AI201 Programming Assignment 4
Comparison of Boosted Perceptrons and SVM

Anthon Van M. Asares

Date of Submission: 2024-11-25

1. INTRODUCTION

In the realm of supervised learning, classification algorithms play a vital role in tackling a wide array of problems, from image recognition to predictive analytics. Two widely studied and applied approaches are Support Vector Machines (SVM) and AdaBoost, a boosting-based ensemble method often adapted to enhance simple models like perceptrons. While both methods aim to achieve high accuracy, they do so through fundamentally different mechanisms.

Support Vector Machines operate by finding a hyperplane that maximizes the margin between classes, ensuring robust generalization to unseen data. The flexibility of SVMs is further enhanced by kernel functions, enabling them to effectively address both linear and non-linear classification tasks, even in high-dimensional spaces [Cortes and Vapnik, 1995].

AdaBoost, on the other hand, combines multiple weak learners—often perceptrons—into a strong classifier through an iterative process. By focusing on misclassified samples in each round, AdaBoost adaptively adjusts the weights of both the data points and the classifiers, yielding a powerful ensemble capable of tackling complex decision boundaries [Schapire, 2013].

This paper presents a comparative analysis of SVM and AdaBoost (boosted perceptron) algorithms, examining their theoretical frameworks, computational requirements, and performance on diverse datasets. The discussion highlights the strengths and limitations of each method, offering practical guidance for selecting the optimal approach in different machine learning scenarios.

2. OBJECTIVES

The primary objective of this report is to conduct a comprehensive comparison of the Boosted Perceptron (implemented using AdaBoost) and Support Vector Machine (SVM) algorithms in terms of their performance on the banana and splice datasets. The deliverables include the implementation of both algorithms in Python, detailed performance analysis through plots, and an evaluation of the impact of SVM kernel and parameter settings. Through this comparison, the report aims to highlight the strengths and limitations of each approach, providing insights into their suitability for different classification tasks.

3. METHODOLOGY

3.1 Perceptron Classifier Construction

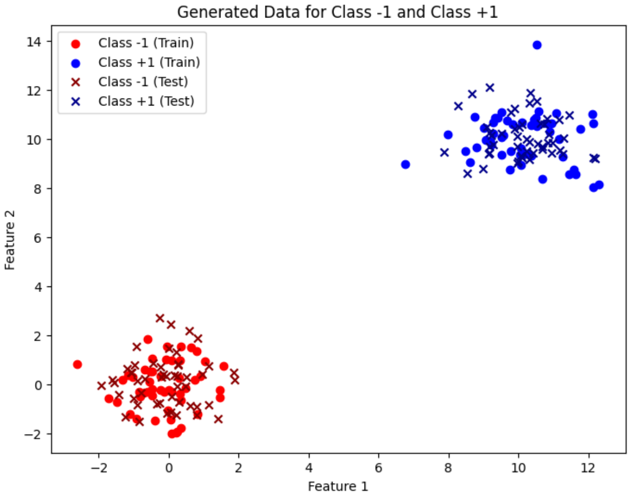

Figure 1: Synthetic data to start testing our perceptron classifier

A perceptron classifier was constructed, utilizing the Pocket Algorithm as its learning mechanism. A synthetic dataset was first created; two sets of 100 two-dimensional vectors were generated from independent normal distributions. The first set, representing class "-1," was sampled from a normal distribution with a mean vector of $\mu_1 = [0, 0]^T$, and a covariance matrix of $\sigma_1 = I$. The second set, representing class "+1," was sampled from a normal distribution with a mean vector of $\mu_2 = [10, 10]^T$ and a covariance matrix of $\sigma_2 = I$. Each data point in these sets was labeled according to its respective class.

For both classes, the datasets were divided into training and test subsets. Specifically, 50 points from each class were randomly assigned to the training set, while the remaining 50 points were allocated to the test set. The training subsets from both classes were then combined to form the final training dataset, and the test subsets were similarly merged to create the final test dataset. This process ensured a balanced representation of both classes in the training and test sets, which is illustrated in in Figure 1.

Then a function named `classify(···)` was written that implemented the Pocket Algorithm for perceptron learning. This function will train the perceptron on the given dataset

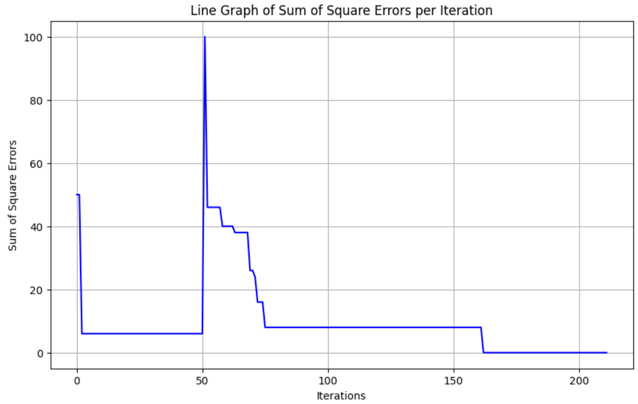

Figure 2: Sum of Square Errors for our perceptron stayed at 0 starting from somewhere around the 160th iteration and was stopped even before reaching the 10000th iteration.

by iterating through the training examples and updating the weights based on misclassifications. The algorithm will run for up to $maxitercnt = 10,000$ iterations, with the weight vector being updated during each iteration. The "pocket" aspect of the algorithm means that the best weight vector (in terms of minimizing the error) will be retained throughout the process.

Additionally, a `predict(···)` function will be developed to evaluate the classifier's performance on the test set. This function will apply the learned weights from the Pocket Algorithm to the test data, generating predictions for each test instance. To measure the accuracy of the model, the sum of squared errors (SSE) will be computed, which quantifies the difference between the predicted values and the actual labels in the test set.

As part of the initial implementation, a loop will be used from 0 to maxitercnt, and after each iteration, the sum of squared errors will be calculated and tracked. At the end of the process, a plot was generated to show how the sum of squared errors evolves over the iterations, providing a visual representation of the classifier's convergence and performance over time, which can be found in Figure 2.

3.2 Adaboost Construction and Evaluation

Suppose we have examples $S = \{(x_1, y_1), ..., (x_N, y_N)\}$ where we have $x \in R^d$, $y_i \in \{+1, -1\}$, while d is the number of features, and N is the number of training examples. The following algorithm was then implemented within `adabtrain`. With that said, the pocket algorithm was slightly modified to make gathering accuracy scores for different learners more efficient, and can be observed in Algorithm 1.

We also create an `adabpredict` function that classifies unknown/unseen data using the list of α and h from `adabtrain`. Specifically, `adabpredict` just functions as the ensemble classifier and is denoted in Equation 1. Both `adabtrain` and `adabpredict` were then used to train the boosted perceptron algorithm to measure the training and test accuracies, as well as their training and test times, for the ensemble classifier having K learners (Use $K = 10, 20, ..., 1000$). The training and test accuracies were then plotted for the banana and the splice datasets. The details about the number of training and test points for both datasets can be found in Table 1.

Algorithm 1 Training Algorithm xxx

```
if $w_t(i)$ is None then
    $t = 1$ as no weights were initialized yet and $w_i = 1/N$ (uniform distribution)
else
    Use the weights passed from previous K's run
end if
if No list of $\alpha_t$ is initialized then
    We initialize the list where we'll store different values of $\alpha$
else
    Use previous run with different K's list of $\alpha_t$
end if
if No list of $h_t$ is initialized then
    We initialize the list where we'll store different values of $h_t$
else
    Use previous run with different K's list of $h_t$
end if
for each t in (1, 2, ..., K) do
    Select new training set $S_t$ from $S$ with replacement according to $w_t$
    Train weak learner $L$ on $S_t$ to obtain hypothesis $h_t$
    Compute training error $\epsilon_t$ on $h_t$ on $S$
    Compute $\alpha_t$
    Compute the new weights on S which will then be used in the next iteration
end for
return list of $\alpha$, $h$, and $w$
```

Dataset	Train Points	Test Points
Banana	400	4900
Splice	1000	2175

Table 1: Total for banana dataset is 5300 while total for splice dataset is 2991. SMOTE was used to oversample some datapoints to achieve mutual exclusivity between train and test sets while also adhering to the instructions.

$$H(x) = sgn(\sum_{t=1}^{K} \alpha_t h_t(x)) \qquad (1)$$

3.3 SVM Classifier

The same partitioned datasets that were used in evaluating the boosted perceptron for evaluating the SVM classifier. nlike the construction of AdaBoost, which was implemented using only `numpy`, the `sklearn` package was utilized to instantiate the SVM model. Afterwards, `GridSearchCV` was then utilized to fetch the combination of kernel and parameters that gave the highest accuracy. The kernels and the parameters to be tested are outlined in Table 2.

Once the optimal combination of parameters has been fetched

Parameters	Values to Text
kernel	[linear, rbf, poly]
C	[0.1, 1, 10, 100]
gamma	[scale, auto, 0.01, 0.1, 1]
degree	[2, 3, 4]

Table 2: List of parameters to be fed into GridSearchCV to find the best combination that gives the highest accuracy.

Dataset	Kernel	C	Degree	Gamma
Banana	RBF	100	2	scale
Splice	RBF	10	2	auto

Table 3: The optimal params resulting from GridSearchCV for both the banana and splice datasets.

for SVM, we then train the model with those parameters and retest again using the banana and splice datasets. The accuracy was computed and the training and test times were determined.

4. EXPERIMENTAL RESULTS

In this section, we present the results of experiments conducted to evaluate the performance of the Boosted Perceptron (using AdaBoost) and Support Vector Machine (SVM) models on the `banana` and `splice` datasets. The objective is to compare the models' accuracy, training time, and performance across different configurations, including the number of learners K for AdaBoost and varying SVM kernel types and parameters.

The performance of the Boosted Perceptron on the `banana` dataset, for both training and test sets, is shown in Figure 3. Similarly, Figure 4 illustrates the corresponding performance on the `splice` dataset. For both datasets, it is evident that accuracy improves as the number of learners increases. However, for the `banana` dataset, accuracy plateaus after approximately 200 learners. In contrast, for the `splice` dataset, the test accuracy improves much more slowly than the training accuracy, indicating potential overfitting.

In addition to accuracy, Figure 5 presents the training and testing times. The results suggest that testing time—the time required to make predictions—remains relatively consistent and does not significantly vary with the number of learners. However, there is a clear linear relationship between the number of learners and training time, highlighting the computational cost of adding more learners to the ensemble.

The results obtained from using `GridSearchCV`, with the parameter values outlined in Table 2, are summarized in Table 3. Interestingly, the `banana` and `splice` datasets share the same kernel type and degree, but differ in their optimal values for C and γ. With these optimal parameters identified, we proceed to train the SVM models for both datasets and evaluate their performance. The results are presented in Table 4 for the `banana` dataset and Table 5 for the `splice` dataset.

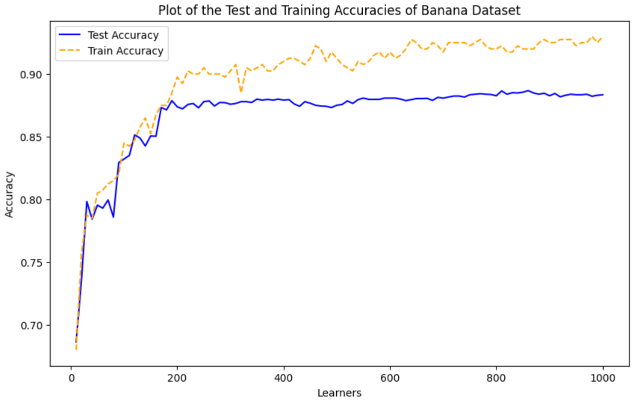

Figure 3: Performance of the boosted perceptron with different number of learners on the banana dataset.

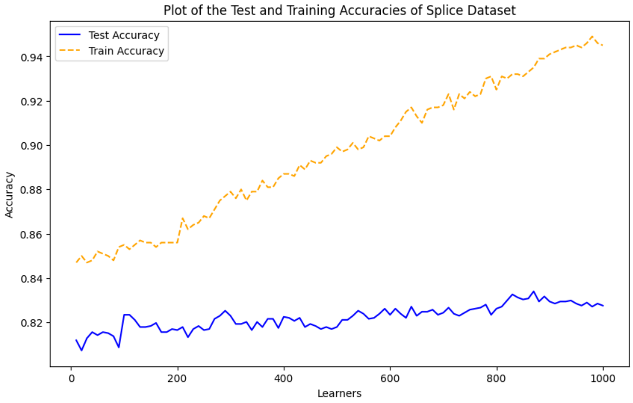

Figure 4: Performance of the boosted perceptron with different number of learners on the splice dataset.

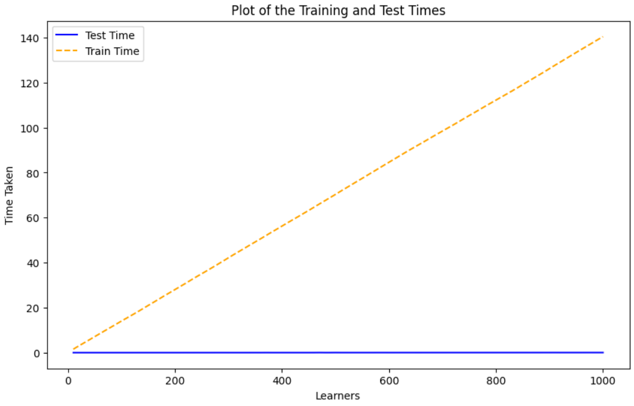

Figure 5: Time taken to complete training and testing per k learners. The plot for both datasets are nearly identical.

Classifier	Train Time	Test Time	Accuracy
Adaboost	140.33 s	0.027 s	88.35%
SVM	0.04 s	0.0012 s	89.2 %

Table 4: List of parameters to be fed into GridSearchCV to find the best combination that gives the highest accuracy.

Classifier	Train Time	Test Time	Accuracy
Adaboost	140.40 s	0.034 s	82.76%
SVM	0.04 s	0.0012 s	89.52 %

Table 5: List of parameters to be fed into GridSearchCV to find the best combination that gives the highest accuracy.

5. ANALYSIS AND DISCUSSION OF RESULTS

This study aimed to compare the performance of boosted perceptrons (AdaBoost) and support vector machines (SVM) for classification tasks under varying data conditions. First, we analyze the performance of our boosted perceptron on the Banana dataset. As observed, the test accuracy closely follows the training accuracy. However, at some point, the test accuracy plateaus around 200 learners. Mease and Wyner [2008] argued that AdaBoost should be run for a long time until it converges. In our case, it appears to have converged earlier, suggesting that further training beyond this point may have been unnecessary.

For the Splice dataset, the training accuracy increases significantly compared to the test accuracy. Initially, we considered the possibility of class imbalance as a contributing factor. However, this was ruled out since SMOTE was applied to balance the class distributions, and the dataset was properly partitioned using `train_test_split`. Notably, while the training accuracy continues to rise, albeit in small increments, the test accuracy also shows gradual improvement. This indicates that increasing the number of learners further might enhance test set performance. Examining the training and testing times, we observe that training time scales linearly with the number of learners, while testing time remains relatively constant. This behavior aligns with the complexity analysis of AdaBoost, which is $O(Kf)$, where K is the number of learners and f is the runtime of the weak learner [Wang et al., 2019]. Although the plot does not show a significant increase in test time, the ensemble classifier's growing size requires more α_t and h_t evaluations as the number of learners increases. Nevertheless, modern machines handle this computational demand with ease.

Moving on to SVM, we refer to the optimal parameters in Table 3. The primary difference between the models lies in the C and gamma parameters. To understand this, we must first explore what these parameters represent. In SVM, kernels are mathematical functions that transform input data into a higher-dimensional space, enabling linear separation of classes. Common kernel types include linear, polynomial, radial basis function (RBF), and sigmoid. The kernel choice determines the SVM's ability to capture data patterns. Gamma, specific to RBF and polynomial kernels, controls the influence of individual training examples: smaller gamma values lead to a broader influence, while larger values localize influence, potentially causing overfitting. The C parameter acts as a regularization term, balancing the trade-off between maximizing the margin and minimizing classification errors. A high C value emphasizes minimizing errors, risking overfitting, while a low C promotes generalization by prioritizing a larger margin. For polynomial kernels, the degree parameter determines the complexity of the polynomial, affecting its capacity to model non-linear relationships [Cortes and Vapnik, 1995, Scholkopf and Smola, 2001].

The differences in C and gamma between the Banana and Splice datasets reflect the distinct characteristics of each dataset. The C parameter for the Banana dataset is set to a higher value (100), indicating a preference for closely fitting the training data, likely due to greater noise or a more complex decision boundary. In contrast, the Splice dataset uses a lower C value (10), focusing on maintaining a larger margin, which better captures patterns in a less noisy or more structured dataset. Similarly, the gamma parameter is set to "scale" for Banana, adapting gamma based on feature variance, making it suitable for data with moderate scaling and variability. For Splice, gamma is set to "auto," ignoring variance, likely due to more uniform feature distributions. Other parameters, such as the degree and kernel, remain unchanged since the RBF kernel is effective for non-linear problems in both datasets, and the degree parameter does not affect RBF kernel behavior. These differences demonstrate how SVM parameters are tailored to align with the specific properties of each dataset.

When comparing the results between SVM and AdaBoost, as seen in Table 4 and Table 5, it is clear that SVM consistently outperforms AdaBoost across both datasets, although AdaBoost is not far behind. This performance gap can be attributed to the strengths of SVM with an RBF (Radial Basis Function) kernel, which tends to outperform AdaBoost in scenarios where the data exhibits complex, non-linear relationships. The RBF kernel transforms the input data into a higher-dimensional space, allowing SVM to create a linear decision boundary that separates classes which may not be linearly separable in the original space. This ability to model non-linear decision boundaries provides SVM with an advantage over AdaBoost, which relies on a series of weak learners and may struggle to capture such complexities.

In addition, SVM is more robust to noise compared to AdaBoost. SVM focuses on maximizing the margin between classes, with the decision boundary being influenced primarily by the support vectors, thereby minimizing the impact of noisy or outlier data points. In contrast, AdaBoost increases the weights of misclassified samples during training, which can lead to overfitting, especially when noise is present in the data. Another key advantage of SVM is its regularization capabilities. The C and γ parameters in SVM-RBF offer explicit control over generalization, enabling fine-tuning to balance underfitting and overfitting. AdaBoost, however, lacks a comparable mechanism, and its performance is heavily reliant on the choice and strength of the weak learners, which may not generalize well to unseen data.

SVM-RBF also performs well in high-dimensional spaces and with sparse data. The kernel function allows SVM to create flexible decision boundaries that adapt to the complexity of the data, whereas AdaBoost's performance can degrade in high-dimensional settings unless the weak learners are specifically tailored to handle such data. Furthermore, SVM-RBF excels when dealing with class overlaps, as the kernel enables the creation of intricate decision boundaries that can better separate the classes. AdaBoost, on the other hand, may struggle with this, particularly when using weak learners like the perceptron in our experiment.

This superiority of SVM-RBF over AdaBoost has been empirically supported by studies, such as the one conducted by Rahman et al. [2015], which found that SVM-RBF outperforms AdaBoost, particularly in the case of imbalanced datasets.

6. CONCLUSION

In conclusion, the performance of AdaBoost with varying numbers of learners reveals important insights into the behavior and effectiveness of the algorithm. As the number of learners increases, AdaBoost generally improves in terms of training accuracy, though this improvement may plateau or slow down after a certain point. In particular, the test accuracy tends to stabilize, indicating that further increases in the number of learners might not yield significant gains. This plateau could suggest that the model has already reached a point of convergence, where additional learners no longer improve generalization and might even contribute to overfitting, especially if the data is noisy or if there is class imbalance.

As for SVM, this study highlights the strengths and limitations of both SVM with an RBF kernel and AdaBoost for classification tasks. While AdaBoost demonstrates strong performance, SVM-RBF consistently outperforms it, particularly in scenarios where the data exhibits complex, non-linear relationships, noisy elements, or high-dimensional features. The RBF kernel's ability to map data into a higher-dimensional space, combined with SVM's focus on maximizing the margin between classes, allows it to capture intricate decision boundaries and generalize better to unseen data. Additionally, SVM's robustness to noise and the explicit regularization controls provided by the C and γ parameters further enhance its performance. On the other hand, AdaBoost, while effective in certain contexts, can be more susceptible to overfitting, especially in noisy or imbalanced datasets, due to its reliance on weak learners and iterative weight adjustments for misclassified points.

6.1 Recommendations

Since SVM is already well-optimized in libraries like `sklearn`, and the best parameters can be effectively selected using `GridSearchCV`, the focus of the recommendations section should be on the implementation of AdaBoost. The most commonly used base estimator for AdaBoost is likely the decision tree, making it worthwhile to compare the performance of boosted perceptrons with decision trees. Additionally, an interesting approach discussed by the user Sherf [2023] in a Cross Validated post—an online statistics forum—replaces decision trees with weighted logistic regression for AdaBoost, achieving an error rate of around 0.5%. Exploring this alternative could potentially lead to further improvements in AdaBoost's performance.

References

C. Cortes and V. Vapnik. Support vector networks. *Machine Learning*, 20:273–297, 1995.

David Mease and Abraham Wyner. Evidence contrary to the statistical view of boosting. *J. Mach. Learn. Res.*, 9: 131–156, June 2008. ISSN 1532-4435.

Hezlin Aryani Abd Rahman, Yap Bee Wah, Haibo He, and Awang Bulgiba. Comparisons of adaboost, knn, svm and logistic regression in classification of imbalanced dataset. In Michael W. Berry, Azlinah Mohamed, and Bee Wah Yap, editors, *Soft Computing in Data Science*, pages 54–64, Singapore, 2015. Springer Singapore. ISBN 978-981-287-936-3.

Robert E Schapire. Explaining adaboost. In *Empirical inference*, pages 37–52. Springer, 2013.

Bernhard Scholkopf and Alexander J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, Cambridge, MA, USA, 2001. ISBN 0262194759.

Oran Sherf. Improve adaboost that using weighted logistic regression instead of decision trees. Cross Validated, 2023. URL `https://stats.stackexchange.com/q/483796`. URL:https://stats.stackexchange.com/q/483796 (version: 2020-08-19).

Feng Wang, Dingde Jiang, Hong Wen, and Houbing Song. Adaboost-based security level classification of mobile intelligent terminals. *The Journal of Supercomputing*, 75, 11 2019. doi: 10.1007/s11227-019-02954-y.