Adaboost

Lloyd Beaufils, Jerry Bonnell, Gururaj Shriram

November 24, 2017

1 Task

Implement a basic version of *Adaboost*. The number of voting classifiers is determined by a user-set constant. Another user-set constant specifies the number of examples in each training set, T_i. The weights of the individual classifiers are obtained with the help of *perceptron learning*. Apply the program task to some of the benchmark domains from the UCI repository. Make observations about this program's performance on different data. For each domain, plot a graph showing how the overall accuracy of the resulting classifier depends on the number of subclassifiers. Also, observe how the error rate on the training set and the error rate on the testing set tend to converge with the growing number of classifiers.

2 Introduction

Adaboost is a machine learning meta-algorithm. It is a boosting algorithm derived from Schapires boosting designed to minimize randomness and error. Schapires Boosting seeks to minimize the randomness found in bagging, but it is often too strict and restrictive, and thus it is not often possible to find enough examples that satisfy its requirements. Adaboost takes the core idea of Schapires boosting but examples are chosen probabilistically, allowing for a greater ease of selecting examples for the individual classifiers.

Adaboost creates classifiers one at a time, inducing each from a different training subset whose composition depends on the behavior of previous classifiers it has created. The training examples used are decided probabilistically as those on which the old classifiers would not perform well. Each example in the training set has a certain probability of being chosen, and those probabilities are altered with each induced classifier (examples that are repeatedly misclassified receive higher and higher probabilities). Another change from Schapires boosting is that Adaboost does not stop at three classifiers- it induces a great many classifiers and makes the final decision with weighted majority voting between them.

3 Method

For this project, we used two k-NN classifiers written in Python to examine our datasets. The first is our own algorithm which was implemented using Table 3.2 in [3]. The second is imported from SciKit-learn, a popular machine learning library that has implemented many of the classifiers present in the textbook [4]. This additional step is a cross-check between our "simple" classifier and one implemented by experts, i.e. Scikit-learn. If the classifiers perform similarly, then we expect our findings to be reliable.

Irrelevant attributes are added in groups of two, with an initial control group of zero irrelevant attributes. To add irrelevant attributes, we used a normal distribution where $\mu = 10$ and $\sigma = 1$. The purpose of using this distribution, rather than employing a random number generator, is to create values that mimic irrelevant attributes in realistic domains. Akin to circumstances where engineers do not know which attributes are relevant and which are not, we strive to simulate similar conditions by adding irrelevant attribute values that "act" like any other attribute. The value of μ and σ are gradually increased as groups of irrelevant attributes are added. This resulted in spreading out the values of the irrelevant attributes without dominating the rest of the data with scaling errors. Additional research could compare the impact of irrelevant attribute values created by a random number generator and that of the normal distribution that we used.

Finally, we randomly partitioned the datasets into testing and training sets with a ratio of 3:7, respectively. We employed random subsampling to make certain that our training set was representative of the data, and reported the average of all the testing sets. This will ensure that the results are reliable. The division of training and testing set was done 100 times, yielding 100 testing sets.

We ran these classifiers on two separate datasets – the Iris dataset from the UCI repository and a synthetic Animal dataset. Finally, we examined the Wine dataset from the UCI repository to verify that it is the addition of irrelevant, not *relevant*, attributes that lead to poor performance. The results of our experiments are summarized below.

4 Iris Dataset

We obtained the Iris dataset from the UCI machine learning repository [1]. The attributes are `sepal` and `petal` length and width, and the examples are classified into one of three species of iris. This domain is well-known for its sheer simplicity, and this aids us in demonstrating our findings. However, we are keen to note that `sepal width` is removed from the dataset because of its negative correlation to the data. As required by the task, our goal is to guarantee datasets that are as clean as possible.

With no irrelevant attributes, the k-NN classifiers (with k ranging from 1 to 9) all classify randomly-partitioned testing sets with at least 95% accuracy. Our data clearly show that adding irrelevant attributes immediately worsens our accuracy rate, plummeting about twenty points when two irrelevant attributes are added to the dataset. This downward trend continues all the way until the accuracy rate is around an abysmal 30% with 18 irrelevant attributes. This is the case for both k-NN classifiers, with varied values for k. The error rate is over 66%, indicating that having irrelevant attributes makes the classifier do worse than random. The irrelevant attributes actively mislead the learner's decision-making, thus yielding a classifier that does worse than a class determined by a fair dice roll. The sharpest drop in accuracy occurs with the addition of the first 6 irrelevant attributes. This makes sense because the amount of bad values overshadows the amount of good values in the distance formula (3 relevant attributes), so at this point, the accuracy rate reaches and dips just below 40%, which is marginally worse than random.

The two classifiers, ours and Scikit, performed similarly with slight differences. Their output is visualized in Figure 1. For the k=1 case, the behavior of the two algorithms is identical. This is reassuring. However, differences arise when $k > 0$. We determine this difference to be in how we break ties that arise from having more than two possible classes. When determining the class of an attribute vector, our implementation sorts class labels by the number of nearest neighbors for that class. We then randomly choose a class from the subset of classes with the highest frequency of nearest neighbors. This randomness leads to minor differences between the two algorithms.

5 Animal Dataset

The Animal dataset yielded similar results to the Iris dataset. We generated this data programmatically, specifying ranges for valid attributes and generating random numbers within those ranges. Ranges were chosen such that they correlated with the 5 selected classes, each expressing a different animal. But they were also chosen so that a small degree of overlap would occur between the classes. Otherwise, our dataset is subject to overestimating performance. The following table summarizes a snapshot of the synthetic dataset:

Height	**Body Width**	**Class**
14.27	11.43	Giraffe
12.8	11.25	Giraffe
6.2	8.53	Elephant
7.48	13.33	Elephant
4.19	4.86	Tiger
3.63	6.55	Tiger
0.68	1.3	Rabbit
0.7	1.99	Rabbit
1.82	4.25	Dog
2.6	3.69	Dog

The k-NN classifiers do a slightly poorer job with no irrelevant attributes when compared to those classifying the Iris dataset. With only relevant attributes, both classifiers average about 91% accuracy across varying values for k. The higher the k value, the more accurate the classifier is. Again, as with the Iris dataset, the classifiers decrease in accuracy with the addition of irrelevant attributes. However, for this dataset, the decline is steadier and more gradual. The accuracy rate yields around 25%, which is marginally lower than the performance on the Iris dataset. However, this makes sense since there are more classes (5 in the Animals dataset vs 3 in the Iris dataset), so when the classification approaches random selection, there is a lower chance that any specific class is selected. This situation is depicted in Figure 5, with respect to the Iris dataset.

6 Wine Dataset

To verify that the decrease in accuracy is a result of adding irrelevant attributes and not a result of simply adding attributes in general, we performed an additional test on the Wine dataset from the UCI repository [2]. This dataset contains 13 attributes, two of which were found to negatively contribute to the classifier's performance: `magnesium` and `total phenols`. These were subsequently removed from our dataset. Our test was as follows. First, we selected only 2 of the remaining 11 attributes and ran our k-NN classifier on it. Then, we repeated this process, adding two of the valid attributes at a time until 10 of the attributes were used. This stands in contrast to our previous tests, as no irrelevant attributes were introduced. What we found was as expected – the accuracy did not decrease as a result of adding more attributes. Instead, the accuracy maintained status quo or performed better as a result of more relevant attributes. This finding gives added weight to the theory that irrelevant attributes are responsible for the increased error rate in the k-NN classifiers. Our findings are noted in Figure 6.

7 Conclusion

We conclude that irrelevant attributes are detrimental to the performance of Nearest Neighbor classifiers. Intuition agrees and our experimental results are further evidence to this claim. The irrelevant attributes add terms to the distance formula that end up obfuscating the result and cause the classifier to deteriorate to a technique that performs worse than random. We tested two classifiers on two datasets and reached the same conclusion. Furthermore, we tested relevant attributes to see if their addition would net a similar negative effect. As our findings show, it did not. The cause of the decrease in accuracy is irrelevant attributes, and these tests verify how dangerous they can be. To reduce a classifier from near-perfect accuracy to worse than random performance is an engineering issue that should not be taken lightly. Great care must be given to ensure that the ratio of relevant to irrelevant attributes remains high in the dataset; otherwise, the results of the Nearest Neighbor classifiers are prone to disappointment.

References

[1] Iris dataset. URL: `https://archive.ics.uci.edu/ml/datasets/iris`.

[2] Wine dataset. URL: `https://archive.ics.uci.edu/ml/datasets/Wine`.

[3] Miroslav Kubat. *An Introduction to Machine Learning*, chapter Nearest Neighbor Classifiers. Springer International, second edition, 2017.

[4] Scikit-learn. Nearest neighbors classification. URL: `http://scikit-learn.org/stable/auto_examples/neighbors/plot_classification.html`.

Figure 1: These two graphs show the performance of our classifier against the Scikit k-NN classifier. They behave identically for $k = 1$ but begin to vary at k-values higher than one.

Figure 2: This shows our classifier with all values of k we tested on the Iris dataset. For all classifiers, the steepest drop in accuracy is with the introduction of the first 6 irrelevant attributes, after which our classifier converges at around 30% accuracy.

Figure 3: Irrelevant attributes again are shown to decrease the accuracy rate for all values of k in our classifier. Here, the descent is a bit more gradual, but once more the accuracy ends up at around 30%.

Figure 4: The first color map shows where an example would have to land in the Iris dataset to be classified as one of the three classes. This map uses two relevant attributes on the axes. The latter two color maps show that as the ranges of irrelevant attributes increase, the classification suffers as there is no correlation between an example's classification and the example's irrelevant attribute value.

Figure 5: This graph compares the performance of our 1-NN classifier on both datasets. It demonstrates how the irrelevant attributes cause a sharper decline in the Iris dataset and how they decrease the classifiers accuracy on both datasets.

Figure 6: This graph shows our 3-NN on the Wine dataset. It demonstrates that the lower accuracy demonstrated in all other graphs is not merely a result of additional attributes, but of specifically irrelevant attributes. Here we add relevant attributes two at a time and the accuracy is improved.