D_{train} \; (4) \end{gathered}$$

Then, "weak learner" models are produced in a parallel way during the training phase, one for each B dataset. During the testing phase, each model will make its predictions, and our final output is the majority vote:

$$h(y|\boldsymbol{x}) = \sum_{j=1}^{M} h_j(y|\boldsymbol{x}) \; where \; h_j \; is \; trained \; on \; B_j \; (5)$$

An example of Bagging is the Random Forest Algorithm.

Boosting, in contrast to Bagging, consists of *iteratively* training "weak classifiers" and adding them to generate a strong classifier, giving more emphasis on misclassified data points after each iteration. The training and testing is done using the entire dataset. At the first iteration of the algorithm, all training data points are assigned equal weights, and the algorithm then finds the best model of the "weak learner" for these data points:

$$\begin{gathered} D = \{(\boldsymbol{x}_1, y_1), \dots, (\boldsymbol{x}_N, y_N)\} = \{d_1, \dots, d_N\} \\ s.t. \forall i \; D_0(i) = \frac{1}{N} \; (or \; d_1, \dots, d_N \; are \; i.i.d.) \\ D_{Train} = \{B_1, \dots, B_T\} \\ where \; T = the \; total \; number \; of \; iterations \\ where \; B_t = \{d_1, \dots, d_N\} \\ s.t. \; \forall j \neq i, d_j \neq d_i \; and \; \therefore \; P(d_j = d_i) = 0 \; (6) \end{gathered}$$

In future iterations, all the previously misclassified data points will get increased weight:

$$\forall t > 0 \begin{cases} D_t(i) > D_{t-1}(i) \; if \; h_t(y_i|\boldsymbol{x}_i) \neq y_i \\ D_t(i) < D_{t-1}(i) \; if \; h_t(y_i|\boldsymbol{x}_i) = y_i \end{cases} (7)$$

The weak learners are sequentially produced during the training phase. The performance of the model is improved thanks to the weights adjustments:

$$h(y|\boldsymbol{x}) = \sum_{t=1}^{T} h_t(y|\boldsymbol{x}) \; where \; h_t \; is \; trained \; on \; B_t \quad (8)$$

$$\forall i, h(y|x_m) = \begin{cases} -1 \; if \; x_m \leq \tau_i \\ 1 \; if \; x_m > \tau_i \end{cases}$$

$$\tau_i = \underset{\tau}{argmin} \sum_{i=1}^{N} L[h(y|x_m)], \; and \; m \in \{1, \dots, d\} \; (9)$$

Geometrically, it can be imagined as a hyperplane orthogonal to one of the basis vectors that separates the feature space into two half spaces:

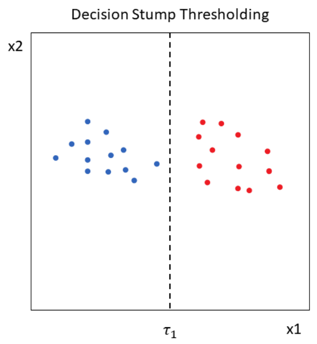


Figure 2: Example of a decision stump separating a dataset.

For a binary-valued feature the decision stump is merely an IF statement:

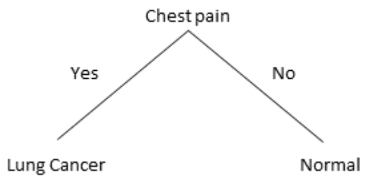


Figure 3: Picturing the decision stump as an IF statement.

AdaBoost is an example of boosting that uses decision stumps as the weak learners that form the model. These weak learners are weighted depending on their respective training errors at each iteration:

$$h(y|\boldsymbol{x}) = \sum_{t=1}^{T} \alpha_t h_t(y|\boldsymbol{x}) \; where \; \alpha_t = \frac{1}{2}\log\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right)$$

$$where \; the \; error \; \varepsilon_t = \sum_{i=1}^{N} D_t(i) \, 1(g_t(\boldsymbol{x}_i) \neq y_i)$$

$$where \; g_t(\boldsymbol{x}_i) = \underset{g_m, \forall m \in \{1,\ldots,d\}}{argmin} \sum_{i=1}^{N} D_t(i) \, 1(g_m(\boldsymbol{x}_i) \neq y_i) \quad (10)$$

The update equation for the weights depends on an exponential loss function:

$$D_{t+1}(i) = \frac{D_t(i)}{z_t} \; x \; \begin{cases} e^{\alpha_t} \; if \; g_t(\boldsymbol{x}_i) \neq y_i \\ e^{-\alpha_t} \; g_t(\boldsymbol{x}_i) = y_i \end{cases}$$

$$where \; z_t = 2\sqrt{\varepsilon_t (1-\varepsilon_t)} \; (for \; normalization) \, (11)$$

The algorithms implementation steps are presented in section 9.2 of the Appendix.

3.2 Analysis and Explanation of the AdaBoost Algorithm

The general idea of AdaBoost is to sum weighted weak classifiers (decision stumps for our implementation) in order to find a stronger classifier. The pseudo code in section 9.2 is intuitive yet we will elaborate on some key points at a high level and present mathematical analysis of the reweighing of points.

In step (a), all training data points are weighed equally, this is because we haven't iterated through our algorithm yet and all points are of equal significance for the first iteration.

In step (b), we calculate all possible weak classifiers. In our case, we used decision stumps and we had to calculate all possible decision stumps for our data. These weak classifiers will be used in part (c).

In part (c), we classify our weighted data using each weak classifier separately and choose the weak classifier that produces the smallest weighted error ε_t. The weighted error is obtained by summing the weights of all the misclassified points.

In part (d), we update our "sum of weak classifiers" by adding the term $\alpha_t * g_t$ to our classifier from the previous iteration F_{t-1}. The term weighs classifiers that produce a lower error greater and weighs classifiers that produce a higher error lower. The relationship between α_t and the weighted error ε_t is described by the following plot:

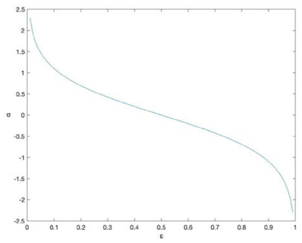


Figure 4: Plot of α_t as ε_t varies from 0 to 1

In part (e), the points of our datasets are associated new weights w_{t+1}. The following equation describes this step:

$$\forall i \in [1{:}n]: w_{t+1}(i) = \frac{w_t(i)}{z_t} \; x \; \begin{cases} e^{\alpha_t} \; if \; g_t(\boldsymbol{x}_i) \neq y_i \\ e^{-\alpha_t} \; g_t(\boldsymbol{x}_i) = y_i \end{cases}$$

$$where \; z_t = 2\sqrt{\varepsilon_t (1-\varepsilon_t)} \; (for \; normalization) \quad (11b)$$

To understand the reweighing equation, we present the following graph and analyze its results:

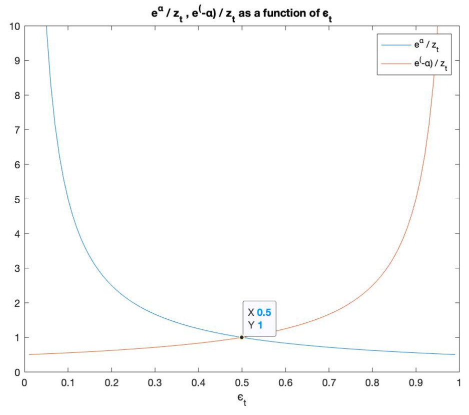


Figure 5: Coefficients of $w_t(i)$ that result from misclassification (blue) and correct classification (orange) as functions of from ε_t 0 to 1.

As described in the equation, incorrectly classified points' weights are scaled by a factor of e^{α_t}/z_t while correctly classified points' weights are scaled by a factor of $e^{-\alpha_t}/z_t$. These factors are only dependent on weighted error for that iteration (and are plotted as a function of ε_t above). As a result of analyzing this graph, we have made some significant observations:

(a). For $\varepsilon_t < 0.5$, the weights of incorrectly classified points are scaled up while the weights of correctly classified are scaled down.

(b). For $\varepsilon_t > 0.5$, the weights of correctly classified points are scaled up while the weights of incorrectly classified as scaled down.

(c). When $\varepsilon_t = 0.5$, weights of all points remain the same value.

At a high level, when the weighted error is less than 0.5, the algorithm will reweigh incorrectly classified points higher as to take them into consideration more during the next iteration of the algorithm.

When the weighted error is greater than 0.5, the current classifier's classifications are insignificant since the weighted error is less than 0.5 (might as well not use the weak classifier) so correctly classified points are reweighed higher in order to take them into consideration again during the next iteration of the algorithm.

When the weighted error is 0.5, the points are not reweighed as the classifier is insignificant (no different than a coin toss). During the next iteration, the previous classifier will produce the lowest weighted error as the points are not reweighed and it will be chosen yet again. Thus, if $\varepsilon_t = 0.5$ the choice of weak classifier will remain the same for all subsequent iterations:

$$g_t = g_{t+1} = \cdots = g_{t+n} \text{ and } \forall i \in [1:n]: w_t(i) = w_{(t+1)}(i) = \cdots = w_{t+n}(i)$$

4. Implementation

The AdaBooost implementation is split up into multiple functions, one for each step of the algorithm.

For the initialization, the first function *calculate_gs* takes in as input the entire dataset and returns all possible weak classifiers (in this case, decision stumps). These will then be analyzed at each iteration of the loop. Still in the initialization, all data points are assigned equal weights, and two vectors are initialized: one for storing each weak classifier chosen, and one for their coefficients. A decision stump can be 'stored' through three characteristic numbers: the feature it concerns, the threshold, and a third number that indicates which class lies on which side of the stump.

In the body of the algorithm, at each iteration of the 'for loop,' the function *calculate_best_g* finds the best decision stump at the current iteration – for the current set of weights. This decision stump is stored and its coefficient is calculated. That coefficient depends on the stump's error rate, and determines that stump's contribution in the final strong classifier. After that, the weight of each data point is updated depending on whether it was correctly classified by this classifier – misclassified points get assigned increased weights.

At the end of the loop, once given a test point, all the T classifiers that were stored are weighed and summed. The prediction of that test point is the sign of this sum. In the end, two vectors define the strong classifier: the one storing all the chosen decision stumps and the one storing their coefficients.

Another function is *test_our_boosted_classifier*, which calculates the correct classification rate for the strong classifier created.

5. Experimental Results

The synthetic datasets (see Appendix) created were used to test that implementation of the decision stumps and AdaBoost. The decision stumps only did well in the case where the dataset was linearly separable by a threshold in one dimension. Otherwise they were not much better than a coin toss meaning the test CCR was somewhere between .5 and .7.

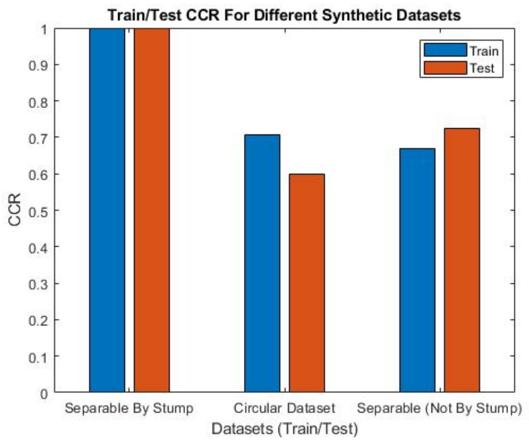


Figure 6: Test/Train CCRs of Optimal Decision Stump on synthetic datasets.

This implementation of AdaBoost involved some tuning parameters such as T, the number of iterations of the algorithm, which also determines the number of decision stumps that make up the final strong classifier. Therefore, the impact of T on the solution was analyzed and plotted in the following figure.

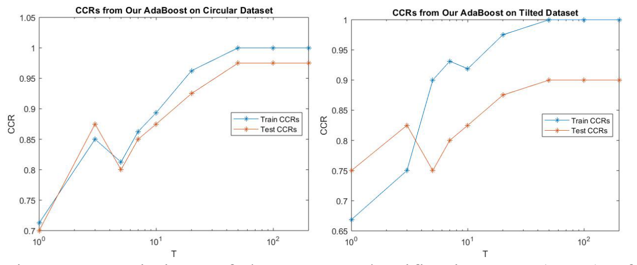


Figure 7: Variations of the correct classification rate (CCR) of the implemented AdaBoost on the synthetic datasets as a function of the number iterations (T) of the algorithm.

Compare to the Train/Test CCR plot of MATLAB's built-in AdaBoost (using fitcensemble and 'AdaBoostM1'):

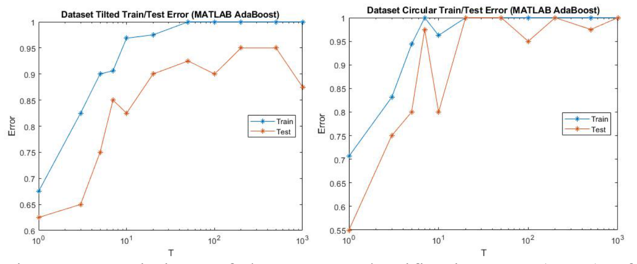


Figure 8: Variations of the correct classification rate (CCR) of MATLAB's built-in AdaBoost on the synthetic datasets as a function of the number iterations (T) of the algorithm.

Our implementation scores a maximum CCR of 1 compared to MATLAB's .95 on the circular dataset. On the "tilted" dataset MATAB scores a maximum CCR of 1 compared to our implementations .95. While the performance of MATLAB's function matches ours, MATLAB's implementation takes a matter of seconds while ours takes approximately 30 seconds. This is most likely due to not vectorising our code.

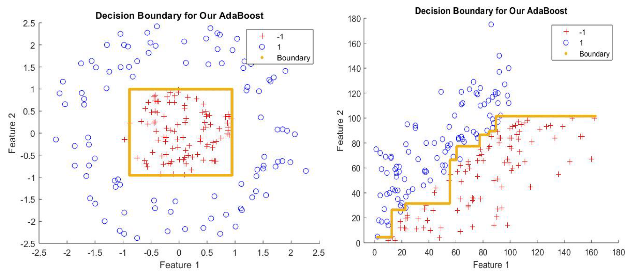


Figure 9: Decision boundaries of our AdaBoost implementation on our non-trivial synthetic datasets.

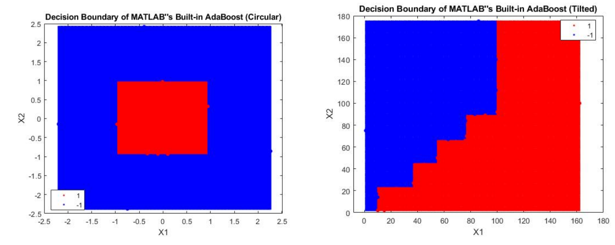


Figure 10: Decision boundaries of MATLAB's built-in AdaBoost.

We also implemented our AdaBoost and MATLAB's built in AdaBoost on the Sonar dataset (see Appendix for info on dataset):

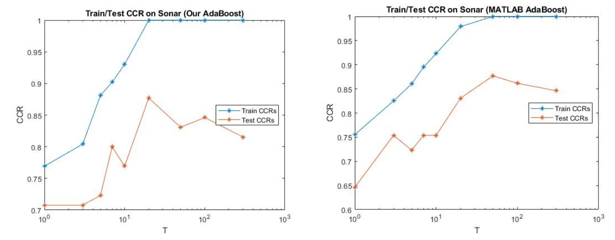
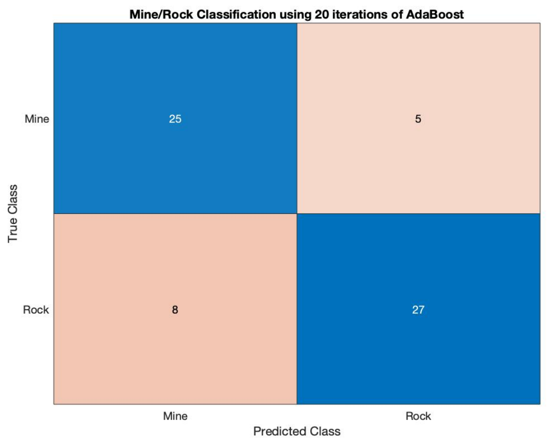


Figure 11: Test/Train CCR Plots for AdaBoost on Sonar Dataset. The left plot corresponds to our implementation and the right corresponds to MATLAB's build-in Adaboost.

Both our implementation of AdaBoost and MATLABs built-in AdaBoost score a maximum CCR of .87. What's clear is that the dataset overfits worse as T increases for both implementations which was not apparent in the CCR plots for the synthetic datasets. We still do not know for sure why overfitting occurs in this case but not in the other. Our initial suspicion was that the significant increase in the number of dimensions leads to greater divergence between test and train CCR.



	Mine	Rock
Mine	25	5
Rock	8	27

Figure 12: Confusion matrix for our implementation of AdaBoost on the Sonar dataset. 5 mines (~ 8 %) wrongly classified as rocks.

We also computed the train and test CCR for linear SVM using MATLAB's built-in SVM function (.85/.7 respectively) as well as SVM with an RBF kernel (.97/.75) to compare with AdaBoost. Admittedly, this was with blind knowledge of the dataset, but we wanted to compare AdaBoost with a "strong classifier." Interesting enough, linear SVM's CCR is not much better than a single decision stump.

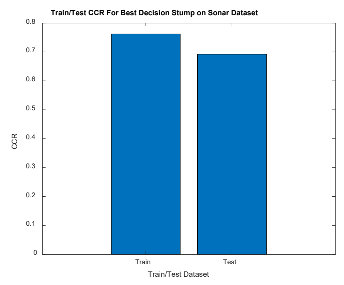


Figure 13: Train/Test CCR for best decision stump on Sonar dataset.

6. Conclusions

AdaBoost performed pretty well on the synthetic datasets even when the datasets were not linearly separable. It underperformed on the Sonar dataset when compared to the synthetic dataset. However, the test CCR did significantly improve by .18 relative to the weak classifier (decision stump).

AdaBoost was compared to MATLAB's linear SVM and kernel SVM with an RBF kernel on the Sonar dataset. At SVM's best, AdaBoost had .10 better Test CCR. The outcome of Linear SVM gives us the suspicion that the separating hyperplane may be the optimal decision stump. The desire was to compare AdaBoost to a strong classifer: however, in these implementation does not appear to be a strong classifier. Perhaps SVM can be used as one of the weak classifiers used in AdaBoost? Given more time we would have pursued this question further.

Another ancillary question was whether the higher dimensionality of the dataset contributed to overfitting on the Sonar dataset. We plotted the CCR of the circular dataset for k dimensions (1-10), and the results do not show that the dimensionality of the dataset contributed to overfitting. Ideally we would have plotted the circular dataset for 60 dimensions to be commensurate with the Sonar dataset, but the algorithm for generating this random dataset was taking far too long to be possible within time constraints. In the future we would like to find a better algorithm for creating a 60 dimensional synthetic dataset.

7. Description of Individual Effort

Very little work was completed outside of meeting together and working together. It is notable, however, that Nadim worked on some of the AdaBoost functions prior to our meeting where we completed our AdaBoost implementation together.

Regarding the report, John Neal wrote the literature review, and the conclusion. He also worked on the problem formulation and solution approaches along with Karim Khalil. Nadim El Helou worked on the introduction, the implementation; and also wrote the experimental results section with John Neal.

8. References

[1] Ben-David, Shai, and Shai Shalev-Shwartz. Understanding Machine Learning: From Theory to Algorithms. Cambridge University Press, 2014.
[2] edureka! "Boosting Machine Learning Tutorial | Adaptive Boosting, Gradient Boosting, XGBoost | Edureka." Youtube, 27 Jun. 2019.
[3] Jiao, Jiantao, et al. "Lecture 19: AdaBoost and Course Review." EE 378A: Statistical Signal Processing, 2016, web.stanford.edu/class/ee378a/lecture-notes/lecture19.pdf.
[4] Murphy, Kevin P. Machine Learning: a Probabilistic Perspective. MIT Press, 2013.
[5] SauceCat. "Boosting Algorithm: AdaBoost." Medium, Towards Data Science, 30 Apr. 2017, towardsdatascience.com/boosting-algorithm-adaboost-b6737a9ee60c.
[6] Starmer, Josh. [StatQuest With Josh Starmer]. (2019, January 14). AdaBoost, Clearly Explained [Video file]. https://www.youtube.com/watch?v=LsK-xG1cLYA
[7] Gorman, R. P., and Sejnowski, T. J. (1988). "Analysis of Hidden Units in a Layered Network Trained to Classify Sonar Targets" in Neural Networks,Vol. 1, pp. 75-89.

9. Appendix

9.1 Dataset Information

9.1.1. Synthetic Dataset:

We will generate three basic synthetic datasets:

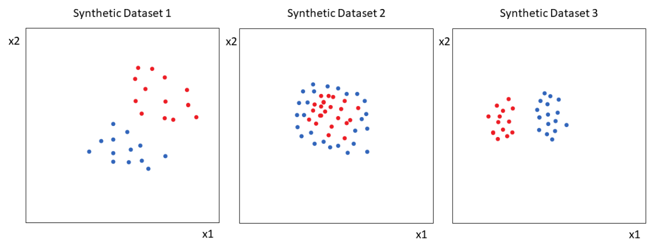


Figure 15: Picturing our three synthetic datasets for testing code.

9.1.2 Smallscale Dataset:

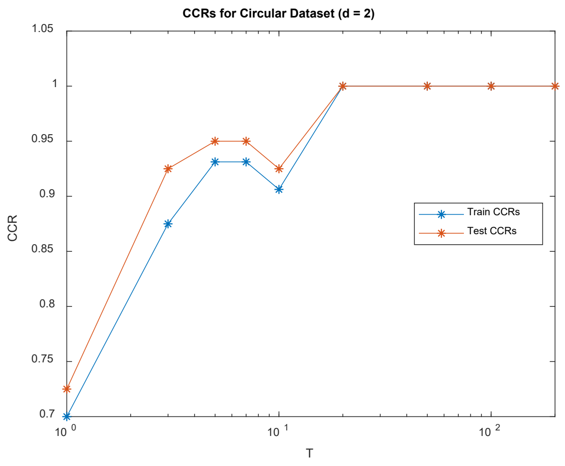
Smallscale dataset: Sonar Dataset - https://www.openml.org/d/40
Cited in [7]

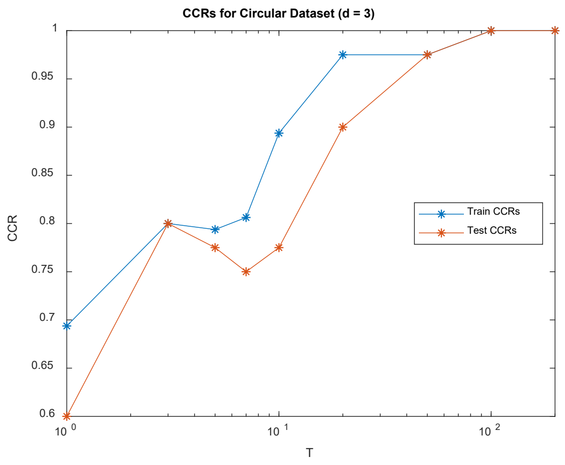
Binary Classification Problem
60 numerical (Float) features, 208 Samples, 86 kB

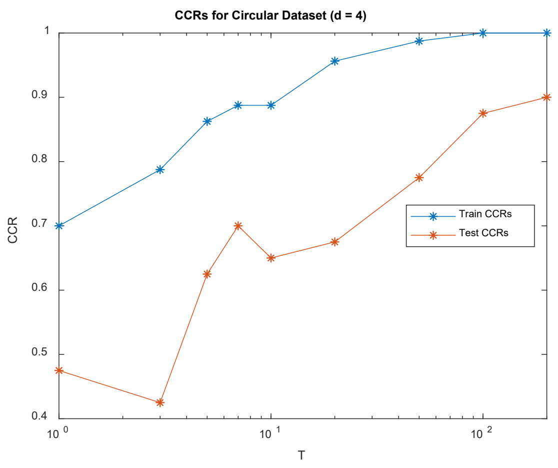
9.2 Algorithm Implementation

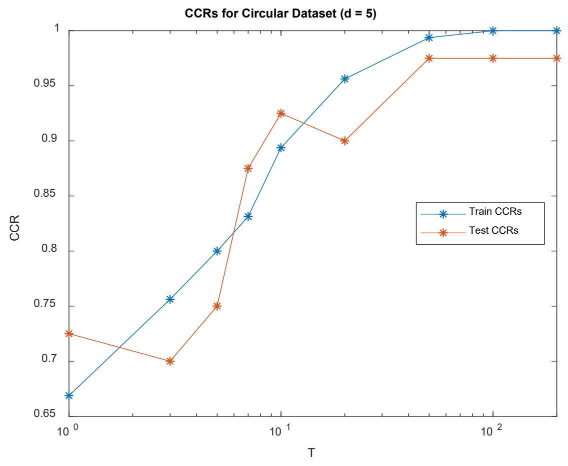
```
AdaBoost Algorithm

Given: $D = \{(\boldsymbol{x}_1, y_1), \dots, (\boldsymbol{x}_n, y_n)\}$
Initialization: $Weights\ w_0(i) = \frac{1}{n}\ \forall i\ [1, n]\ (equal\ weights\ for\ each\ data\ point)$    $(a)$
                $Classifier\ F_0(x) = 0$
                Calculate all possible weak classifiers $g \in G$    $(b)$
for t = 1, 2, ..., until stopping criterion T
    • Choose classifier that minimizes error:
    Choose classifier $g_t \in G$ such that $g_t = \underset{g \in G}{argmin} \sum_{i=1}^{n} w_t(i)\, 1(g(\boldsymbol{x}_i) \neq y_i) = \underset{g \in G}{argmin}\ \varepsilon_t$    $(c)$
    • Update sum of weak classifiers:
    $F_t = F_{t-1} + \alpha_t * g_t\ where\ \alpha_t = \frac{1}{2} \log\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right)$    $(d)$
Update weights on all points:
    $\forall i \in [1{:}n]: w_{t+1}(i) = \frac{w_t(i)}{z_t}\ x \begin{cases} e^{\alpha_t}\ if\ g_t(\boldsymbol{x}_i) \neq y_i \\ e^{-\alpha_t}\ g_t(\boldsymbol{x}_i) = y_i \end{cases}\ where\ z_t = 2\sqrt{\varepsilon_t\,(1-\varepsilon_t)}\ (for\ normalization)$$(e)$
end for
```

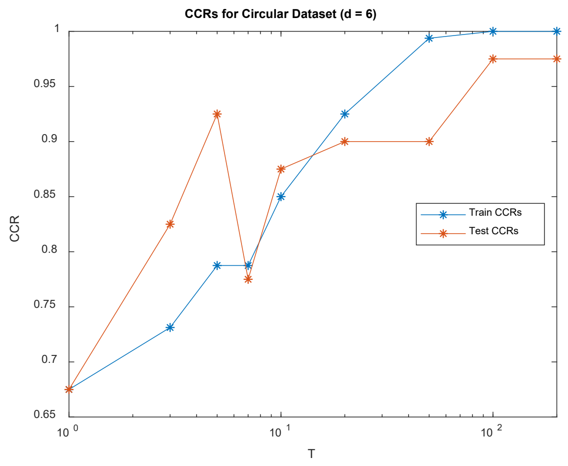
9.3 CCR Plots of Circular Dataset for Each k Dimension

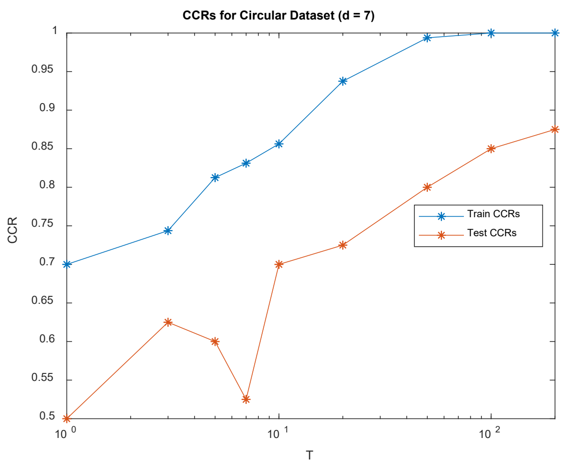


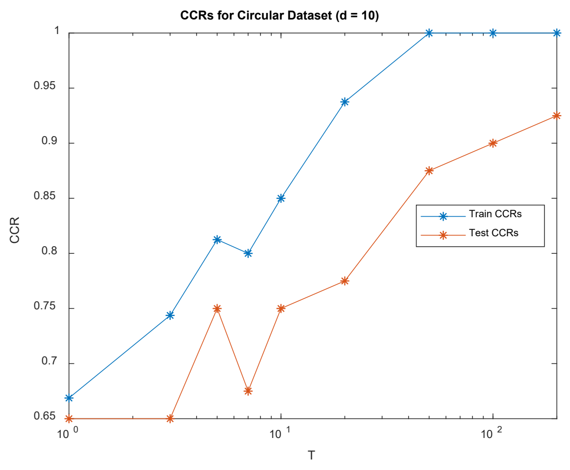


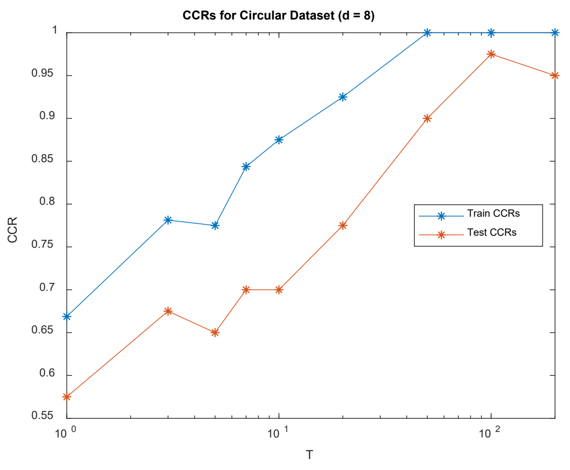


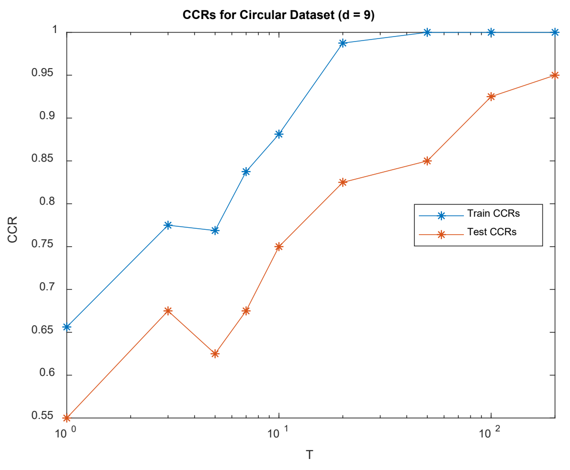





CCRs for Circular Dataset (d = 7)
CCR
T
Train CCRs
Test CCRs


CCRs for Circular Dataset (d = 10)
CCR
T
Train CCRs
Test CCRs


CCRs for Circular Dataset (d = 8)
CCR
T
Train CCRs
Test CCRs


CCRs for Circular Dataset (d = 9)
CCR
T
Train CCRs
Test CCRs