# Machine Learning, Fall 2019: Project 4
# out on 10/31/19 - due on 11/14/19 by noon on Blackboard

Anastasija Mensikova

**Header:** Resources used for the completion of the project:

- Python programming language
- Jupyter Notebook
- Scikit-Learn ML Library for Python
- AdaBoost Scikit Documentation:
  https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html
- Lecture slides
- MatPlotLib for plots

## 1 Single-node Decision Tree As Linear Classifier

**(5 points)** What is the difference between the decision boundary for the decision stump linear classifier and general linear classifiers (i.e., logistic regression) you have learned in class in terms of the type of the linear boundary?

A decision stump is not, in fact, a linear model as it may appear at first. However, it is important to note that the decision boundary of a model can be a line, even if the model is not linear (e.g. for logistic regression). Although both decision stumps and general linear classifiers can have a remotely similar-looking linear decision boundary, the models and their predictions are not the same. A decision stump is normally formed as a single axis-aligned split of the input samples, whereas a general linear classifier is a more complex linear function with a more complex, piece-wise linear decision boundary.

**(10 points)** Sketch or plot a small data set of points (no more than 10 2-dimensional points), which is completely separable using a linear classifier but not completely separable using decision stumps. Include in your sketch a linear classifier that completely separates the data. Briefly explain why you cannot construct a decision stump which completely separates the data you provided.

The below Figure 1 provides a rough sketch of dataset as described. That is, a dataset that is completely separable with a linear classifier but not completely separable with decision stumps. Although it is of course possible to attempt to separate this data with a decision stump, as mentioned previously a decision stump is formed as a single axis-aligned split of the sample. Looking at Figure 1, we can see that the given linear classifier $y$ clearly completely separate the data. However, if using a decision stump, no good thresholds would be established to completely separate the data since a lot of points from both classes have overlapping values for both of the features. No matter the number of boosting iterations, no thresholds/stumps could be established to completely separate the data points.

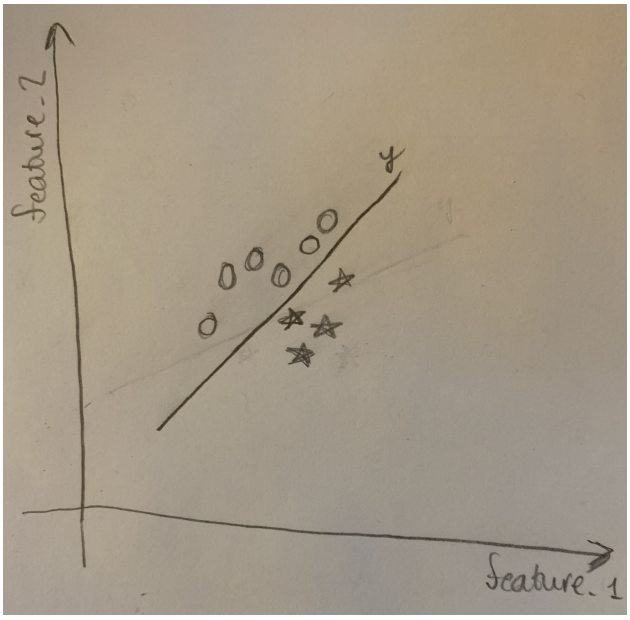


Figure 1: Question 1.2

## 2 General Decision Trees vs. Linear Classifiers

**(10 points)** Decision trees of arbitrary depth can capture more complex decision surfaces. Sketch or plot a small 2D data set which is completely separable using decision trees of arbitrary depth (using decision stumps at each node) but cannot be completely separated using a single linear classifier. Include in your sketch the decision surface obtained by the decision tree.

Figure 2 below portrays a very simple dataset with two main features of age and weight (in kg) that determine whether or not a person has good eye-sight. Eeach sample represents a person. The Figure 2 also contains the decision surface and the decision stumps used in the classification. The vertical and horizontal lines inside the plot represent the stumps and the stars and o's represent people with either good or bad eye-sight, respectively.

**(5 points)** Explain why no linear classifier can completely separate the data. Also, draw the decision tree annotating each node with the decision rule and each edge with True or False (indicating the outcome of the decision rule).

Looking at the data sketched in Figure 2, it is obvious that it cannot be completely separated by a linear classifier as there is no clear linear function that would be able to divide and completely separate all the samples. The relationship between all samples is clearly not linear, and the feature values do not clearly distinguish between people with good and bad eye-sight. Figure 3 shows a sample decision tree drawn for the same dataset, evaluating the values at each stump.

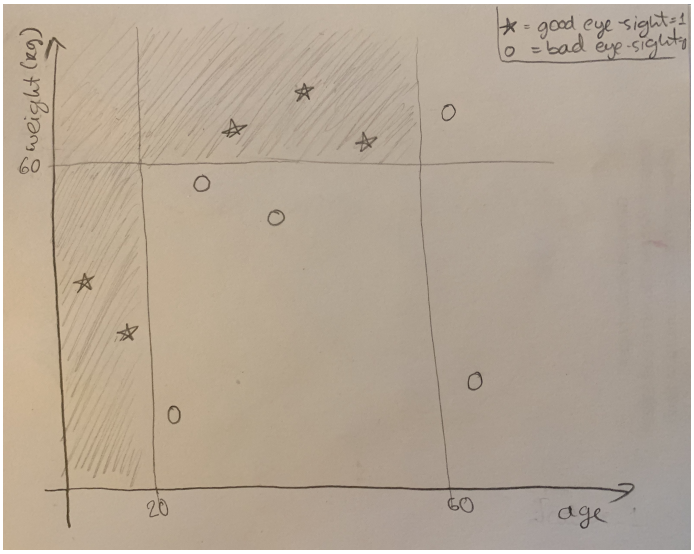


Figure 2: Question 2.1

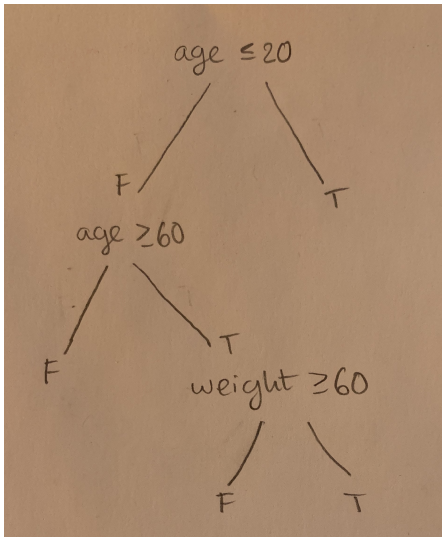


Figure 3: Question 2.2

# 3 Implementation

Given a set of $m$ examples, $(x_i, y_i)$ ($y_i$ is the class label of $x_i$), $i = 1, \ldots, m$, let $h_t(x)$ be the weak classifier obtained at step $t$, and let $\alpha_t$ be its weight. Recall that the final classifier is

$$H(x) = \text{sign}(f(x)), \text{ where } f(x) = \sum_{t=1}^{T} \alpha_t h_t(x).$$

1. Evaluate your AdaBoost implementation on the Bupa Liver Disorder dataset that is available for download at https://drive.google.com/file/d/1I-sYe-EeZYqhqLpOwAQGUya7tWN8iylO/view?usp=sharing. The classification problem is to predict whether an individual has a liver disorder (indicated by the selector feature) based on the results of a number of blood tests and levels of alcohol consumption. Use 90% of the dataset for training and 10% for testing. Average your results over 50 random splits of the data into training sets and test sets. Limit the number of boosting iterations to 100.

   In a single plot show:

   - **(15 points)** average training error after each boosting iteration
   - **(15 points)** average test error after each boosting iteration

   An AdaBoost classifier (with Decision Tree Classifier as the base estimator) was trained on the Bupa Liver Disorder Dataset. Figure 4 below shows the error rate for a singular split of the data into 90% (training) and 10% with 100 boosting iterations. Figure 5 below shows the error rate with 50 random splits of data (into 90:10) averaged out for each of the 100 iterations. Both figures show both train and test error. We can see that the general tendency of the error rate is very similar for both charts, with Figure 5 error rates being more stable as they portray the average over 50 splits. Both train and test error seem to be decreasing as the number of estimators (or iterations) increases, with the train error having a very stable decrease. However, looking at Figure 5, it is obvious that the test error seems to be at its lowest between 30 and 50 iterations, which might be ideal for this dataset for future use. The details of the implementation, as always, can be found in the attached Jupyter Notebook.

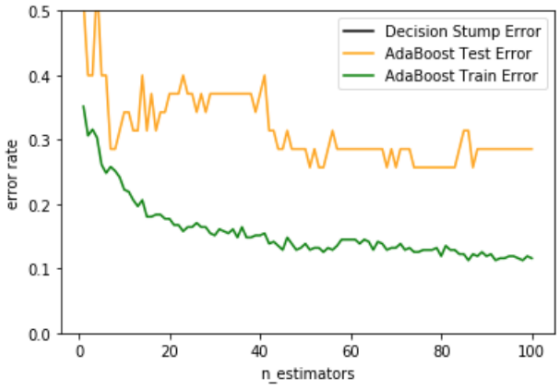


Figure 4: Question 3.1 - 1 split

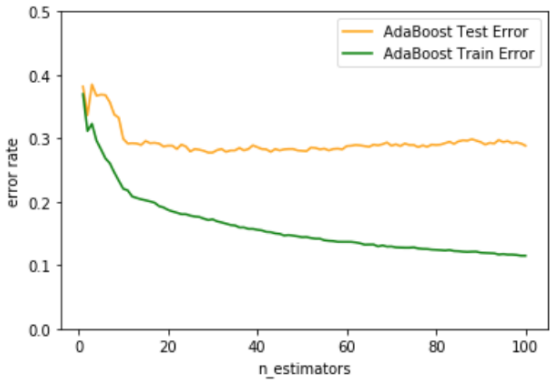


Figure 5: Question 3.1 - 50 random splits

2. **(20 points)** Using all of the data for training, display the selected feature component $j$, threshold $c$, and class label $C_1$ of the decision stump $h_t(\mathbf{x})$ used in each of the first 10 boosting iterations ($t = 1, 2, ..., 10$)

   The following are the selected (top) feature components, their labels and their thresholds per each one of the 10 boosting iterations:

   I Component gammagt threshhold: 0.05
   II Component sgpt threshhold: 0.12
   III Component mcv threshhold: 0.59
   IV Component gammagt threshhold: 0.10
   V Component alkphos threshhold: 0.37
   VI Component gammagt threshhold: 0.01
   VII Component drinks threshhold: 0.28
   VIII Component sgot threshhold: 0.25
   IX Component sgpt threshhold: 0.15
   X Component drinks threshhold: 0.04

   More details on how those were calculated can be found in the Jupyter Notebook attached.

   Figure 6 below is a general exploration of the most "important" features in this dataset.

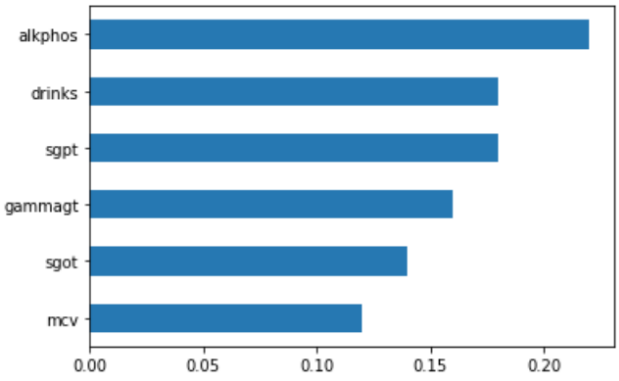


Figure 6: Question 3.2

3. **(20 points)** Using all of the data for training, in a single plot, show the empirical cumulative distribution functions of the margins $y_i f_T(\mathbf{x}_i)$ after 10, 50 and 100 iterations respectively, where $f_T(\mathbf{x}) = \sum_{t=1}^{T} \alpha_t h_t(\mathbf{x})$. Notice that in this problem, before calculating $f_T(\mathbf{x})$, you should normalize the $\alpha_t$s so that $\sum_{t=1}^{T} \alpha_t = 1$. This is to ensure that the margins are between -1 and 1.

Figure 7 portrays the ECDF (empirical cumulative distribution functions) for 10, 50, and 100 iterations respectively. We can see that the distribution when having 50 and 100 distributions is quite similar and quite smooth and even. However, the distribution when having 10 iterations is a little more scattered and a little less evenly distributed. All three seem to have a few points of intersection though, and neither of them are drastically different.

The details on how this was calculated, as well as the entire methodology, can be found in the attached Jupyter Notebook.

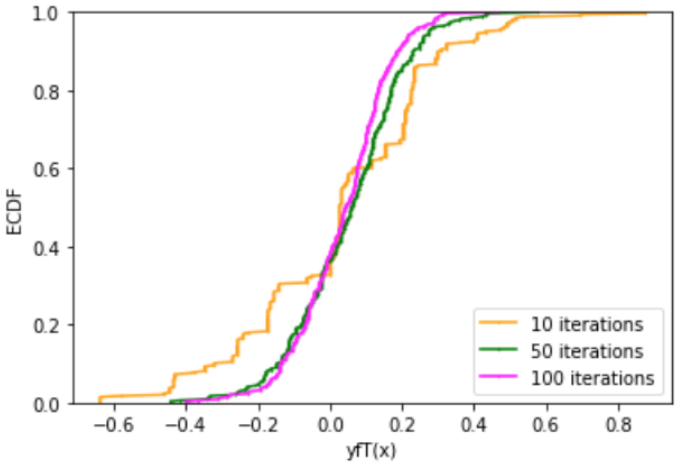


Figure 7: Question 3.3