Brad Pershon  
Data Mining  
Course Project Report

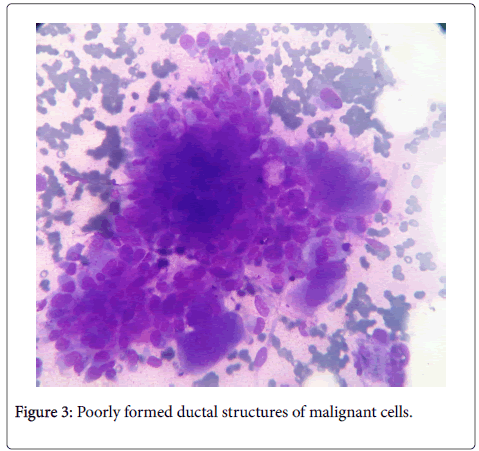
**Introduction**

Can we determine if a tumor is benign or malignant from a digitized image? That is the question we are attempting to answer by analyzing the Breast Cancer Wisconsin dataset. Our project will utilize an ensemble learner to determine if measurements taken from digital images can indeed determine tumor status. Our model will examine 30 attributes and use the Adaboost algorithm outlined in class.

The purpose of our final project was to implement the Ensemble Classifier Adaboost. The Adaboost algorithm combines a collection of unstable learners to improve performance over a single base classifier. It achieves this increased performance by manipulating the training data set which leads to a reduction in variance. Our Adaboost implementation can use either a Decision Tree base classifier or a K-Nearest Neighbors classifier. In our report we attempt to determine if one base classifier outperforms the other within the Adaboost framework.

**Data Set**

The Breast Cancer Wisconsin (Diagnostic) Data Set was obtained from the UCI Machine Learning Repository. As stated on the repositories description, the data set consist of “features computed from a digitized image of a fine needle aspirate of a breast mass. They describe characteristics of the cell nuclei present in the image.” [1] There are 30 measurement features, a record ID, and a class label.

  
*Figure 1: Digitized image of a fine needle aspirate of a breast mass. [2]*

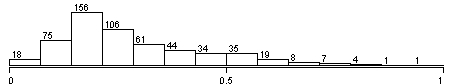
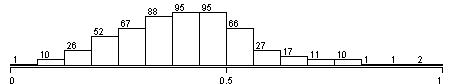
The data set has 569 records split into 2 classes: Benign and Malignant. The malignant class makes up 37.3% of the population at 212 records, while the benign class is 62.7% of the population at 357 records.

*Figure 2: Class distribution of the Wisconsin Breast Cancer data set.*

*Figure 3: Parallel coordinates plot for the first 5 Benign and Malignant records.*

To help visualize the data we created a parallel coordinates plot of the first 5 records belonging to each class. The first 4 features of each record are related to the average size of the growth, based on the graph one might conclude that on average malignant growths are smaller than benign growths. This is a very limited sample size, but it does give a small insight into what might happen in our data.

The data set was transformed during our preprocessing stage to aid our implementation. First the class labels were altered from B and M to 1 and -1 respectively. All 30 measurement features were normalized, and the ID field was removed as it had no influence on the class labels.

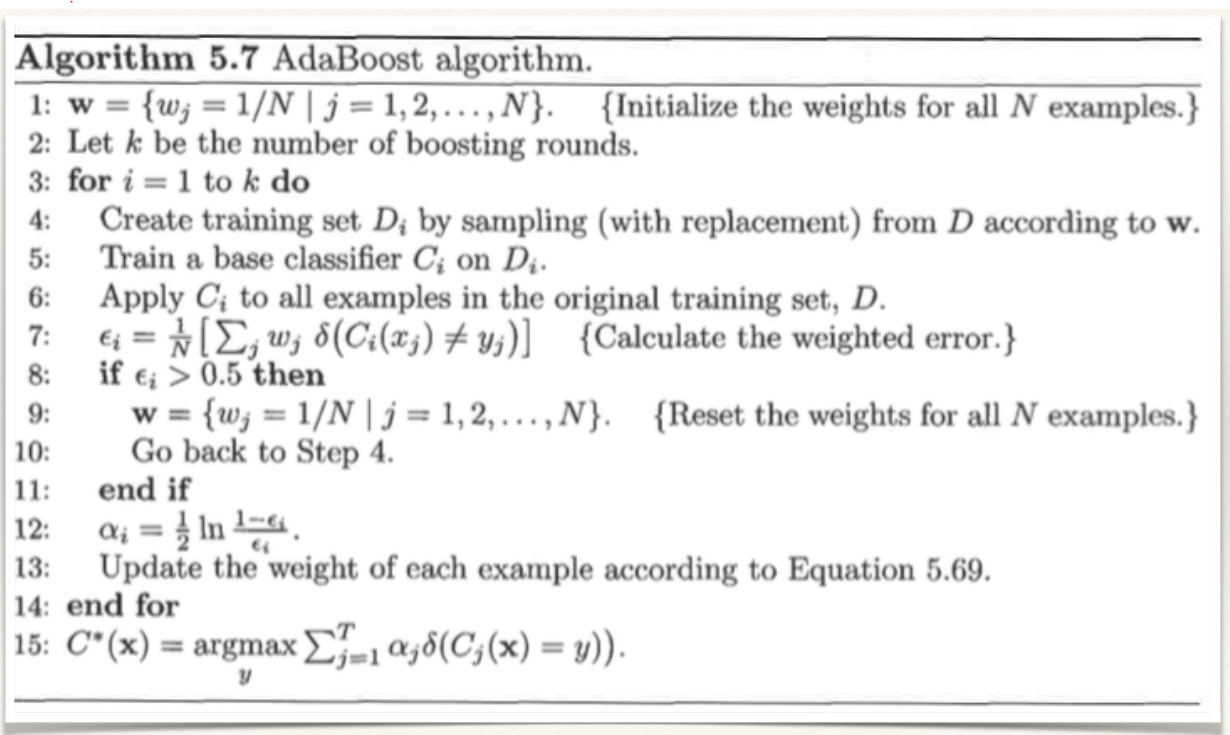
*Figure 5: Smoothness\_worst distribution*

*Figure 4: Peremeter\_worst distribution*

Figure 4 and 5 show the distribution of two features after normalization. Many of the feature distributions appear to be almost Gaussian with a right tail. While there did appear to be a small handful of outliers, we opted to leave the records as is. The data set had no missing values, if in the future we were provided a similar data set with missing values, we would opt to replace those values with the mean of the feature from samples of the same class.

**Implementation**

Our implementation of the Adaboost algorithm was built using Python and the base classifiers come from the scikit-learn library. The application prompts the user to enter the number of boosting rounds and the type of base classifier. The classifier options are scikit-learn’s decision tree or K-Nearest Neighbor with K set to 7. We fixed the value of K for the project as we were interested in the performance of the algorithm in relation to the number of boosting rounds. Once we have the users input we follow the general Adaboost algorithm described below:

*Figure 6: The general Adaboost algorithm. [3]*

At the start all training are assigned an equal weight, this weight is used as the probability that the record will be selected in step 4 of the algorithm. Once assignment is complete we begin the boosting rounds. First we generate a new training set by sampling with replacement from the original training set according to the assigned weights. Next the base classifier is trained using the newly generated training set, either scikit-learn’s decision tree or KNN classifier depending on the users input. Once training is complete, we test the classifier on the complete training set. If the classifier performs worse than 50% on the weighted error function we reset the weights and begin again, otherwise we compute an alpha value for the classifier and update each weight according to the following formula:

In simple terms we increase the weight of an incorrectly classified record, and decrease the weight of a correctly classified record. This increases the likelihood that misclassified records will be present in future training sets, and that future classifiers will be trained to correctly classify them. This continues until we reach the end of the user specified number of boosting rounds.

At the end we have a set of classifiers who performed no worse than 50% weighted error for their given training set and their corresponding alpha values. To compute the ensemble learner’s prediction we use the formula described in step 15 of figure 6, we allow each classifier to predict the class of the record and multiply the prediction with their alpha value. We then sum the weighted predications and look at the sign of the sum. If it is positive we assign it to the benign class otherwise we assign it to the malignant class.

In our implementation the main.py file contains the data preprocessing, obtains user input, and loops through the Adaboost algorithm described above. The main file also generates the training and testing sets from the original data set. The percentage of records in the training set can be set by the training\_size\_perc variable found at the top of the file. The records are shuffled and then randomly split into the two sets. The adaboost.py file contains the ensemble learner class and functions to set training examples, test classifiers, and predict a records class. The util.py file contains a function to generate a new training set give a list of weights and a function to output the results.

**Results**

To test our implementation we split the data into a training set containing 70% of total records and a test set containing the remaining 30% of total records. Because our implementation shuffles the records then randomly assigns each record to a set, we had to average our results over 10 runs per boosting round, as there can be slight fluctuations in performance values based on which records are split into the training and test sets.

First we tested the decision tree based approach vs the k-nearest neighbor approach, the results are shown in table 1 and table 2.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Adaboost-Decision Trees | | | | |
| Boosting Rounds | Accuracy | Precision | Recall | F-Measure |
| 1 | 0.912 | 0.953 | 0.911 | 0.932 |
| 2 | 0.912 | 0.904 | 0.949 | 0.926 |
| 3 | 0.929 | 0.919 | 0.971 | 0.944 |
| 4 | 0.941 | 0.934 | 0.971 | 0.952 |
| 5 | 0.947 | 0.924 | 0.990 | 0.956 |
| 6 | 0.969 | 0.961 | 0.980 | 0.976 |
| 7 | 0.959 | 0.971 | 0.962 | 0.966 |
| 8 | 0.965 | 0.971 | 0.971 | 0.971 |
| 9 | 0.959 | 0.961 | 0.971 | 0.966 |
| 10 | 0.959 | 0.952 | 0.980 | 0.966 |

*Table 1: Decision tree based Adaboost results*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Adaboost-K-Nearest Neighbors | | | | |
| Boosting Rounds | Accuracy | Precision | Recall | F-Measure |
| 1 | 0.959 | 0.945 | 0.990 | 0.967 |
| 2 | 0.959 | 0.970 | 0.961 | 0.966 |
| 3 | 0.918 | 0.943 | 0.926 | 0.935 |
| 4 | 0.947 | 0.924 | 0.990 | 0.956 |
| 5 | 0.906 | 0.942 | 0.907 | 0.925 |
| 6 | 0.947 | 0.938 | 0.981 | 0.959 |
| 7 | 0.924 | 0.962 | 0.917 | 0.939 |
| 8 | 0.924 | 0.940 | 0.931 | 0.935 |
| 9 | 0.935 | 0.972 | 0.929 | 0.950 |
| 10 | 0.929 | 0.935 | 0.966 | 0.950 |

*Table 2: KNN based Adaboost results*

The decision tree based Adaboost performed better than the KNN based approach overall. It had the highest average accuracy at 96.9% with 6 boosting rounds, which was a full percentage point higher than the KNN’s best 95.9% with 1-2 boosting rounds. The decision tree based model had an average accuracy of 94.5% over all boosting rounds, while the KNN model had an average of 93.5%. The precision values between the two approaches were similar. The KNN model had an average over all rounds of 94.7% beating the decision tree model’s 94.5%. The decision tree model had a higher average recall at 96.6% and f-measure at 95.5% than the KNN based model with 95% and 94.8% respectively.

*Figure 7: Plot of the decision tree performance measures*

*Figure 8: Plot of the KNN performance measures*

The most interesting thing about the results outlined above in table 1 and 2 are the trends in the performance measurements. Figure 7 and 8 show plots of the performance measures over the boosting rounds. The decision tree based ensemble learner behaves as you would expect, as you increase the number of boosting rounds the performance increases up until about the 6th boosting round, and then flattens out. The KNN based ensemble learner is all over the place. Its accuracy is highest during the 1st and 2nd boosting rounds. The inverse proportionality of recall and precision are clearly evident, as one spikes up the other spikes down. There is no point at which the model stabilizes.

Next we tested an off the shelf implementation of Adaboost from Weka. Again we tested a decision tree based approach using the Weka J48 implementation, and a KNN based approach using the Weka lBk implementation. The results are shown in table 3 and 4.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Weka-Decision Trees | | | | |
| Boosting Rounds | Accuracy | Precision | Recall | F-Measure |
| 1 | 0.936 | 0.936 | 0.962 | 0.949 |
| 2 | 0.936 | 0.936 | 0.962 | 0.949 |
| 3 | 0.960 | 0.971 | 0.962 | 0.967 |
| 4 | 0.953 | 0.954 | 0.972 | 0.963 |
| 5 | 0.960 | 0.971 | 0.962 | 0.967 |
| 6 | 0.936 | 0.952 | 0.943 | 0.948 |
| 7 | 0.930 | 0.943 | 0.943 | 0.943 |
| 8 | 0.930 | 0.943 | 0.943 | 0.943 |
| 9 | 0.936 | 0.952 | 0.943 | 0.948 |
| 10 | 0.936 | 0.944 | 0.953 | 0.945 |

*Table 3: Weka-Adaboost DT based performance measures*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Weka-K-Nearest Neighbors | | | | |
| Boosting Rounds | Accuracy | Precision | Recall | F-Measure |
| 1 | 0.971 | 0.963 | 0.991 | 0.977 |
| 2 | 0.971 | 0.963 | 0.991 | 0.977 |
| 3 | 0.942 | 0.953 | 0.953 | 0.953 |
| 4 | 0.953 | 0.971 | 0.953 | 0.962 |
| 5 | 0.953 | 0.971 | 0.953 | 0.962 |
| 6 | 0.953 | 0.971 | 0.953 | 0.962 |
| 7 | 0.953 | 0.971 | 0.953 | 0.962 |
| 8 | 0.953 | 0.971 | 0.953 | 0.962 |
| 9 | 0.953 | 0.971 | 0.953 | 0.962 |
| 10 | 0.953 | 0.971 | 0.953 | 0.962 |

*Table 4: Weka-Adaboost KNN based performance measures*

The Weka Adaboost implementations performed opposite to our results In terms of which base classifier model performed better. The KNN based Adaboost implementation outperformed the decision tree based model with an average accuracy of 95.6%, average precision of 96.8%, average recall of 96.1%, and an average f-measure of 96.4%. The decision tree based Adaboost implementation had an average accuracy of 94.1%, average precision of 95%, average recall of 95.5%, and an average f-measure of 95.2%. The value of K was fixed to 7 for the KNN based implementation, as it was in our implementation, for testing purposes.

*Figure 9: Plot of the Weka DT performance measures*

Figure 10: Plot of the Weka KNN performance measures

Once again the performance trends in the decision based model follow expectations, as we increase the number of boosting rounds the performance measures increase. Here they decline slightly and then begin to level off, giving a plot which looks like a local maxima. The Weka KNN plot in figure 10 was surprising, the models best performance were once again found at the lower boosting rounds. The results then dip, and after the 4th boosting round they remain completely static.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Model** | **Adaboost-DT** | **Adaboost-KNN** | **Weka-Adaboost-DT** | **Weka-Adaboost-KNN** |
| Accuracy | 0.945 | 0.935 | 0.941 | 0.956 |
| Precision | 0.945 | 0.947 | 0.950 | 0.968 |
| Recall | 0.966 | 0.950 | 0.955 | 0.961 |
| F-Measure | 0.955 | 0.948 | 0.952 | 0.964 |

*Table 5: Summary of the average performance measures for all models tested*

Overall our decision tree based implementation outperformed the Weka decision tree based implementation. Our model had a higher average accuracy, average recall, and average f-measure. The Weka KNN based implementation outperformed our KNN implementation in every category. The best model on average was the Weka Adaboost implementation using a KNN base classifier with k = 7 and a boosting round of either 1 or 2.

**Conclusion**

The project was a success overall, we were able to implement the Adaboost algorithm based on two different classifiers and then compare our results against an off the shelf implementation from Weka. So can we determine if a tumor is benign or malignant from a digitized image? The answer appears to be yes, our ensemble learner models were able to accurately classify the type of tumor between 90.6%-96.9% of the time. Which base classifier was better within the Adaboost framework? In our implementation the decision tree models worked best, but with the Weka Adaboost implementation the KNN models came out ahead.

Given our results I would recommend the Weka Adaboost implementation using a KNN base classifier with k = 7 and a boosting round of either 1 or 2. Our decision tree model came with 0.1% of the best model in terms of accuracy, which was an off the shelf implementation, for our first attempt at implementing the Adaboost algorithm we felt it was successful.

It was unclear how the Weka Adaboost implementations were selecting their training and test sets. With their software package you are able to select a percentage of your total data set for the models training, and the remainder is set aside for testing. Retraining the model with the same parameters always returned the same values for the performance measures. Their sampling procedure might be a stratified implementation, which may generate better training sets then our implementation. We choose to merely shuffle the complete data set once before the boosting rounds, and then build the training sets.

At first the trend in both KNN based ensemble learners confused me, where performance measures in early boosting rounds outperformed later boosting rounds. Upon further review this makes sense as we increase misclassified samples weight’s over the course of boosting rounds, later classifiers in the ensemble will have the same problem due to the fixed value of K. The training samples will have a high amount of misclassified samples, many with the same K neighbors that caused the misclassification in the first place. One potential solution to this problem would be to reduce the value of K when the performance of the ensemble stabilizes.

If this project were to continue we would implement a better sampling procedure based on stratified sampling. We would also implement a dynamic K value for the KNN base classifier for later boosting rounds if the Ensemble performance had stabilized as in figure 10. Overall we feel like our implementation was a success as we approach the performance measures of the off the shelf implementation.

**References**

[1] (n.d.). Retrieved November 17, 2017, from https://archive.ics.uci.edu/ml/datasets/Breast Cancer Wisconsin %28Diagnostic%29

[2] Ticku, A., & Singh, P. (2016, January 11). Metaplastic Carcinoma of Breast with Osteosarcomatous Differentiation: A Rare Case Diagnosed by Fine Needle Aspiration Cytology. Retrieved November 18, 2017, from <https://www.omicsonline.org/open-access/metaplastic-carcinoma-of-breast-with-osteosarcomatous-differentiationa-rare-case-diagnosed-by-fine-needle-aspiration-cytology-jmsp-1000103.php?aid=66801>

[3] Tan, P., Steinbach, M., Karpatne, A., & Kumar, V. (2019). Introduction to data mining. New York, NY: Pearson Education.